Understanding Natural Language with Commonsense Knowledge Representation, Reasoning, and Simulation

Antoine Bosselut

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington
2020

Reading Committee:
Yejin Choi, Chair
Gina-Anne Levow
Noah Smith

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Understanding Natural Language with Commonsense
Knowledge Representation, Reasoning, and Simulation

Antoine Bosselut

Chair of the Supervisory Committee:
Associate Professor Yejin Choi
Computer Science and Engineering

For machines to understand language, they must intuitively grasp the *commonsense knowledge* that underlies the situations they encounter in text. A simple statement such as *it is raining* immediately implies a bank of shared context for any human reader: they should bring an umbrella, roads will be slippery, increased traffic may make them late, rain boots are preferable to sandals, and many more. Language understanding systems must be able to robustly use this commonsense knowledge to make decisions or take actions. Observations of the world are always more rich and detailed than the information that is explicitly transmitted through language, and machines must be able to fill in remaining details with commonsense inferences.

Recent advances in natural language processing have made considerable progress in identifying the commonsense implications of situations described in text. These methods generally involve training high-parameter language models on large language corpora and have shown marked improvement on a variety of benchmark end tasks in natural language understanding. However, these systems are brittle – often failing when presented with out-of-distribution inputs – and uninterpretable – incapable of providing insights into why these different inputs cause shifted behavior. Meanwhile, traditional approaches to natural language understanding, which focus on linking language to background knowledge from large ontologies, remain

limited by their inability to scale to the situational diversity expressed through language.

In this dissertation, we argue that for natural language understanding agents to function in less controlled test environments, they must learn to reason more explicitly about the commonsense knowledge underlying textual situations. In furtherance of these goals, we draw from both traditional symbolic and modern neural approaches to natural language understanding. We present four studies on learning commonsense representations from language, and integrating and reasoning about these representations in NLP systems to achieve more robust textual understanding.

Acknowledgements

My completion of this degree would not have been possible without a supportive network of mentors, collaborators, colleagues, friends, and family.

First and foremost, I'd like to thank my advisor, Yejin Choi, for her counsel, support, and feedback over the last six years. Her mentorship and advising were crucial for my growth as a researcher. And her guidance and flexibility allowed me to explore challenging problems in commonsense representation and reasoning for NLP, and learn to develop applicable solutions for them. I'm also incredibly grateful to Asli Çelikyilmaz, a friend and colleague, who advised me throughout my stay as a student researcher at Microsoft, and who motivated my passion for NLP topics in language generation. I'd also like thank Peter Clark and Oren Etzioni, who invited me to form fruitful collaborations at the Allen Institute for AI, and whose interest in my work motivated me to continue exploring the ideas in commonsense representation that became the foundation of my thesis. I am also grateful to my committee members Noah Smith, Gina-Anne Levow, and Kevin Knight, who have been mentors I could turn to for counsel at various stages of my research career.

I'd also like to thank the many great collaborators, co-authors, and colleagues I've had the privilege of working with during my PhD. None of of my work would have been as exciting or well-rounded without insights from Dieter Fox, Hanna Hajishirzi, Hannah Rashkin, Maarten Sap, Lianhui Qin, Saadia Gabriel, Aida Amini, Andrew Hoang, Ari Holtzman, Jan Buys, Max Forbes, Peter West, Omer Levy, Corin Ennis, Elizabeth Clark, David Golub, Bhavana Dalvi, Niket Tandon, Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Vered Shwartz, Kyle Lo, Scott Yih, Jena Hwang, Keisuke Sakaguchi, Xinya Du, Jianfeng Gao, Xiaodong He, Po-sen Huang, Urvashi Khandelwal, Marjan Ghazvininejad, Thomas Wolf, Sasha Rush, and Jianfu Chen. My research agenda was broadened and enhanced by the discussions held with them and

the collaborative projects pursued with them.

I've also been lucky to be part of a large NLP community at the University of Washington. Throughout my PhD, I have been able to interact with and learn from the many great researchers that make up UW NLP, including, but not limited to: Luke Zettlemoyer, Yonatan Bisk, Eunsol Choi, Chloe Kiddon, Ioannis Konstas, Dallas Card, Roy Schwartz, Nicholas Fitzgerald, Sam Thomson, Jesse Thomason, Gabriel Stanovsky, Swabha Swayamdipta, Mark Yatskar, Rowan Zellers, Mike Lewis, Kenton Lee, Luheng He, and Xi Victoria Lin.

One of the most rewarding parts of pursuing a PhD is being able to make friends and form strong bonds with fellow students on the same journey. I'd specifically like to thank Maaz Ahmad, Terra Blevins, Jiechen Chen, Srini Iyer, Kiron Lebeck, Niel Lebeck, Jacob Schreiber, and Dave Wadden for making the successes more meaningful, the failures less embittering, and for the great times between asynchronous conference deadlines.

Finally, I want to thank my family: Anne, François, Marion, Remy, Antoinette, Mengsha, Chloe and Theo. It would take too long to list all the ways you've supported and helped me throughout this endeavor, but I couldn't have done it with you.

DEDICATION

To my parents, Anne and Remy

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Understanding narratives requires reasoning about implicit world knowledge related to the causes, effects, and states of situations described in text. At the core of this challenge are two sub-problems: 1) accessing contextually-relevant knowledge on demand, and 2) reasoning over this knowledge to arrive at an understanding of a situation. For example, the simple narrative "It's going to snow tonight. I'll have to wake up 15 minutes earlier tomorrow," contains two sentences that are seemingly unrelated to one another. And yet, most human readers would be able to understand the underlying conditions that make them coherent. First they might identify implicit pieces of commonsense knowledge about the situation that are unstated, but sensible given their experience in the world:

- snow might fall on a car parked outdoors
- snow might fall on sidewalks
- snow on car windshields will have to be de-iced
- de-icing cars takes time
- slippery slidewalks make people more likely to fall
- to avoid falling, people will walk more slowly
- bosses expect people to come into work on time

Using these facts, a person would then reason over this knowledge to come to one of many possible understandings of the narrative, such as:

- ((snow falls on car $\Rightarrow$ car will need to be de-iced $\Rightarrow$ de-icing car takes time) $\wedge$ boss expects people to be at work on time) $\Rightarrow$ have to wake up earlier

or perhaps:

- ((snow falls on sidewalks $\Rightarrow$ sidewalks will be slippery $\Rightarrow$ people will walk more slowly $\Rightarrow$ it will take longer to walk to work)) $\wedge$ boss expects people to be at work on time) $\Rightarrow$ have to wake up earlier

As illustrated, even this simple narrative requires considerable reasoning abilities to be understood, and, in fact, still cannot be fully resolved without additional information (e.g., does this person drive to work? do they park in a garage?). And yet, narratives such as this one are found throughout written text, as humans are able to rely on learned experience and prior interaction with the world to fill in the gaps needed to understand situations. Machines, however, do not have background knowledge learned through experience and interaction with the world. Instead, they must be given this knowledge directly, or find some other way of learning it. In light of these two options, prior approaches to providing machines with narrative understanding capabilities have generally focused on hand-crafting structured, rule-based systems or learning these capabilities from large-scale unstructured data.

Early approaches for linking written language to corresponding commonsense knowledge generally involved mapping text to symbolic representations (Mccarthy, 1960; Gordon and Hobbs, 2017; Levesque, 2017) that could be used to query pre-constructed knowledge bases of commonsense knowledge (Lenat, 1995; Speer et al., 2017). However, these approaches suffer two drawbacks. First, due to the cost and difficulty of curating commonsense knowledge bases, knowledge bases lack the coverage to handle all situations described in text. As a result, practitioners design noisy and error-prone heuristics to select relevant situational knowledge from the database. Second, because these knowledge bases are often canonicalized, this selection step decontextualizes the concepts described in text, resulting in polysemy challenges and retrieved knowledge that is potentially less relevant to the situation. In the example above, knowing that a “wake" is a vigil for someone who has recently passed is immaterial. As a result of these challenges, symbolic approaches to commonsense reasoning have generally been considered too brittle to scale to the situational diversity needed to understand natural language.

As a result, modern approaches to this problem have generally shifted to using statistical (Manning

and Schütze, 1999) and neural (Goldberg, 2016) models of text that learn relevant knowledge implicitly from training on large-scale data. With the advent of large-scale pretraining of language models (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018), this shift has become even more pronounced. Large, expressive deep neural language models can learn to represent considerable amounts of information, and can be quickly adapted to be usable for many tasks in natural language understanding and generation. However, these systems present their own modeling and data challenges. Designed as general neural architectures to be adaptable to a wide variety of end tasks, their behavior is opaque and disconnected from known reasoning strategies (Linzen et al., 2018, 2019). Meanwhile, the representations and reasoning capabilities they learn are affected by the quality of data, allowing negative signals such as artifacts (Gururangan et al., 2018) and biases (Zhao et al., 2019; Sheng et al., 2019; Tan and Celis, 2019) to be encoded.

In this dissertation, we are motivated by prior work in neuro-symbolic AI (Besold et al., 2017) to explore various fusions of neural and symbolic methods for commonsense representation and reasoning. By leveraging the strengths of each approach: the structure, interpretability, and programmability of symbolic algorithms, and the expressivity of neural networks, we tackle the challenges of increasing the coverage of commonsense knowledge representations and the robustness of commonsense reasoning systems for natural language understanding and generation.

1.1 Thesis Outline

In this dissertation, we explore novel neuro-symbolic approaches for commonsense representation and reasoning. Each chapter focuses on one of these two problems, and discusses a new system for combining neural networks with symbolic biases in furtherance of these goals.

In Chapter 2, we provide introductions on concepts that will be useful for understanding various sections of this dissertation. We convey background on topics in machine learning, neural networks, and natural language processing that arise throughout the document.

In Chapter 3, we focus on developing large-scale commonsense knowledge representations by learning neural representations of knowledge bases. Our approach, COMET, transfers language representations learned from pretraining on a large textual corpus to the task of hypothesizing knowledge tuples through language generation. Our empirical investigation shows that the knowledge tuples the model learns to gen-

erate are often deemed plausible and that the model can generalize to new situations it is presented with efficiently.

In Chapter 4, we explore how machines can learn to use large-scale commonsense knowledge to reason about how described events represent implicit changes to a latent world depicted in the text being read. Our approach uses COMET to dynamically construct contextual commonsense knowledge graphs about the situations described in text. We also propose a novel algorithm for reasoning over these generated graphs to answer questions about the situations presented. Our empirical studies demonstrate that our approach can be used to answer questions about narratives with no additional end task training.

In Chapter 5, we investigate the integration of commonsense reasoning through neural architecture design. We develop a modular reasoning architecture, the Neural Process Network, that tracks world state through simulation of the causal effects of actions. Our approach uses learned action function representations to change entity representations based on the text being read by the model. As the model makes errors in predicting attributes caused by actions it applied, it learns better transformation functions to represent the world underlying the language it reads. Our empirical studies show that modeling the world by reasoning about knowledge-grounded transformations leads to representations that are more transferable to domain-relevant downstream tasks.

In Chapter 6, we explore integrating commonsense understanding into neural models of text through rewarded interaction. We define a new training algorithm that uses reinforcement learning with neural knowledge-grounded rewards to guide a text generator to produce coherent text. As the text generator outputs text, it is rewarded for producing sequences that exhibit temporal knowledge typically associated with its domain. Over the course of multiple iterations of rewards, the text generator learns to write text that more closely reflects expected discourse. Our empirical evaluations demonstrate that generators trained in this manner produce more coherent sequences, particularly as the length of the generated text increases.

Finally, in Chapter 7, we summarize the work discussed in this dissertation and outline future research directions in uniting the fields of natural language understanding and commonsense reasoning.

1.2 Publications

Portions of this dissertation have appeared in the following publications:

- Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Çelikyilmaz, and Yejin Choi. 2019. COMET: Commonsense Transformers for Automatic Knowledge Graph Construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*

- Antoine Bosselut and Yejin Choi. 2019. Dynamic Knowledge Graph Construction for Zero-shot Commonsense Question Answering. *ArXiv*, abs/1911.03876

- Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018a. Simulating Action Dynamics with Neural Process Networks. In *Proceedings of the 6th International Conference for Learning Representations (ICLR)*

- Hannah Rashkin, Antoine Bosselut, Maarten Sap, Kevin Knight, and Yejin Choi. 2018. Modeling naive psychology of characters in simple commonsense stories. In *Proceedings of the Conference of the Association for Computational Linguistics*

- Antoine Bosselut, Asli Çelikyilmaz, Xiaodong He, Jianfeng Gao, Po-Sen Huang, and Yejin Choi. 2018b. Discourse-Aware Neural Rewards for Coherent Text Generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*

Chapter 2

Background

2.1 Basics

In this section, we provide the basic building blocks for many of the machine learning models introduced in later chapters. Deviations from these algorithms in later chapters will be described in detail.

2.1.1 Logistic Regression

Logistic regression is one of the bedrock algorithms used for classification (i.e., predicting a set of classes from a set of features). It is defined in the following way:

$$y = \sigma(\mathbf{w}^\intercal \mathbf{v} + b) \tag{2.1}$$

where $\mathbf{w} \in \mathbb{R}^D$ is a vector of weights, $\mathbf{v} \in \mathbb{R}^D$ is a vector of input features, D is the dimensionality of the input space, b is a scalar bias, and σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

which is a non-linear function that bounds a scalar value in the range [0, 1]. The σ function is often used to estimate probability masses.

Multi-class Logistic Regression We can generalize the definition of logistic regression by increasing the number of output classes that can be predicted and re-defining Equation (2.1):

$$\mathbf{y} = \texttt{softmax}(\mathbf{W}\mathbf{v} + \mathbf{b}) \tag{2.3}$$

where $\mathbf{W} \in \mathbb{R}^{C \times D}$ of weights and $\mathbf{b} \in \mathbb{R}^C$, $\mathbf{y} \in [0, 1]^C$, C is the number of classes that can be predicted by the classifier, and:

$$\texttt{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=0}^{C} e^{x_j}} \tag{2.4}$$

where C is the dimensionality of vector $\mathbf{x}$ (usually the number of classes being predicted) and i and j are arbitrary indices of vector $\mathbf{x}$. The `softmax` function allows us to convert a vector of scalar values to a probability distribution over a set of classes.

2.1.2 Feedforward Neural Networks

Logistic regression can be viewed as a neural network with no hidden layers, where input features $\mathbf{v}$ are projected directly to the output space $\mathbf{y}$. However, we can introduce additional projections between these two spaces, which we call hidden layers. In the simplest case with one hidden layer:

$$\mathbf{y} = \texttt{softmax}(\mathbf{W}_1\phi(\mathbf{W}_0\mathbf{v} + \mathbf{b}_0) + \mathbf{b}_1) \tag{2.5}$$

where $\mathbf{W_1} \in \mathbb{R}^{C \times D}$, $\mathbf{W}_0 \in \mathbb{R}^{D \times D}$, $\mathbf{b}_1 \in \mathbb{R}^C$, $\mathbf{b}_0 \in \mathbb{R}^D$, D is the dimensionality of the hidden layer, C is the dimensionality of the output layer, and ϕ is a non-linear function. Typically, to simplify notation, we would define $\mathbf{h}_1$ as the hidden layer (or hidden state) after the first projection:

$$\mathbf{h}_1 = \phi(\mathbf{W}_0\mathbf{v} + \mathbf{b}_0) \tag{2.6}$$

$$\mathbf{y} = \texttt{softmax}(\mathbf{W}_1\mathbf{h}_1 + \mathbf{b}_1) \tag{2.7}$$

If we set $\mathbf{h}_0 = v$ as the initial input state, we can generalize this equation to specify arbitrarily deep feedfor-

ward neural networks:

$$\mathbf{h}_\ell = \phi(\mathbf{W}_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_\ell) \tag{2.8}$$

$$\mathbf{y} = \texttt{softmax}(\mathbf{W}^{\mathbf{out}} \mathbf{h}_L + \mathbf{b}_{\mathbf{out}}) \tag{2.9}$$

where $\ell \in [1, L]$ defines the layer of the network, L defines the depth of the network, $\mathbf{W}_\ell \in \mathbb{R}^{D \times D}$ and $\mathbf{b}_\ell \in \mathbb{R}^{D \times D}$ define unique weights and biases at each layer, and $\mathbf{W}_{\mathbf{out}} \in \mathbb{R}^{C \times D}$ and $\mathbf{b}_{\mathbf{out}} \in \mathbb{R}^C$ specify a unique projection to the output classes. In practice, unless stated otherwise, we generally maintain the same dimensionality D among all the hidden layers and maintain the same activation function ϕ between layers. However, while this is common practice, it is not necessary, and varying the structure of neural networks is a common area of research (Greff et al., 2016; Zoph and Le, 2016).

2.1.3 Activation Functions

Feedforward neural networks require non-linearities (or activation functions) at the outputs of their layers to model more complex families of functions. In Equations (2.2) and (2.4), we introduced two non-linear functions that can serve as activation functions in neural networks, but many other functions can be used. We outline some of them below that are used throughout our work:

Hyperbolic Tangent A historically popular activation function for deep neural networks is the hyperbolic tangent function:

$$\texttt{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.10}$$

which bounds its output in the range [-1,1]. It is used as an activation function in most gated recurrent neural network architectures such as the Long Short-term memory (LSTM) and gated recurrent unit (GRU).

Recitified Linear Unit (ReLU) More recently, activation functions based on the rectified linear unit (ReLU, Glorot et al., 2011) have emerged in popularity. The basic ReLU unit is defined in the following

way:

$$\text{ReLU}(x) = \max(0, x) \tag{2.11}$$

meaning the output of a ReLU unit is lower bounded by 0, but not upper bounded. A downside of the ReLU activation is that a non-optimal initialization to the network can cause subnetworks of the network to be "shut off," as their forward values never exceed 0. The parametric rectified linear unit (PReLU, He et al., 2015) was designed to offset this shortcoming by lower-bounding the ReLU less strictly:

$$\text{PReLU}(x) = \begin{cases} x & x \geq 0 \\ ax & x < 0 \end{cases} \tag{2.12}$$

where a is a learnable parameter that serves as the slope of the input to the unit when $x < 0$.

Gaussian Error Linear Unit (GeLU) The final activation function used in this work is the Gaussian Error Linear Unit (GeLU; Hendrycks and Gimpel, 2016), which, once again, is an extension of the ReLU that displays better convergence properties:

$$\text{GeLU}(x) = 0.5x(1 + \texttt{tanh}(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3))) \tag{2.13}$$

2.1.4 Layer Normalization

A challenge in training deep neural networks is the principle of "covariate shift," where the gradients with respect to the weights in one layer $\nabla \mathbf{W}_\ell$ are highly dependent on the output activations of the previous layer $\mathbf{h}_{\ell-1}$, leading to slower training and potentially saturated activations (Ioffe and Szegedy, 2015). Various techniques for normalizing layer inputs, such as batch normalization (Ioffe and Szegedy, 2015), weight normalization (Salimans and Kingma, 2016), and layer normalization (Ba et al., 2016), have been proposed to combat this problem.

Here, we focus briefly on layer normalization (Ba et al., 2016), a common building block in modern neural architectures for language modeling. Layer normalization involves normalizing the input neurons to the next layer based on layer-wise statistics. The mean (μ) and standard error (σ) of the output activations

are computed for each layer in the network:

$$\mu^{\ell} = \frac{1}{D}\sum_{i=1}^{D}\tilde{h}_i^{\ell} \tag{2.14}$$

$$\sigma^{\ell} = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(\tilde{h}_i^{\ell} - \mu^{\ell})^2} \tag{2.15}$$

where D is the dimensionality of the output vector of activations, and $\tilde{h}_i^{\ell}$ corresponds to the i^{th} activation of the output vector $\tilde{\mathbf{h}}^{\ell}$. Using these statistics, the output vector is normalized before being passed to the next stage in the network:

$$\mathbf{h}^{\ell} = \frac{\mathbf{g}^{\ell}}{\sigma^{\ell}} \odot (\tilde{\mathbf{h}}^{\ell} - \mu^{\ell}) + \mathbf{b}^{\ell} \tag{2.16}$$

where $\mathbf{g}$ and $\mathbf{b}$ are learned parameter vectors for each layer ℓ, and $\mathbf{h}^{\ell}$ is the normalized layer output. Using layer normalization stabilizes inter-layer distributional shift, stabilizing and accelerating learning.

2.2 Deep Learning for Natural Language Processing

2.2.1 Language Models

Given a sequence of words $X = \{x_0, ..., x_n, ..., x_N\}$, a language model's purpose is to be able to predict the likelihood of the sequence:

$$P(X) = P(x_0, ..., x_n, ..., x_N) \tag{2.17}$$

where $P(X)$ defines the joint probability of these words appearing in this order. A common strategy to simplify the computation of this estimate is to use the chain rule to condition each token's probability solely

on its history:

$$P(X) = P(x_0) \times P(x_1|x_0) \times ... \times P(x_N|x_{N-1}, ..., x_1, x_0) = P(x_0) \times \prod_{n=1}^{N} P(x_n|\{x_{<n}\}) \tag{2.18}$$

which simplifies the problem of language modeling to being able to predict the likelihood of a particular word given its preceding history. It is standard to represent a language model's distribution in terms of its log-likelihood:

$$\log P(X) = \log P(x_0) + \log P(x_1|x_0) + ... + \log P(x_N|x_{N-1}, ..., x_1, x_0) = \log P(x_0) + \sum_{n=1}^{N} \log P(x_n|\{x_{<n}\}) \tag{2.19}$$

While there is a long history of approaches designed for language modeling, recent work in this area has focused on using deep neural networks, such as recurrent neural networks (§2.3) and transformers (§2.4), as estimators for $P(X)$.

2.2.2 Word Embeddings

As seen in the previous section, most machine learning models require vector inputs for computation. Word embeddings are used to map discrete words that are found in language to a vector space that can be processed by learning agents. Early works in constructing word embeddings for a fixed vocabulary V involved initializing a sparse vector $\mathbf{x} \in \mathbb{R}^V$ where each index in the vector mapped from a word in the vocabulary and whose value corresponded to statistics about the word within the document of interest (Luhn, 1957; Jones, 1972). We use models trained on these frequency-based embeddings as baselines in chapters 4 and 5.

While these embeddings were powerful tools for many applications, they could not capture the polysemous nature of language. Each word was ultimately represented by a single index in the V-dimensional vector. As a result, Bengio et al. (2003) were the first to propose learnable dense representations for word embeddings. They initialized embeddings as dense vectors $\mathbf{x} \in \mathbb{R}^D$, allowing each word to be represented by a unique vector with D dimensions. To train a language model, they initialized a database of word embeddings $\mathbf{X} \in \mathbb{R}^{V \times D}$ where each row corresponded to a vector representation for a unique word in the

vocabulary V. Whenever a word was processed by the model, the model would index its corresponding vector and use it in future steps. By training on a language modeling objective (§2.2.1), these embeddings eventually learned parameters that made it more likely that the language model would be able to predict the next word in the sequence.

Modern approaches to training word embeddings have built on top of this original approach and developed new model architectures, new training algorithms, and used larger datasets to learn better representations of words (Mikolov et al., 2013a,b; Pennington et al., 2014). However, in all of these approaches, the practice of maintaining a database of embeddings $\mathbf{X} \in \mathbb{R}^{V \times D}$ and indexing words from it remains the dominant paradigm for mapping language to an input that deep neural networks can process.

We use pretrained word embeddings in several instances throughout this work. In chapters 3, 4, and 5, we use GloVe embeddings (Pennington et al., 2014) as components of baseline approaches. In Chapter 5, we also use GloVe embeddings (Pennington et al., 2014) and word2vec embeddings (Mikolov et al., 2013b) as initialization embeddings for our models.

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are expressive neural models that can encode representations for sequential inputs. These models are particularly useful in the field of natural language processing, where language is often presented as sequences of tokens. These models can compress full-text sequences down to arbitrarily-sized vector representations. Below, we explore popular variants of the recurrent neural network, and describe how they are commonly used in modern NLP pipelines.

2.3.1 Elman Networks

Elman networks (Elman, 1990) were an early proposal for using recurrent neural networks to model sequential data. Often viewed as the *vanilla* RNN, the Elman network is defined according to the following behavior:

$$\mathbf{h}_t = \texttt{tanh}(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{2.20}$$

where $\mathbf{h}_t$ is the new hidden state of the network, $\mathbf{h}_{t-1}$ is the hidden state at the previous time step, and $\mathbf{x}_t$ is the input to the RNN at the current time step. The initial state $\mathbf{h}_0$ of an Elman network is often set as a vector of 0s or initialized as learnable vector of parameters. Much like feedforward networks, recurrent neural networks can have multiple layers:

$$\mathbf{h}_t^\ell = \texttt{tanh}(\mathbf{W}_x \mathbf{h}_t^{\ell-1} + \mathbf{W}_h \mathbf{h}_{t-1}^\ell + \mathbf{b}_h) \tag{2.21}$$

where $\mathbf{h}_t^{\ell-1}$ corresponds to the output of the RNN at the previous layer and $\mathbf{h}_{t-1}^\ell$ is previous hidden state at the current layer. In many natural language processing applications, these output states $\mathbf{h}_t$ serve as representations of the sequence and are passed as input to downstream task models. For example, a logistic regression classifier introduced in Section 2.1.1 could set $v = \mathbf{h}_t$ and predict classes based on this input feature representation.

However, a shortcoming of the Elman network is that it suffers from the *vanishing gradient* problem, whereby the recursive definition of the network requires multiple passes through a bounded activation function. As a result, it becomes difficult to backpropagate gradients from the task loss to the representations of the early steps of a sequence. To combat this issue, gated recurrent neural networks such as the long short-term memory (LSTM) and gated recurrent unit (GRU) have been more commonly used in recent years.

2.3.2 Long Short-term Memory (LSTM)

The Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) is a recurrent neural network architecture that controls gradient flow through its network using a cell state that is not passed through a bounded activation function. An LSTM contains a cell state $\mathbf{c}_t$ at its core, which encodes a hidden representation of the information collected by the LSTM up to a point in time t. Multiple gates control how input information is added to the maintained cell state. The input gate, $\mathbf{i}_t$, controls the information that is fed into the cell state. The forget gate, $\mathbf{f}_t$, controls the information retained by the cell state. The output gate, $\mathbf{o}_t$, determines how much information should be output from the cell state after the hidden representation has been computed. The values of these gates are computed in the following way:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) \tag{2.22}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f) \tag{2.23}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) \tag{2.24}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ are the gates described above, all $\mathbf{W}$ are unique projection matrices, all $\mathbf{b}$ are unique bias vectors, $\mathbf{x}_t$ corresponds to the input vector to the LSTM at any time step, and $\mathbf{h}_{t-1}$ corresponds to the output of the LSTM at the previous step. Similar to Elman networks, the initial hidden state $\mathbf{h}_0$ is often set as a vector of 0s or set as an additional parameter and learned. Once the gates are set, the representation of the cell state $\mathbf{c}_t$ can be updated and the output of the LSTM, $\mathbf{h}_t$ can be computed:

$$\mathbf{g}_t = \phi(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g) \tag{2.25}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \tag{2.26}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \phi(\mathbf{c}_t) \tag{2.27}$$

where $\mathbf{g}_t$ is a projected representation of the input to the LSTM, all $\mathbf{W}$ are unique projection matrices, all $\mathbf{b}$ are unique bias vectors, and $\odot$ represents the Hadamard product. As with the hidden state, the initial cell state $\mathbf{c}_0$ is often set as vectors of 0s or set as an additional learnable parameter vector.

An augmentation of the LSTM, the bidirectional LSTM (biLSTM), computes a representation of a sequence in both the forward ($\mathbf{h}_t^f$) and backward ($\mathbf{h}_t^b$) directions using two distinct sets of parameters. At the output, these representations are concatenated:

$$\mathbf{h}_t = [\mathbf{h}_t^f, \mathbf{h}_t^b] \tag{2.28}$$

to form a single representation of the encoded sequence. We use LSTMs as baseline approaches in the empirical studies of Chapters 3, 4, and 5.

2.3.3 Gated Recurrent Units (GRU)

The gated recurrent unit (Cho et al., 2014) is an alternate augmentation of the Elman network that uses an update gate $\mathbf{z}_t$ and reset gate $\mathbf{r}_t$ to control gradient flow:

$$\mathbf{z}_t = \sigma(\mathbf{W}_{xz}\mathbf{x}_t + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_z) \tag{2.29}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xr}\mathbf{x}_t + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_r) \tag{2.30}$$

where $\mathbf{z}_t$ and $\mathbf{r}_t$ are the update and reset gate, respectively, all $\mathbf{W}$ are unique projection matrices, all $\mathbf{b}$ are unique bias vectors, $\mathbf{x}_t$ corresponds to the input vector to the GRU at any time step, and $\mathbf{h}_{t-1}$ corresponds to the output of the GRU at the previous step. As before, the initial hidden state $\mathbf{h}_0$ is often set as a vector of 0s or set as an additional parameter and learned. Once the gates are set, the output of the GRU $\mathbf{h}_t$ can be computed:

$$\mathbf{g}_t = \phi(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_g) \tag{2.31}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \mathbf{g}_t \tag{2.32}$$

where $\mathbf{g}_t$ is the computed update representation of the input, all $\mathbf{W}$ are unique projection matrices, all $\mathbf{b}$ are unique bias vectors, and $\odot$ represents the Hadamard product. Similar to the LSTM, the bidirectional GRU (biGRU) is an augmentation of the GRU that computes a representation of a sequence in both the forward ($\mathbf{h}_t^f$) and backward ($\mathbf{h}_t^b$) directions and concatenates them.

While empirical results have shown that there are capabilities that are not learnable by the GRU that are by the LSTM (e.g., counting; Weiss et al., 2018), they are generally used interchangeably with little difference in performance (Chung et al., 2014). We use gated recurrent units as a model base in Chapter 6 and as baseline approaches in the empirical studies of Chapter 5.

2.4 Transformers

A shortcoming of recurrent neural networks is that the representation of the input $\mathbf{h}_t$ at any step depends on an iterative state representation of previous steps $\mathbf{h}_{t-1}$. This recursive definition of the hidden state $\mathbf{h}_t$ has two clear disadvantages. First, it induces a computational bottleneck, as tokens can only be processed in series, limiting the benefits offered by modern parallel computing hardware. Second, it causes an impediment to learning, as gradients must backpropagate through many steps before affecting representations of earlier tokens in the sequence.

In response to these issues, Vaswani et al. (2017) proposed the transformer, a multi-layer, non-recurrent neural sequence model that instead uses a new *self-attention* component to construct expressive representations of its input. Self-attention allows the model to construct step representations of the input in parallel, as a state representation is re-computed from the full input at every step. These representations do not rely on a time-dependent state that is standard in recurrent models. We explore these components in greater detail in the following sections.

2.4.1 Transformer Components

Transformer Block A transformer is composed of multiple layers ℓ of architecturally identical transformer blocks (though with unique trainable parameters). Each transformer block applies the following transformations to the input of the block $\mathbf{h}^{\ell-1}$:

$$\tilde{\mathbf{g}}^{\ell} = \textsc{MultiAttn}(\mathbf{h}^{\ell-1}) \tag{2.33}$$

$$\mathbf{g}^{\ell} = \textsc{LayerNorm}(\tilde{\mathbf{g}}^{\ell} + \mathbf{h}^{\ell-1}) \tag{2.34}$$

$$\tilde{\mathbf{h}}^{\ell} = \text{FFN}(\mathbf{g}^{\ell}) \tag{2.35}$$

$$\mathbf{h}^{\ell} = \textsc{LayerNorm}(\tilde{\mathbf{h}}^{\ell} + \mathbf{g}^{\ell}) \tag{2.36}$$

where MULTIATTN is a multi-headed self-attention mechanism (defined below), FFN is a two-layer feedforward network (§2.1.2), and LAYERNORM represents a layer normalization (Ba et al., 2016) operation that is applied to the output of the self-attention and the feedforward network. The inputs to the LAYER-

NORM operations contain a residual connection that sums the output of and input to the previous operation. We note that this particular architecture of the transformer block is a transformer encoder and is the specific architecture of the general purpose transformer (GPT; Radford et al., 2018). Other transformer implementations may deviate from these exact equations (e.g., Yang et al., 2019), but they generally all include a multi-headed attention component followed by a feedforward network.

Multi-headed Attention The multi-headed attention module of each transformer block, as defined by Vaswani et al. (2017), has three inputs: a query $\mathbf{q} \in \mathbb{R}^D$, a set of keys $\mathbf{K} \in \mathbb{R}^{N \times D}$, and a set of values $\mathbf{V} \in \mathbb{R}^{N \times D}$. The attention is made of multiple $heads$ that each compute a unique scaled dot product attention distribution over $\mathbf{V}$ using $\mathbf{q}$ and $\mathbf{K}$:

$$\text{ATTENTION}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{K}^T}{\sqrt{D}}\right)\mathbf{V} \tag{2.37}$$

where D is the dimensionality of the input vectors representing the query, keys and values. For each of the heads, $\mathbf{q}$, $\mathbf{K}$, and $\mathbf{V}$ are uniquely projected prior to the attention being computed:

$$A_i = \text{ATTENTION}(\mathbf{q}\mathbf{W}_i^q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{2.38}$$

where A_i is the output of a single attention head and $\mathbf{W}_i^q$, $\mathbf{W}_i^K$, and $\mathbf{W}_i^V$ are head-specific projections for $\mathbf{q}$, $\mathbf{K}$, and $\mathbf{V}$, respectively. The outputs of the attention heads A_i are then concatenated:

$$\text{MULTIH}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = [A_1; ...; A_b]\mathbf{W}^o \tag{2.39}$$

where $\mathbf{W}^o$ is an output projection of the concatenated outputs of the attention heads. Typically, the output of the previous layer's transformer block is used as the query input for the multi-headed attention of the next block. The keys $\mathbf{K}$ and values $\mathbf{V}$ are the sets of outputs of the previous layer's block for other time steps:

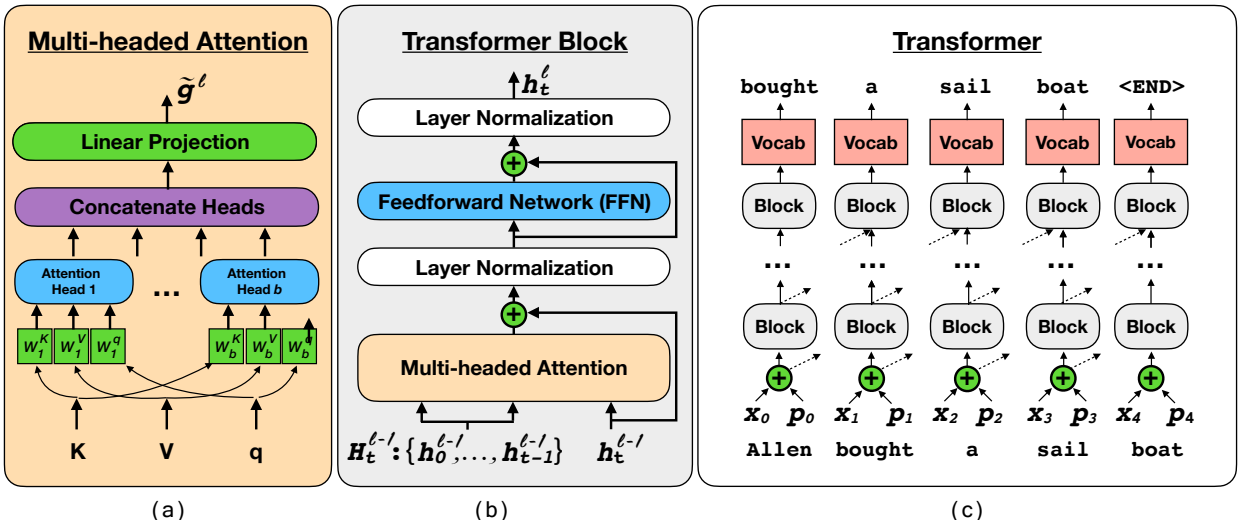


Figure 2.1: Transformer Language Model. **(a)** In the multi-headed attention module, the keys, values, and query all pass through a head-specific projection before a scaled dot-product attention is computed between them. The outputs of the heads are concatenated and projected. **(b)** Inside the transformer block, the outputs of all the previous layer blocks from earlier time steps $\mathbf{H}_t^{\ell-1} = \{\mathbf{h}^{\ell-1}\}_{<t}$ are input to the multi-headed attention with the preceding block for the current time step as the query $\mathbf{h}_t^{\ell-1}$. **(c)** Each token is an input to a first-layer block along with all preceding tokens. Dotted lines indicate outputs to all future blocks in the next layer and inputs from all preceding blocks in the previous layer.

$$\textsc{MultiAttn}(\mathbf{h}_t^{\ell-1}) = \textsc{MultiH}(\mathbf{h}_t^{\ell-1}, \mathbf{H}_t^{\ell-1}, \mathbf{H}_t^{\ell-1}) \tag{2.40}$$

where $\mathbf{H}_t^{\ell-1}$ corresponds to a subset of previous layer transformer block outputs. In left-to-right transformers (§2.4.2), for example, $\mathbf{H}_t^{\ell-1} = \{\mathbf{h}^{\ell-1}\}_{<t}$, the set of previous layer transformer block outputs for time steps preceding t.

Positional Embeddings Since the transformer, a self-attention model, has no concept of ordering of tokens, a position embedding p_t is initialized for each absolute position in the sequence (Vaswani et al., 2017). For any input word $x_t \in X$, our encoding of the input is the sum of its word embedding, $\mathbf{x}_t$ with a position embedding encoding its absolute position in the sequence X:

$$\mathbf{h}_t^0 = \mathbf{x}_t + \mathbf{p}_t \tag{2.41}$$

where $\mathbf{p}_t$ is the position embedding for time step t, and $\mathbf{h}_t^0$ is the input to the first transformer layer.

2.4.2 Left-to-Right Transformer Language Models

Left-to-right transformer language models (Radford et al., 2018, 2019; Brown et al., 2020) are trained on a typical language modeling objective (Eq. 2.19). They are trained to maximize the conditional log-likelihood of predicting the next token in a sequence given the preceding ones. As a result, they mask the tokens of future time steps when computing self-attention distributions over their inputs. Figure 2.1 shows the internal processes of a left-to-right transformer. In Figure 2.1(b), we see that $\mathbf{H}_t^{\ell-1} = \{\mathbf{h}^{\ell-1}\}_{<t}$, indicating that only the keys and values corresponding to preceding time steps can be attended to by the query of the current time step. We use left-to-right transformer language models as primary components of our approaches in Chapters 3 and 4.

2.4.3 Bidirectional Transformer Language Models

A final class of transformer language model is the bidirectional transformer, whose most famous iteration, BERT, was first introduced by Devlin et al. (2018), and then improved through successive modifications (Joshi et al., 2020; Liu et al., 2019; Lan et al., 2020). In contrast to left-to-right models that only condition on preceding tokens when learning to predict a token at a particular position in a sequence (Eq. 2.19), bidirectional transformers can condition on future tokens as well. As such, they cannot be trained with a typical language modeling objective, which would make it trivial to predict the token at a particular time step. Instead, a new objective function for *masked language modeling* is proposed, where for a token $x_t \in X$, its log-likelihood is predicted as:

$$\log P(x_t) = \log P(x_t | \{x_j\}_{j \neq t}, \texttt{[MASK]}) \tag{2.42}$$

where `[MASK]` is a special token that replaces the token x_t as the input of the sequence, and $\{x_j\}_{j \neq t}$ is the set of all other tokens in X that are not x_t. The right-hand term of the equation is approximated using the bidirectional transformer. While BERT can not be used for classical language modeling, the representations it learns for words and sequences allow it to provide a powerful boost to models across a wide variety of NLP tasks. As a result, BERT and its variants have become standard baselines in most empirical studies

in natural language understanding. We use bidirectional transformer language models as baselines in the empirical study in Chapter 4.

Chapter 3

Commonsense Transformers as Neural Representations of Knowledge Graphs

A core sub-challenge in developing natural language processing agents that can use commonsense knowledge in their predictive pipelines is having access to high-coverage knowledge bases. However, the scale of general commonsense knowledge makes historically-used manual approaches to designing knowledge bases unscalable. Meanwhile, extractive automatic methods for constructing knowledge bases only gather explicitly mentioned relationships, missing implicit commonsense knowledge that is often unstated (Gordon and Van Durme, 2013).

In this chapter, we explore how to construct large-scale, general commonsense knowledge representations for downstream use by task models. We posit that an important step toward automatic commonsense acquisition is the development of *generative* models of commonsense knowledge. Consequently, we propose ***COM**monsEnse **T**ransformers* (COMET), new models that learn to generate rich and diverse commonsense descriptions in natural language. Despite the challenges of commonsense modeling, our investigation reveals promising results when implicit knowledge from deep pre-trained language models is transferred to the task of generating tuples of explicit knowledge for commonsense knowledge graphs.

We present a comprehensive study of automatic commonsense knowledge base construction for two prevalent commonsense knowledge graphs: ATOMIC (Sap et al., 2019a) and ConceptNet (Speer et al., 2017).

The material in this chapter is adapted from Bosselut et al. (2019)

Empirical results demonstrate that COMET is able to generate novel knowledge tuples that humans rate as high quality, which approaches human performance for these resources. These findings suggest that using generative commonsense models for automatic commonsense knowledge base construction could soon be a plausible alternative to extractive methods.

3.1 Introduction

When reading text, humans make commonsense inferences that frame their understanding of the narrative being presented. For machines to achieve this capability, they must be able to acquire relevant and correct commonsense for an unbounded set of situations. In our approach, we cast commonsense acquisition as knowledge base construction and investigate whether large-scale language models can be effectively used to hypothesize the knowledge necessary to automatically construct a commonsense knowledge base (KB).

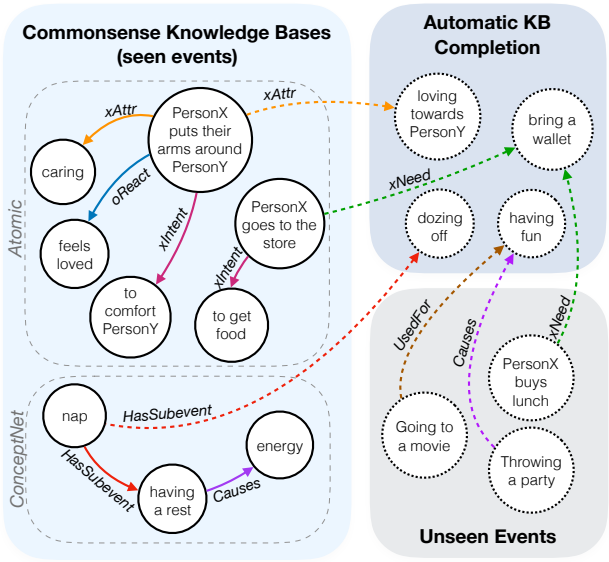


Figure 3.1: COMET learns from an existing knowledge base (solid lines) to be able to generate novel nodes and edges (dashed lines).

Automatic KB construction is a long-standing goal of artificial intelligence research due to the difficulty of achieving high concept coverage in high-precision curated KBs (Lenat, 1995; Miller, 1995). Previous work has developed models capable of reading and extracting semi-structured text (Suchanek et al., 2007; Hoffart et al., 2013; Auer et al., 2007; Bollacker et al., 2008) and unstructured text (Dong et al., 2014; Carlson et al., 2010; Nakashole et al., 2011, 2012; Niu, 2012) into relational schemas that can be queried for downstream applications. A common thread of these approaches, however, is the focus on encyclopedic knowledge, which lends itself to a well-defined space of entities and relations that can be modeled.

Commonsense knowledge, however, does not cleanly fit into a schema comparing two entities with a known relation, leading current approaches to model "entities" as natural language phrases and relations as any concept that can link them (Li et al., 2016b; Sap et al., 2019a). OpenIE approaches display this property

of open text entities and relations (Etzioni et al., 2011; Fader et al., 2011; Mausam et al., 2012), but being extractive, they only capture knowledge that is explicitly mentioned in text, limiting their applicability for capturing commonsense knowledge, which is often implicit (Gordon and Van Durme, 2013).

Meanwhile, recent progress in training deep contextualized language models (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018) provides an opportunity to explore beyond extractive methods as an avenue for commonsense KB construction. These large-scale language models display impressive performance when their underlying representations are tuned to solve end tasks, achieving state-of-the-art results on a variety of complex problems. Exploiting these benefits, we define the ***COM****monsEnse* ***T****ransformer* (COMET), which constructs commonsense KBs by using existing tuples as a seed set of knowledge on which to train. Using this seed set, a pre-trained language model learns to adapt its learned representations to the task of generating knowledge tuples.

We summarize our contributions in this chapter as follows. First, we develop a generative approach to knowledge base construction. A model must learn to produce new nodes and identify edges between existing nodes by generating phrases that coherently complete an existing seed phrase and relation type. Second, we develop a framework for using large-scale transformer language models to learn to produce commonsense knowledge tuples. Finally, we perform an empirical study on the quality, novelty, and diversity of the commonsense knowledge produced by our approach for two domains, ATOMIC and ConceptNet, as well as an efficiency study on the number of seed tuples needed to learn an effective knowledge model. The results indicate that COMET is able to produce high quality tuples as human judges find that 77.5% of generated tuples for ATOMIC events and 91.7% of generated tuples for ConceptNet relations are correct.

3.2 Related Work

3.2.1 Knowledge Base Construction

Previous work has looked at constructing knowledge bases as relational schemas using expert knowledge (Lenat, 1995; Bodenreider, 2004; Miller, 1995), semi-structured text extraction (Suchanek et al., 2007; Hoffart et al., 2013; Auer et al., 2007; Bollacker et al., 2008) and unstructured text extraction (Dong et al., 2014; Carlson et al., 2010; Nakashole et al., 2011, 2012; Niu, 2012). In our work, we focus on construction

of commonsense knowledge bases which require the use of open-text events rather than a well-defined relational schema structure. Other work in information extraction can also be applied to knowledge base construction with open-text entities (Soderland et al., 2010; Etzioni et al., 2011; Fader et al., 2011; Mausam et al., 2012; Fan et al., 2010; Cui et al., 2018), but these methods typically extract explicitly stated text relations. Conversely, our approach uses language generation to hypothesize examples of knowledge that are often unstated in text, as commonsense information typically is (Gordon and Van Durme, 2013).

3.2.2 Knowledge Base Completion

A parallel task to constructing knowledge bases is that of knowledge base completion, which explores predicting links between known entities in a knowledge base (Riedel et al., 2013; Bordes et al., 2013). However, it is still an open question as to whether knowledge base completion methods can be used for constructing knowledge graphs. These methods require a known set of entities to pre-populate the knowledge base and cannot add new entities to the knowledge base without human intervention. In recent years, many approaches have approached this problem using deep learning methods that embed entities into a vector space and define functions to predict entities that connect to them through known links (Toutanova et al., 2015; Yang et al., 2015; Liu et al., 2016c; Nguyen et al., 2016; Nickel et al., 2016; Trouillon et al., 2016; Dettmers et al., 2018). In contrast to these approaches that predict entities through classification over a known entity vocabulary, we *generate* the tokens that make up the entity, allowing us to produce novel entities that can then be added back to the knowledge graph.

3.2.3 Commonsense Knowledge Base Completion

Existing work on generation of novel commonsense knowledge has also used ConceptNet and ATOMIC as underlying KBs. Specifically, Li et al. (2016b) proposed a set of neural network models for scoring tuples in ConceptNet. Our work differs from this approach as their models evaluate full tuples rather than learning to generate the phrases to make new nodes in the knowledge graph. Saito et al. (2018) builds upon Li et al. (2016b)'s work by proposing a joint model for completion and generation of commonsense tuples. Their work, however, focuses on using tuple generation to augment their KB completion model, rather than to increase coverage in commonsense KB construction. Finally, Sap et al. (2019a) use LSTM encoder-

decoder models to hypothesize commonsense knowledge about social situations. We use transformers and investigate the effect of using pre-trained representations (Radford et al., 2018) for initialization.

3.2.4 Transformers and Pre-training

Finally, our work builds on previous studies on adapting pre-trained language models for various sequence labeling, classification, and NLI end tasks (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018; Liu et al., 2019). Our research investigates how pre-trained language models can be used for large-scale commonsense KB construction by generating new graph nodes and edges between nodes.

3.3 Learning to Generate Commonsense Knowledge Descriptions

COMET is an adaptation framework for constructing commonsense knowledge bases from language models by training the language model on a seed set of knowledge tuples. These tuples provide COMET with the KB structure and relations that must be learned, and COMET learns to adapt the language model representations learned from pre-training to add novel nodes and edges to the seed knowledge graph.

3.3.1 Task

More specifically, the problem assumes COMET is given a training knowledge base of natural language tuples in $\{s, r, o\}$ format, where s is the phrase subject of the tuple, r is the relation of the tuple, and o is the phrase object of the tuple. For example, a ConceptNet tuple relating to "taking a nap" would be: (s="take a nap", r=Causes, o="have energy"). The task is to generate o given s and r as inputs.

Notation We define $X^s = \{x_0^s, ..., x_{|s|}^s\}$ as the tokens that make up the subject of the tuple, $X^r = \{x_0^r, ..., x_{|r|}^r\}$ as the tokens that make up the relation of the tuple, and $X^o = \{x_0^o, ..., x_{|o|}^o\}$ as the tokens that make up the object of the tuple. The embedding for any word x is denoted as $\mathbf{x}$.

3.3.2 Transformer Language Model

While COMET is agnostic to the language model with which it is initialized, we use the transformer language model architecture introduced in Radford et al. 2018 (GPT), which uses multiple transformer blocks of

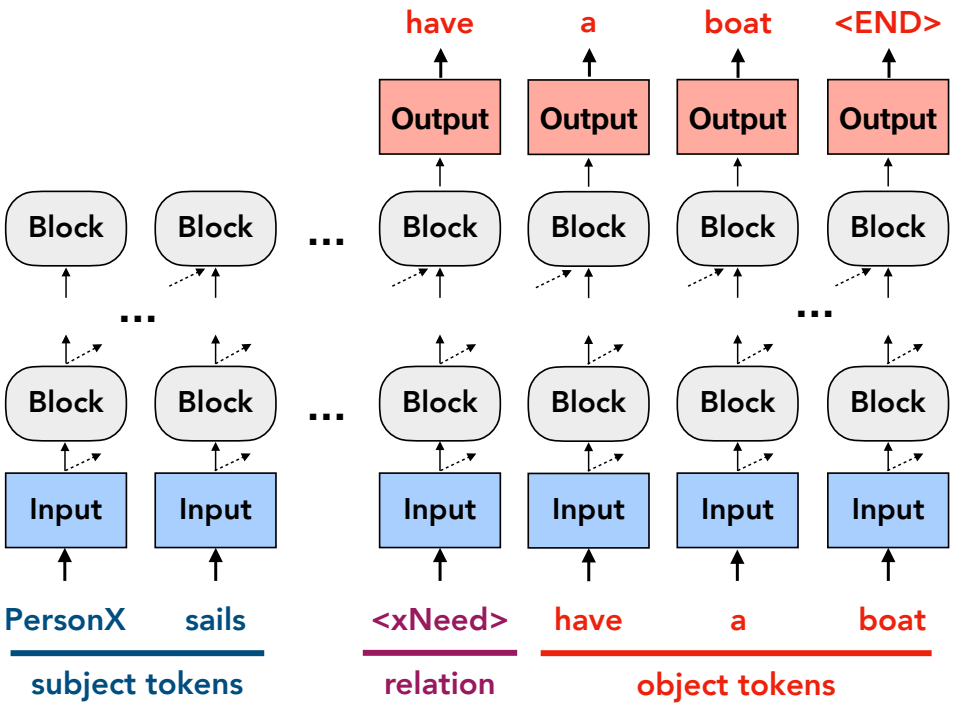


Figure 3.2: Model diagram. COMET receives the subject tokens and relation as input, and learns to generate the object tokens, thereby learning the structure of a tuple from the knowledge base.

multi-headed scaled dot product attention and fully connected layers to encode input text (Vaswani et al., 2017). Figure 2.1 depicts different components of the GPT architecture and we show how this approach is tuned for training COMET in more depth below.

Input Layer As input to the model, we represent a knowledge tuple $\{s, r, o\}$ as a concatenated sequence of the words of each item of the tuple:

$$X = \{X^s, X^r, X^o\} \tag{3.1}$$

For any input word $x_t \in X$, our encoding follows the procedure introduced in Section 2.4.1: we sum its word embedding $\mathbf{x}_t \in \mathbb{R}^D$ with a position embedding encoding its absolute position in the sequence X:

$$\mathbf{h}_t^0 = \mathbf{x}_t + \mathbf{p}_t \tag{3.2}$$

where $\mathbf{p}_t \in \mathbb{R}^D$ is the position embedding for time step t, $\mathbf{h}_t^0 \in \mathbb{R}^D$ is the input to the first transformer layer, and D is the dimensionality of the embeddings.

Transformer Blocks We review the structure of the transformer block to clarify its application in training COMET. Each intermediate layer in the transformer between the input and output layers (see Figure 3.2) contains an architecturally identical transformer block (though with unique trainable parameters). At every time step t the following transformations are applied to the input to the block:

$$\tilde{\mathbf{g}}^\ell = \textsc{MultiAttn}(\mathbf{h}^{\ell-1}) \tag{3.3}$$

$$\mathbf{g}^\ell = \textsc{LayerNorm}(\tilde{\mathbf{g}}^\ell + \mathbf{h}^{\ell-1}) \tag{3.4}$$

$$\tilde{\mathbf{h}}^\ell = \text{FFN}(\mathbf{g}^\ell) \tag{3.5}$$

$$\mathbf{h}^\ell = \textsc{LayerNorm}(\tilde{\mathbf{h}}^\ell + \mathbf{g}^\ell) \tag{3.6}$$

where $\mathbf{h}^{\ell-1} \in \mathbb{R}^D$ corresponds to the output of the previous block (or the output of input encoder in the case of the first block, $\ell = 1$). MULTIATTN is the multi-headed self-attention mechanism defined in section 2.4.1, FFN is a two-layer feed-forward network (§2.1.2), and LAYERNORM represents a layer normalization (Ba et al., 2016) operation that is applied to the output of the self-attention and the feedforward network.

Output Head The output head receives the final-layer representation from the transformer blocks at each time step $\mathbf{h}_t^\ell$ and computes a distribution over possible words to generate:

$$P(x_t|x_{<t}) = \text{softmax}(\mathbf{W}^{\mathbf{out}}\mathbf{h}_t^\ell) \tag{3.7}$$

where $\mathbf{W}^{\mathbf{out}} \in \mathbb{R}^{V \times D}$ and V is the size of the vocabulary of the underlying language model. We note that the projection weights $\mathbf{W}^{\mathbf{out}}$ are tied with the input word embeddings.

3.4 Training COMET

COMET is trained to learn to produce the phrase object o of a knowledge tuple given the tuple's phrase subject s and relation r. More specifically, given the concatenation of the tokens of s and r: $[X^s, X^r]$ as

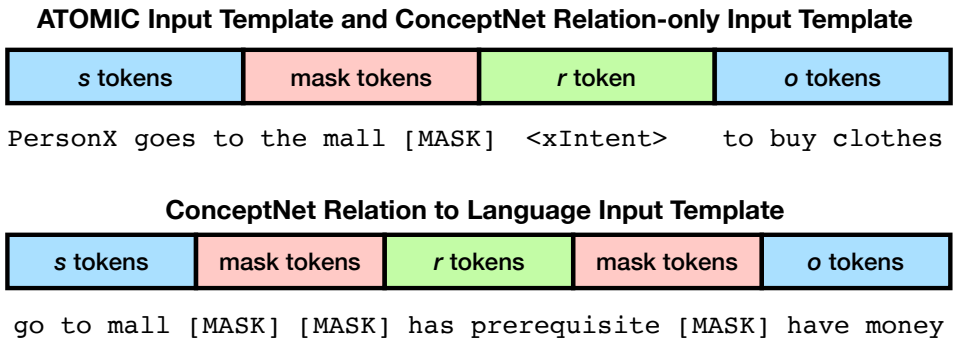


Figure 3.3: Input token setup for training configurations. For the ATOMIC dataset, the tokens of the subject, X^s (e.g., PersonX goes to the mall) are followed by masking tokens, which is followed by a single relation token X^r (e.g., `xIntent`), and then the object tokens X^o (e.g., to buy clothes). The model receives the same input for ConceptNet, except that a second set of masking tokens separate X^r and X^o because X^r can have a variable number of tokens for ConceptNet (§3.6.2)

input, the model must learn to generate the tokens of o: X^o (See §3.3.1 for definitions of these variables).

Loss Function To achieve this goal, COMET is trained to maximize the conditional loglikelihood of predicting the phrase object tokens, X^o:

$$\mathcal{L} = -\sum_{t=|s|+|r|}^{|s|+|r|+|o|} \log P(x_t|x_{<t}) \tag{3.8}$$

where $|s|$, $|r|$, and $|o|$ are the number of tokens in the subject phrase, relation, and object phrase, respectively. Figure 3.3 outlines how the tokens in s, r, and o are organized for different training tasks.

Datasets COMET relies on a seed set of knowledge tuples from an existing KB to learn to produce commonsense knowledge. We use the ATOMIC (Sap et al., 2019a) and ConceptNet (Speer et al., 2017) knowledge graphs as seed sets, but other commonsense knowledge resources could have been used as COMET can be used to adapt language models to arbitrary knowledge graphs.

Initialization Parameters are initialized with the final language model weights from Radford et al. (2018). Additional special tokens that are added to the vocabulary for fine tuning (e.g., relation embeddings – `oReact` for ATOMIC and `IsA` for ConceptNet) are initialized by sampling from the normal distribution.

Hyperparameters Following Radford et al. (2018)'s design of the GPT model, we initialize COMET with 12 layers, 768-dimensional hidden states, and 12 attention heads, and use use a dropout rate of 0.1. During training, our batch size is 64. For ATOMIC, we use a maximum learning rate of $\eta = 6.25\text{e-}5$ with a warmup period of 100 minibatches. After, we decay the learning rate linearly until the end of training. We train for 50k minibatches and use early stopping. We clip gradients when their norm is greater than 1. The remainder of our hyperparameters are the same as in Radford et al. (2018). For ConceptNet, we use a maximum learning rate of $\eta = 1\text{e-}5$ and a warm-up period of 200 minibatches. The learning rate is decayed linearly until the end of training, which lasts for 100k minibatches. All other hyperparameters are the same as for training on the ATOMIC corpus.

3.5 ATOMIC Study

The ATOMIC dataset, released by Sap et al. (2019a), contains 877K tuples covering a variety of social commonsense knowledge around specific event prompts (e.g., "X goes to the store"). Specifically, ATOMIC distills its commonsense in nine dimensions, covering the event's causes (`xIntent`, `xNeed`; e.g., "X needs to drive there"), its effects on the agent (`xAttr`, `xWant`, `xReact`, `xEffect`; e.g., "gets food") and its effect on other direct (or implied) participants (`oWant`, `oReact`, `oEffect`; e.g., "Others will be fed"). More details about ATOMIC can be found in Appendix A.3. For our experiments, ATOMIC events (e.g., "X goes to the store") are phrase subjects, s, the dimension (e.g., `xIntent`) is the phrase relation, r, and the causes/effects (e.g., "to get food") are phrase objects, o. We use the dataset splits from Sap et al. (2019a), resulting in 710k training, 80k development, and 87k test tuples.

3.5.1 Setup

Metrics Following Sap et al. (2019a), we evaluate our method using BLEU-2 as an automatic evaluation metric. We also report the perplexity of the model on its gold generations. The remaining automatic metrics in Table 3.2 measure the proportion of generated tuples and generated objects which are not in the training set. We report the proportion of all generated tuples that are novel (% N/T sro) and that have a novel object (% N/T o).[1] To show that these novel objects are diverse (i.e., the same novel object is not the only one being

[1] a new o represents a new node in the knowledge graph

generated), we also report the number of novel objects as a function of the set of *unique* objects produced for all test set events (% N/U o). Our purpose in focusing on novelty is to evaluate whether we are able to hypothesize new commonsense knowledge tuples, or are merely recycling tuples from the training set. Similarly, our goal in evaluating unique objects o is to illustrate the advantge of our method over knowledge base completion methods (§3.2.2), which cannot discover new entities.

Finally, we perform a human evaluation using workers from Amazon Mechanical Turk (AMT). Workers are asked to identify whether a model generation of ATOMIC-style commonsense knowledge adequately completes a plausible tuple of phrase subject, relation, and phrase object. Following the setup of Sap et al. (2019a), we evaluate 100 randomly selected events from the test set. As each event is paired with 9 relations, this yields 900 total (event, relation) pairs. For each event and relation type, 10 candidate endings are generated using beam search and the full beam is evaluated by five different workers. Overall, n=5000 ratings are produced (100 events $\times$ 5 workers $\times$ 10 candidates) for each relation. We measure accuracy in Table 3.1 as the percentage of distinct worker responses where the (event, relation, completion) tuple is marked as valid (i.e., $\frac{\#valid}{5}$). The **Avg** column is an average of the accuracies for all 9 relations, and is thus computed from n=45000 ratings. We use Pitman's test (Noreen, 1989) with 100k permutations to test for statistical significance. Because 50 different hypotheses are tested (9 relations + the total), the Holm-Bonferroni method (Holm, 1979) is used to correct significance thresholds. Example events from the development set and their generated phrase objects are available in Table 3.3.

Baselines We report the performance of our method against baselines presented in Sap et al. (2019a). These baselines include a nearest neighbor baseline (NEARESTNEIGHBOR in Table 3.2) that randomly selects a paired object from the closest subject in the training set. The remaining baselines use LSTM sequence-to-sequence models (Sutskever et al., 2014) to encode the input subject and relation and decode an output object word-by-word. These models differ in the way they segment the relations for training the encoder, but always train a separate decoder for each relation type. The 9ENC9DEC baseline initializes a separate encoder for each relation and trains each pair of encoder and decoder on the examples in the training set corresponding to that relation. The remaining baselines, Event2(IN)VOLUN, Event2PERSONX/Y, and Event2PRE/POST, use the same encoder for various groupings of the relations, which can be found in Table A.2.

Model	oEffect	oReact	oWant	xAttr	xEffect	xIntent	xNeed	xReact	xWant	Avg
9ENC9DEC	22.92	32.92	35.50	52.20	47.52	51.70	48.74	63.57	51.56	45.32
Event2(IN)VOLUN	26.46	36.04	34.70	52.58	46.76	61.32	49.82	71.22	52.44	47.93
Event2PERSONX/Y	24.72	33.80	35.08	52.98	48.86	53.93	54.05	66.42	54.04	46.41
Event2PRE/POST	26.26	34.48	35.78	52.20	46.78	57.77	47.94	72.22	47.94	46.76
COMET (- pretrain)	25.90	35.40	40.76	48.04	47.20	58.88	59.16	64.52	65.66	49.50
COMET	**29.02**	**37.68**	**44.48**	**57.48**	**55.50**	**68.32**	**64.24**	**76.18**	**75.16**	**56.45**

Table 3.1: Human score of ATOMIC generations. We present comparisons to the baselines from Sap et al. (2019a). Underlined results are those where COMET is not significantly better at $p < 0.05$

3.5.2 Results

Overall performance The result of the human evaluation, shown in Table 3.1, shows COMET reporting a statistically significant relative **Avg** performance increase of 18% over the top baseline, Event2IN(VOLUN). This performance increase is consistent, as well, with an improvement being observed across every relation type. We note specifically, the large relative improvements for relations such as `xWant` (39.08%), `oWant` (24.32%), `xNeed` (18.85%), `xEffect` (13.59%), `xIntent` (11.42%) which tend to require generating longer sequences of text than relations such as `xReact, xAttr, oReact,` which can often be single words (see Table A.1 in the appendix for example completions from the training set for each relation type).

While automatic evaluation metrics often do not correlate strongly with human quality assessments of text generation results (Liu et al., 2016b; Nema and Khapra, 2018; Yang et al., 2018a), the BLEU-2 results in Table 3.2 support the observations of the human evaluation. COMET exceeds the performance of all baselines, achieving a 51% relative improvement over the top performing model of Sap et al. (2019a).

Novelty In addition to the quality improvements, Table 3.2 shows that COMET produces more novel tuple objects (i.e., endings) than the baselines. While the number of novel tuples is 100% for all methods due to the dataset splits, COMET produces more novel endings both as a function of the number of total novel endings generated, and the number of unique novel endings generated. **N/U** o is an important measure of novelty because many short endings (particularly those associated with emotions e.g., *happy, sad, content*), which make up a large number of the total, will rarely be novel. As a result, the generated endings for the `oReact, xReact, xAttr` relations are less likely to be novel. The high number of novel endings as a function of unique endings is promising, as it indicates that we are generating a diverse set of novel endings.

Model	PPL[2]	BLEU-2	N/T sro[3]	N/T o	N/U o
9ENC9DEC (Sap et al., 2019a)	-	9.63	100.00	8.61	40.77
NEARESTNEIGHBOR (Sap et al., 2019a)	-	6.92	-	-	-
Event2(IN)VOLUN (Sap et al., 2019a)	-	9.39	100.00	9.52	45.06
Event2PERSONX/Y (Sap et al., 2019a)	-	8.95	100.00	8.22	41.66
Event2PRE/POST (Sap et al., 2019a)	-	9.55	100.00	7.38	41.99
COMET (- pretrain)	15.42	13.19	100.00	7.25	45.71
COMET	**11.14**	**14.43**	100.00	**9.71**	**51.20**

Table 3.2: Automatic evaluations of quality and novelty for ATOMIC generations. No novelty scores are reported for the NearestNeighbor baseline because all retrieved sequences are in the training set.

However, the lower number of novel endings as a function of total endings (**N/T** o) implies that diversity may be an issue among all endings, as a small number of known endings is generated the most often. Regardless, the ability to generate novel endings validates our approach over commonsense knowledge base completion methods that would not be able to produce new nodes for the graph.

Qualitative Analysis In Table 3.3, we list example generations from COMET produced from examples in the development set. These examples were randomly chosen from a subset of novel generated tuples. In general, we see that the model is able to greedily decode strong tuple objects for the subjects and relations it is given. Most of the tuples are marked plausible by human annotators. These assessments extend to generating a larger number of candidates as well. In Figure 3.4, we show the result of providing the event "PersonX gives PersonY a pep talk" to COMET and having it generate 5 candidate endings for each relation, most of which end up forming plausible knowledge tuples related to this event.

Shortcomings include the fact that lexical cues still play a role in the commonsense knowledge hypotheses that are generated. In the example tuple, ("PersonX spoils somebody rotten", `xIntent`, "to be mean") in Table 3.3, it is clear that COMET is misinterpreting the intent of "[spoiling] someone rotten," which is generally associated with being generous. Instead, the model is likely picking up on lexical associations of the words "spoil" and "rotten" to make its prediction.

Also, COMET struggles with metaphorical language, such as in the generated tuple ("PersonX pisses on PersonY's bonfire", `oEffect`, "gets burned"). The idiom "[pissing] on someone's bonfire" means to

[2] Sap et al. (2019a)'s models were trained with a different vocabulary so a direct perplexity comparison is not possible.

[3] All test set s do not appear in the training set so all full tuples must be novel.

Seed Concept	Relation	Generated	Plausible
X holds out X's hand to Y	xAttr	helpful	✓
X meets Y eyes	xAttr	intense	✓
X watches Y every ____	xAttr	observant	✓
X eats red meat	xEffect	gets fat	✓
X makes crafts	xEffect	gets dirty	✓
X turns X's phone	xEffect	gets a text	
X pours ____ over Y's head	oEffect	gets hurt	✓
X takes Y's head off	oEffect	bleeds	✓
X pisses on Y's bonfire	oEffect	gets burned	
X spoils somebody rotten	xIntent	to be mean	
X gives Y some pills	xIntent	to help	✓
X provides for Y's needs	xIntent	to be helpful	✓
X explains Y's reasons	xNeed	to know Y	✓
X fulfills X's needs	xNeed	to have a plan	✓
X gives Y everything	xNeed	to buy something	✓
X eats pancakes	xReact	satisfied	✓
X makes ____ at work	xReact	proud	✓
X moves house	xReact	happy	✓
X gives birth to the Y	oReact	happy	✓
X gives Y's friend ____	oReact	grateful	✓
X goes ____ with friends	oReact	happy	✓
X gets all the supplies	xWant	to make a list	✓
X murders Y's wife	xWant	to hide the body	✓
X starts shopping	xWant	to go home	✓
X develops Y theory	oWant	to thank X	✓
X offer Y a position	oWant	to accept the job	✓
X takes ____ out for dinner	oWant	to eat	✓

Table 3.3: Generations that were **randomly selected** from a subset of **novel** generations from the ATOMIC development set. A novel generation is a sro tuple not found in the training set. Manual evaluation of each tuple indicates whether the tuple is considered plausible by a human annotator. We prune "Person" prefixes on X and Y in the presented examples.

discourage someone. The model generates a much more literal answer. Similarly, the subject of the example ("PersonX takes PersonY's head off", oEffect, "bleeds") also has metaphorical connotations that are not recognized by COMET, but this is marked plausible due to the literal interpretration of the tuple conveniently being correct too.

One of the most interesting properties of COMET, however, is that COMET can handle names even though it was only trained on examples from ATOMIC that have a templatized "PersonX" as the main character. In Figures 3.5, and 3.6, we give complete sentences such as "Tom asked Jessica if he could use her car" as input to COMET. In these cases, we see that COMET is able to generate coherent hypotheses of commonsense knowledge despite never having seen names in the knowledge graph, highlighting the strength of using a pretrained language model to seed COMET. We next explore this observation empirically.

Learning knowledge from language To quantitatively evaluate how pre-training on a large corpus helps the model learn to produce knowledge, we also train a version of COMET that is not initialized with pretrained weights learned from language – COMET (- pretrain) in Tables 3.1 and 3.2. Instead, this model is initialized with random weights and trained on the ATOMIC knowledge graph from scratch.

We observe a 12% relative drop in overall human performance when training from scratch. This difference confirms that the language representations learned by the GPT model are transferable to generating natural language commonsense knowledge. Additionally, we observe that novelty scores are lower across the board compared to COMET, indicating that language pretraining also allows COMET to generate more diverse knowledge tuples. We note that training a transformer-based model on ATOMIC (COMET without pretraining) performs better than LSTM-based baselines in term of **Avg**, implying that the transformer is a more expressive model than the LSTM. However, this result is not consistent across all relation types, highlighting the importance of learning from language.

Importantly, transferring representations learned from language to hypothesizing knowledge tuples is not without risk. Biases which are common in learned language representations (Sheng et al., 2019) could be reflected in the examples of knowledge put forth by COMET. For example, the most likely hypothesized piece of knowledge for the oWant relation for the event "John buys an assault rifle" is "none." If we modify the event to "Jamal buys an assault rifle," the most likely tuple ending becomes "to fire at personx" (where in this case personx corresponds to "Jamal"), outlining a much more negative belief about others' reactions

COMET Decoding method	oEffect	oReact	oWant	xAttr	xEffect	xIntent	xNeed	xReact	xWant	Avg
Top-5 random sampling (n=2500)	34.60	44.04	35.56	64.56	55.68	58.84	46.68	80.96	58.52	53.27
Top-10 random sampling (n=5000)	25.20	37.42	27.34	49.20	47.34	47.06	38.24	72.60	48.10	43.61
Beam search - 2 beams (n=1000)	43.70	54.20	47.60	**84.00**	51.10	73.80	50.70	85.80	78.70	63.29
Beam search - 5 beams (n=2500)	37.12	45.36	42.04	63.64	**61.76**	63.60	57.60	78.64	68.40	57.57
Beam search - 10 beams (n=5000)	29.02	37.68	44.48	57.48	55.50	68.32	64.24	76.18	75.16	56.45
Greedy decoding (n=500)	**61.20**	**69.80**	**80.00**	77.00	53.00	**89.60**	**85.60**	**92.20**	**89.40**	**77.53**
Human validation of gold ATOMIC	84.62	86.13	83.12	78.44	83.92	91.37	81.98	95.18	90.90	86.18

Table 3.4: Human evaluation testing effect of different decoding schemes on candidate tuple quality. The number of ratings n made per relation for each decoding method is provided in the first column.

to this event. Because the ATOMIC knowledge graph contains no information about names, this change in prediction must be due to patterns learned during language pretraining. The difficulty in quantifying the extent of these biases in language models and contextualized representations (active areas of research in natural language processing) makes it especially important to be aware of the ways they might contaminate the types of knowledge predicted by COMET.

Effect of decoding algorithm Because the ultimate goal of our method is to be able to perform high-quality, diverse knowledge base construction, we explore how various decoding schemes affect the quality of candidate knowledge tuples. We present the effect of the following generation strategies: argmax greedy decoding, beam search with beam sizes, b=2, 5, 10, and top-k sampling with k = 5, 10.

For each decoding method, we conduct the human evaluation on the number of final candidates produced by each method. For Beam-2/5/10 and top-5/10 sampling generations, we used the model to generate 2, 5, or 10 (respectively) possible completions per (event, relation) pair. Workers were shown the full set and asked to select all of the completions that were valid for the (event, relation) pair. Each set of tuples was rated by 5 workers. For greedily-decoded generations, we used the model to generate one possible completion per (event, relation) pair. Workers were shown the completed knowledge tuple (event, relation, completion) and asked whether it is valid or not. Once again, each tuple was rated by 5 workers.

In Table 3.4, we show the effect of different generation policies on knowledge quality. The most interesting result is that using greedy decoding to produce knowledge tuples only results in a 10% relative performance gap compared to a human evaluation of the ATOMIC test set, showing that the knowledge produced by the model approaches human performance when only its most confident predictions are used. While

producing more total candidates does lower overall performance, quality assessments still hover around 55% for a beam size of 10. However, this number is partially low due to the many "none" generations for the oEffect, oReact, oWant relations as "none" is often a correct answer for events which do not include other participants besides PersonX (e.g., "PersonX brushes their teeth"). In any set of 10 candidates, "none" can only be predicted once, which causes most candidates in the beam to be incorrect if "none" is the appropriate answer. We could ignore candidates that are less likely than "none," when computing these metrics, but that would deviate from the experimental setup designed by Sap et al. (2019a). However, we note that when evaluating 10 candidates only on relations associated with PersonX, performance jumps to 66%. These results suggest that COMET could be effective with human evaluators in the loop to confirm the correctness of generated tuples, but that more work is needed to accurately generate more candidate tuples for constructing large commonsense knowledge graphs.

% training data	PPL	BLEU-2	N/T *o*	N/U *o*
1% training data	23.81	5.08	7.24	49.36
10% training data	13.74	12.72	**9.54**	**58.34**
50% training data	11.82	13.97	9.32	50.37
FULL Training (no pretraining)	15.18	13.22	7.14	44.55
FULL Training	**11.13**	**14.34**	9.51	50.05

Table 3.5: Effect of amount of training data on automatic evaluation of ATOMIC generations

Data Efficiency Because not all domains will have large available commonsense KBs on which to train, we explore how varying the amount of training data available for learning affects the quality and novelty of the knowledge that is produced. Our results in Table 3.5 indicate that even with only 10% (∼71k examples) of the available training data, the model is still able to produce generations that are coherent, adequate, and novel. Using only 1% (∼7k examples) of the training data clearly diminishes the quality of the produced generations, with significantly lower observed results across both quality and novelty metrics. Interestingly, we note that training the model without pre-trained weights performs comparably to training with 10% of the seed tuples, quantifying the impact of using pre-trained language representations.

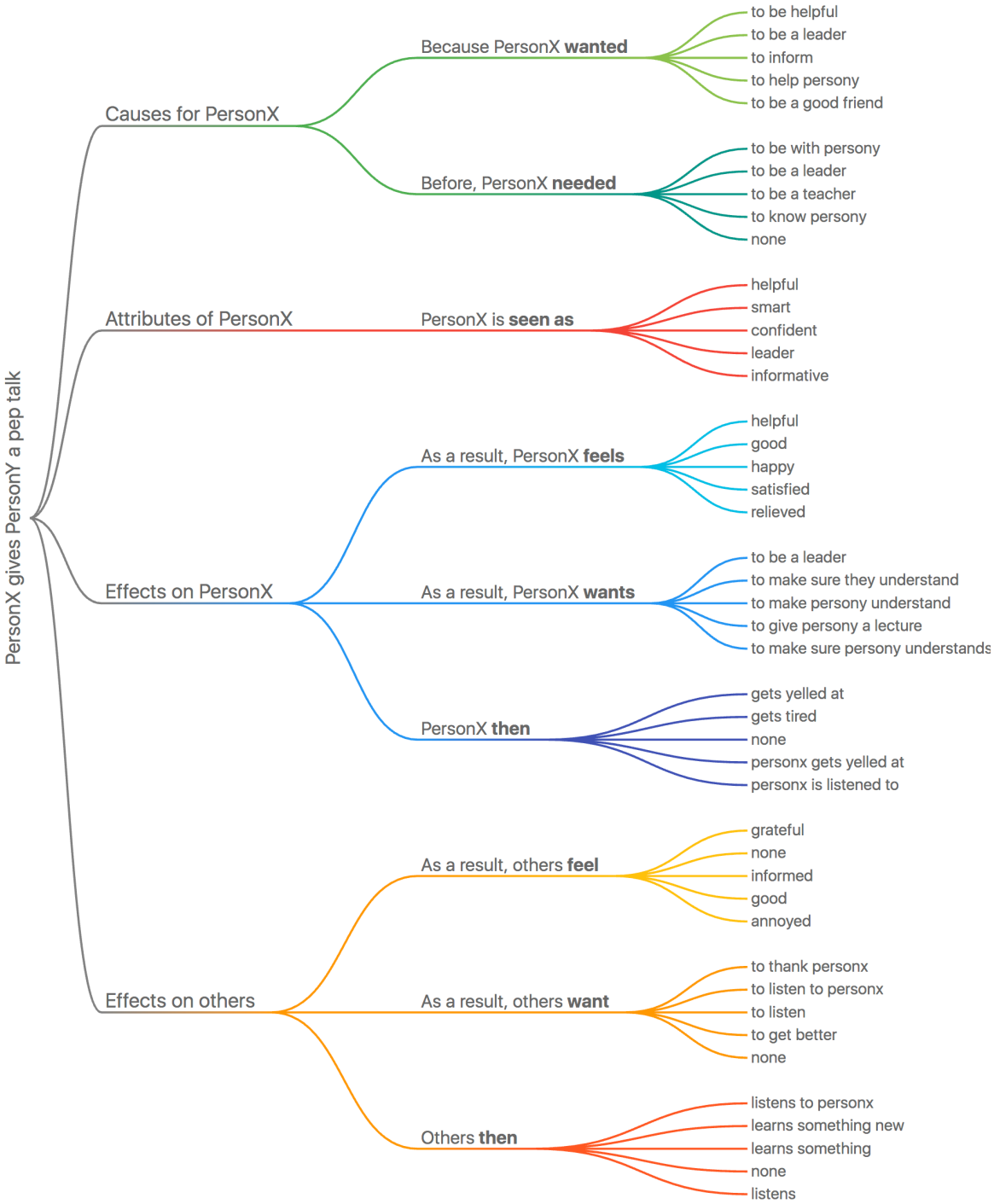


Figure 3.4: Example outputs for the event "PersonX gives PersonY a pep talk" from COMET trained on the ATOMIC knowledge graph. Generations from a demo of COMET at https://mosaickg.apps.allenai.org

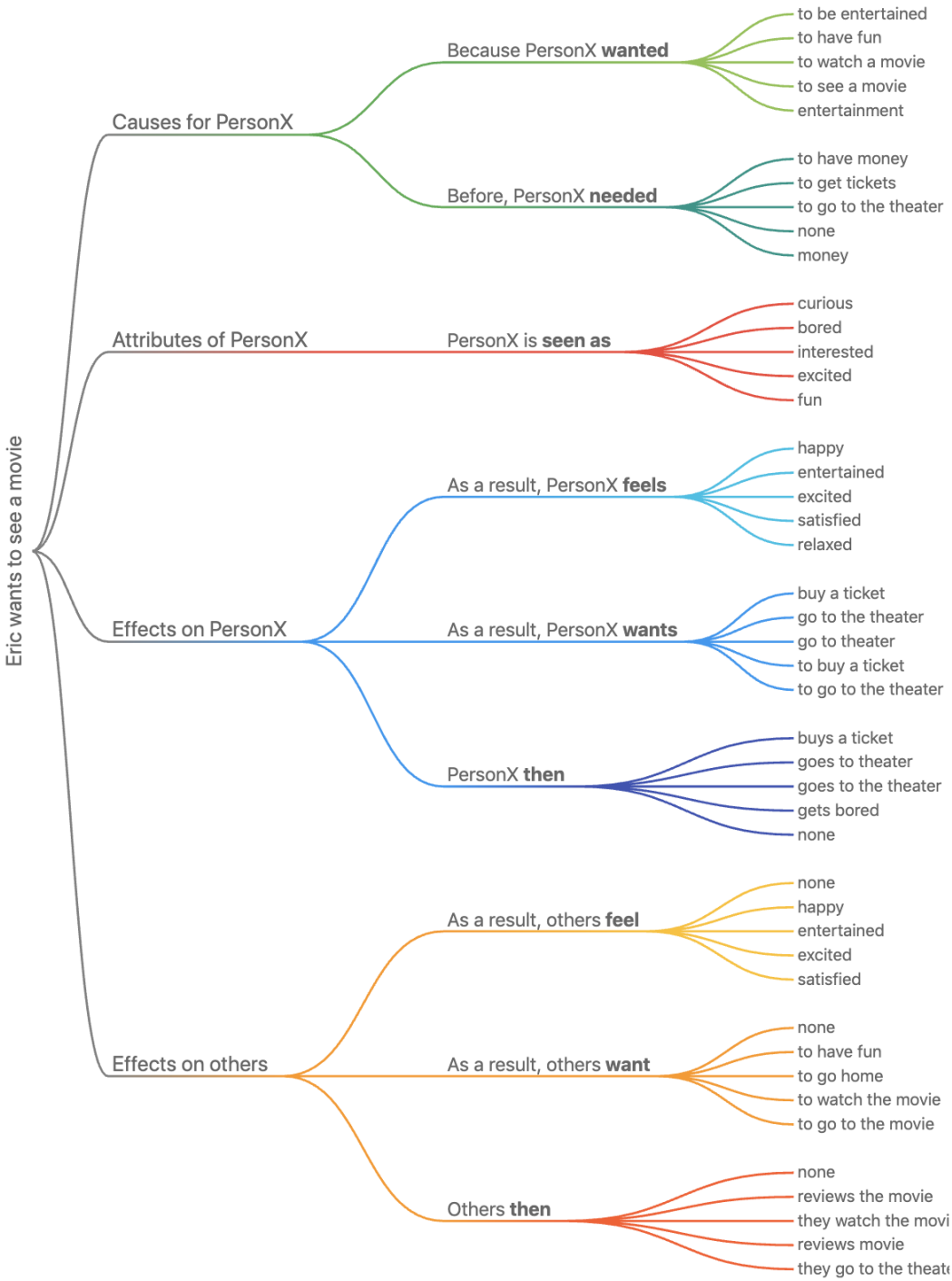


Figure 3.5: Example outputs for the event "Eric wants to see a movie" from COMET trained on the ATOMIC knowledge graph. COMET is able to generalize beyond the templates of the ATOMIC knowledge graph (i.e., PersonX) and can be used directly with names. Generations from a demo of COMET at https://mosaickg.apps.allenai.org

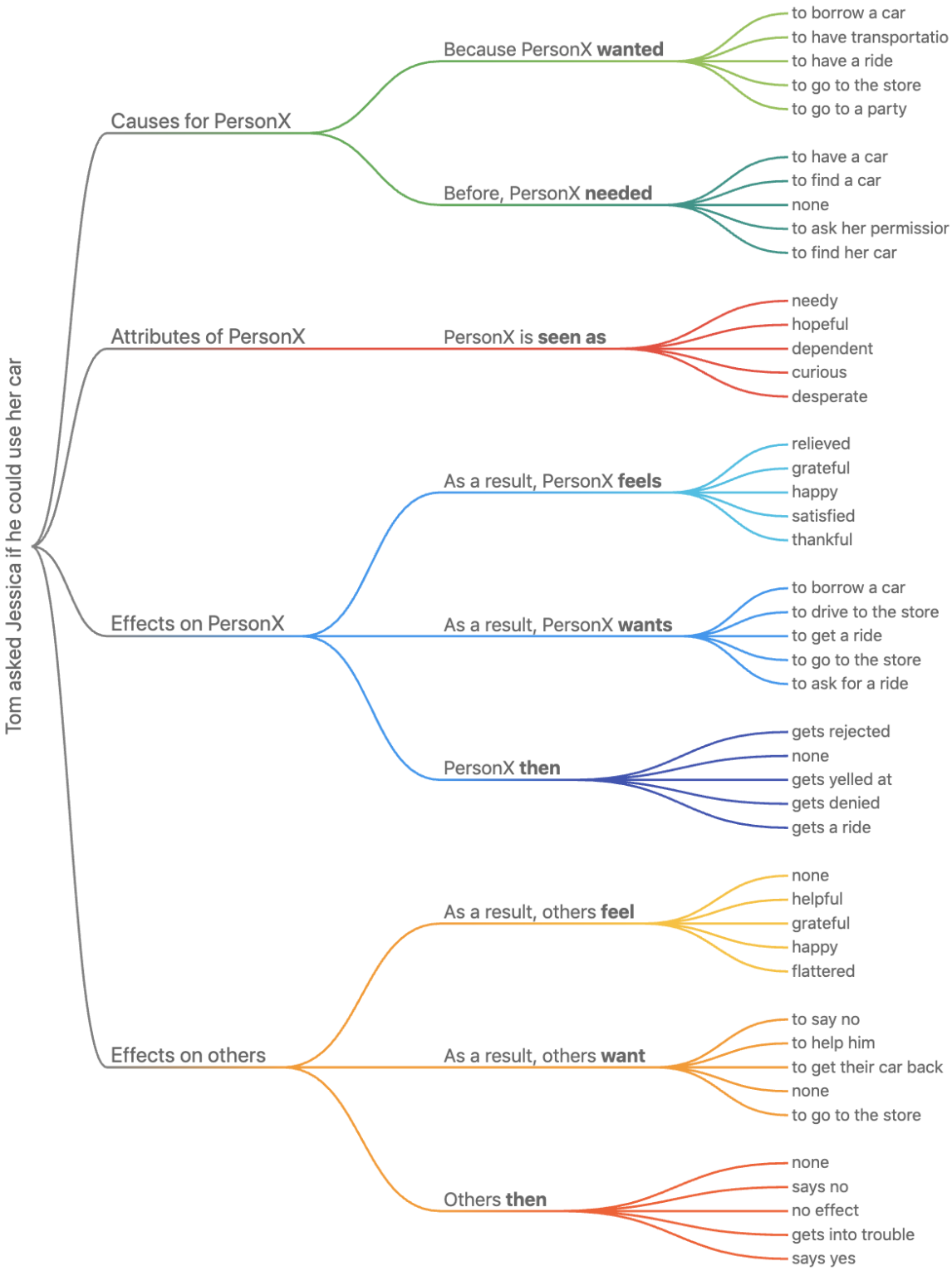


Figure 3.6: Example outputs for the event "Tom asked Jessica if he could use her car" from COMET trained on the ATOMIC knowledge graph. Generations from a demo of COMET at https://mosaickg.apps.allenai.org

3.6 ConceptNet Study

The ConceptNet dataset[4] we used, provided by Li et al. (2016b), consists of tuples obtained from the Open Mind Common Sense (OMCS) entries in ConceptNet 5 (Speer et al., 2017). These entries were crowd-sourced as natural language statements, and then converted to tuples using a preset list of 34 relations. Tuples are in the standard *sro* form – (e.g., "take a nap", `Causes`, "have energy"). The most confident 1200 tuples were used to create the test set, while the next 1200 tuples were used to create two development sets, which we combine for this experiment. The 100k version of the training set was used to train models. Unlike ATOMIC, which focused on social commonsense knowledge related to the motivations, desires, and emotions of people, ConceptNet does not limit the source of commonsense knowledge. As a result, tuples are drawn from a larger space of commonsense knowledge varieties. Example generated tuples for ConceptNet in Table 3.7 illustrate the different structure and theme of ConceptNet tuples relative to tuples in ATOMIC, which were depicted in Table 3.3.

3.6.1 Setup

Metrics We evaluate models that generate knowledge tuples for ConceptNet relations using the following metrics. First, we report the perplexity of the gold relations in the test set (PPL). To evaluate the quality of the generated knowledge tuples, we also report the number of generated positive examples in the test set that are scored as correct by the pre-trained Bilinear AVG model developed by Li et al. (2016b).[5] For a given *sro* tuple, this model produces a probability for whether the tuple is correct. We threshold scores at 50% probability to identify positive predictions. On the completion task originally proposed in Li et al. (2016b), this model achieved 92.5% accuracy on the test set, indicating that it is a suitable proxy for automatically evaluating whether a generated tuple is correct. Finally, we report the same novelty metrics as for ATOMIC: **N/T** *sro* and **N/T** *o*, which correspond to the number of novel tuples and novel endings generated.

Baselines As a baseline, we re-implement the BiLSTM model proposed by Saito et al. (2018) with minor modifications outlined in Appendix A.1.1. This model is trained to learn to encode knowledge in both

[4] https://ttic.uchicago.edu/~kgimpel/commonsense.html

[5] A pre-trained model can be found at https://ttic.uchicago.edu/~kgimpel/comsense_resources/ckbc-demo.tar.gz

Model	PPL	Score	N/T sro	N/T o	Human
LSTM - s	-	60.83	**86.25**	7.83	63.86
CKBG (Saito et al., 2018)	-	57.17	**86.25**	**8.67**	53.95
COMET (- pretrain)	8.05	89.25	36.17	6.00	83.49
COMET - RELTOK	4.39	95.17	56.42	2.62	**92.11**
COMET	**4.32**	**95.25**	59.25	3.75	91.69

Table 3.6: Automatic and human evaluation of ConceptNet generations for test set tuple prefixes. In all, n=1200 tuples are evaluated)

directions: $sr \rightarrow o$ and $or \rightarrow s$ to help augment a knowledge base completion model. It is only evaluated on the $sr \rightarrow o$ tuple generation task, however. We also include the result from a LSTM model that is only trained on the $sr \rightarrow o$ task (LSTM - s).

3.6.2 Results

Quality Our results indicate that high-quality knowledge tuples can be proposed by the model: the low perplexity scores in Table 3.6 indicate high model confidence in its predictions, while the high classifier score (95.25%) indicates that the commonsense knowledge base completion model of Li et al. (2016b) scores the generated tuples as correct in most of the cases. While adversarial generations that learn to exploit the classifier could be responsible for this high score, a human evaluation (following the same design as for ATOMIC) scores 91.7% of greedily decoded tuples as being correct. Furthermore, we note that neither of our baselines trained using LSTMs approach the performance of COMET on the automated score or the human score of generated tuples.

Randomly selected examples provided in Table 3.7 also point to the quality of knowledge produced by the model. As with ATOMIC, we randomly select novel generated tuples from the ConceptNet development set. For most examples, COMET is able to generate a tuple object that is relevant to the subject with respect to the relation. These pieces of knowledge range from taxonomic knowledge such as ("math", `IsA`, "subject"), to affordances ("doctor", `CapableOf`, "save life"), to simple preconditions ("play game", `HasProperty`, "have game"), and even to existential realities, ("live", `HasLastSubevent`, "die").

Seed	Relation	Completion	Plausible
piece	PartOf	machine	✓
oldsmobile	IsA	car	✓
happiness	IsA	feel	✓
math	IsA	subject	✓
mango	IsA	fruit	✓
maine	IsA	state	✓
planet	AtLocation	space	✓
dust	AtLocation	fridge	
puzzle	AtLocation	your mind	🤔
dental chair	AtLocation	dentist	✓
finger	AtLocation	your finger	
sing	Causes	you feel good	✓
doctor	CapableOf	save life	✓
post office	CapableOf	receive letter	✓
dove	SymbolOf	purity	✓
sun	HasProperty	big	✓
bird bone	HasProperty	fragile	✓
earth	HasA	many plant	✓
yard	UsedFor	play game	✓
get pay	HasPrerequisite	work	✓
print on printer	HasPrerequisite	get printer	✓
play game	HasPrerequisite	have game	✓
live	HasLastSubevent	die	✓
swim	HasSubevent	get wet	✓
sit down	MotivatedByGoal	you be tire	✓
all paper	ReceivesAction	recycle	✓
chair	MadeOf	wood	✓
earth	DefinedAs	planet	✓

Table 3.7: Randomly selected and novel generations from the ConceptNet development set. Novel generations are sro tuples not in the training set. Manual evaluation of each tuple indicates whether the tuple is considered plausible by a human annotator

Novelty In addition to being high quality, it is important for generated tuples to be novel, so that we are not merely generating pieces of knowledge that were already in the training set. We observe that 59.25% of the tuples are not present in the training set, showing that the model is capable of generating new edges between nodes, and even creating new nodes – 3.75% of o nodes are novel – to extend the size of the knowledge graph. One shortcoming, however, is that novel generations are sometimes simplified forms of tuples from the training set. In Table 3.7, for example, the tuple ("doctor", `CapableOf`, "save life") is not present in the training set, but ("doctor", `CapableOf`, "save person life") is. Many tuples, however, are completely novel, such as ("bird bone", `HasProperty`, "fragile") and ("driftwood", `AtLocation`, "beach"), which have no related tuples in the training set.

To explore further, we investigate by how much novel tuples from the development set differ from training set phrase objects for the same s, r using minimum edit distance of phrase objects. We measure the edit distance of phrase object o_{dev} in the tuple (s, r, o_{dev}) to the o_{trn} from the nearest training tuple (s, r, o_{trn}). Edit distance is measured using word tokens (excluding stop words) and normalized by the larger of the number of words in o_{dev} or o_{trn}. The maximum edit distance is one (i.e., entirely different word sequences) and the minimum edit distance is zero (i.e., the same sequence excluding stopwords). Figure 3.7 shows the percentage of novel development set tuples that have an edit distance from the closest training set tuple of at least the value on the x-axis. Over 75% of the novel tuples have objects that are a normalized edit distance of ≥ 0.5 from the training phrase objects, indicating that most of the novel phrase objects have significantly different word sequences from their closest analogues in the training set.

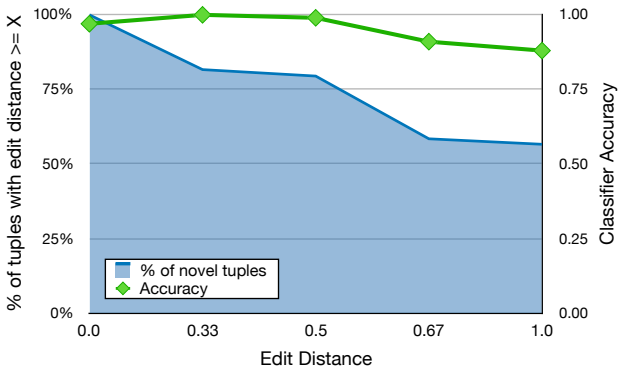


Figure 3.7: The percentage of novel ConceptNet development set tuples per minimum edit distance from training tuples. In green: classifier-scored accuracy of each subset.

Learning knowledge from language Similarly to ATOMIC, we explore how initializing COMET with a pretrained language model (Radford et al., 2018) affects its ability to generalize commonsense knowledge. We train comparison model, COMET (- pretrain), that is initialized from scratch, rather than with a pretrained

language model. The importance of language pretraining is apparent in Table 3.6, with a clear improvement on automatic and human evaluations (relative 10.3%) by the pretrained COMET over the randomly initialized model. Qualitatively, we observe this effect, as well. For example, in Table 3.7, the generated example tuple ("mango", IsA, "fruit") is not present in the training set. The only tuple containing the "mango" entity in the training set is ("mango", UsedFor, "salsa"), which is not informative enough to learn this property from scratch. As confirmation, we observe that the output from COMET (- pretrain) is ("mango", IsA, "spice"), which, while incorrect, would be a reasonable inference given the information about "mango" in the seed set of knowledge available for training.

Representing relations with language As shown in Figure 3.3, we map ConceptNet relation names to natural language (e.g., IsA $\rightarrow$ "is a"; HasSubevent $\rightarrow$ "has subevent") when we input the relation to the model. This decision allows the model to learn to represent these concepts from language, as opposed to learning a special embedding for each relation from scratch (Levy et al., 2017). To evaluate whether letting the model learn relations as natural language terms is helpful for learning its implications, we train a model without converting relation tokens to natural language (e.g., IsA $\not\rightarrow$ "is a"), which we denote COMET - RELTOK.

While the automatic metrics point to insignificant differences when comparing models with symbol relations and those with natural language relations (Table 3.6), examples can provide qualitative insights into the benefits of representing relations as language. For example, the only non-ornithological (i.e., not related to bird taxonomies) reference to a "dove" in the ConceptNet training set is ("dove", CapableOf, "fly"). However, our model learns to generalize to produce the tuple ("dove", SymbolOf, "purity"). The model that uses symbol relation embeddings only manages to produce the relation ("dove", SymbolOf, "submarine"), which seems to relate "submarine" to the nautical (and unrelated) word sense of "dove". We note that this capability could be particularly useful for low-coverage relations with few training examples. Learning a relation embedding from scratch for these cases (such as SymbolOf, which has only 159 training examples) could be more challenging.

Conversely, for high-coverage relations, mapping relations to language can have unforeseen consequences. As an example, for the relation IsA, that has over 15k training examples, the pretrained language model's understanding of other senses of "is" can lead to generated tuples such as ("go to mall", IsA, "good

idea"). In this example, the taxonomic nature of the IsA relation is superseded by the language model's polysemous understanding of its component words.

3.7 Summary

In this chapter, we introduce COMmonsense Transformers (COMET) for automatic construction of commonsense knowledge bases. COMET is a framework for adapting the weights of language models to learn to produce novel and diverse commonsense knowledge tuples. Empirical results on two commonsense knowledge bases, ATOMIC and ConceptNet, show that COMET frequently produces novel commonsense knowledge that human judges evaluate as plausible. Importantly, we find significant benefits to learning to transfer neural representations that have been pretrained on language to the task of generating knowledge graph tuples. Both quantitative and qualitative results show the influence that knowledge learned from language has on the forms of knowledge that can be queried from the model.

The implications of these observations extend beyond constructing large-scale commonsense knowledge bases, however. In essence, our work shows that pretrained language models already implicitly encode a commonsense knowledge base neurally. Training on a seed set of examples of knowledge allows an interface to this knowledge to be learned, yielding a queryable *neural representation* of a knowledge base. We show that this interface allows us to extract high-quality commonsense knowledge for even rich compositional contexts that deviate considerably from the format of knowledge found in the knowledge graph.

These positive results point to multiple potential streams of future research, such as extending the approach to other types of knowledge bases, investigating whether COMET can learn to induce new types of relations from language, researching how we can reliably access the knowledge already encoded in large-scale language representations for downstream NLP tasks, and looking into whether we can use COMET for dynamic commonsense extraction in real-time when presented with situations described in language.

Chapter 4

Commonsense Reasoning with Dynamic Knowledge Graph Construction

Understanding narratives requires reasoning about implicit world knowledge related to the causes, effects, and states of situations described in text. At the core of this challenge is how to access contextually relevant knowledge on demand and reason over it. In this chapter, we present studies in commonsense question answering that approach the task by reasoning over dynamically-generated contextual commonsense knowledge graphs. Previous studies that integrate knowledge into systems for natural language understanding rely on *retrieval* of existing knowledge from large *static* knowledge graphs that are pre-curated. However, contextually-relevant knowledge is often *not* present or easily accessible in these existing knowledge bases. Therefore, we present a novel approach that *generates* contextually-relevant symbolic knowledge structures on demand using *generative neural commonsense knowledge models* introduced in Chapter 3. Empirical results on two datasets demonstrate the efficacy of our neuro-symbolic approach for dynamically constructing knowledge graphs for reasoning. Our approach achieves relative performance boosts of 23.8% over pretrained language models, and 7.3% over vanilla knowledge models.

The material in this chapter is adapted from Bosselut and Choi (2019)

4.1 Introduction

Understanding narratives requires reasoning about all the implicit, but trivially inferable, details of a situation based only on what is explicitly stated in text. A statement as simple as "they went to the club" instantly invokes a bank of commonsense expectations: that they had to get dressed, that they might be going dancing, that they likely had drinks, and so forth. These reasoning capabilities are missing in most existing neural language understanding models that learn task-specific representations without acquiring rich background knowledge about the social and physical world.

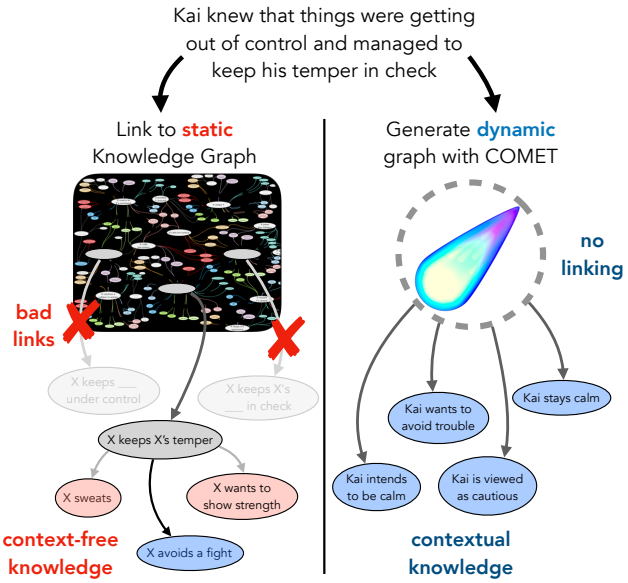


Figure 4.1: Previous approaches for accessing knowledge link situational contexts to **static** knowledge graphs. Our work generates knowledge **dynamically** from neural representations of knowledge graphs.

In response, recent work has investigated augmenting deep learning models with retrieval mechanisms over large-scale commonsense knowledge graphs (Mihaylov and Frank, 2018; Bauer et al., 2018; Paul and Frank, 2019). However, these approaches assume an entity linking step between the written text and knowledge graph. By canonicalizing entities, they discard key context surrounding the input, and can often retrieve semantically irrelevant knowledge (e.g., the fact that a club is a blunt weapon is irrelevant to the earlier situation).

In this chapter, we define an approach to *generate* new knowledge that is contextually relevant instead of *retrieving* existing knowledge as is. Chapter 3 introduced *Commonsense Transformers* (COMET), a new framework for training neural representations of knowledge graphs. This new class of neural *knowledge model* provides a powerful representational tool for connecting commonsense knowledge to downstream task models. Because COMET represents knowledge graphs neurally, it can generate commonsense inferences for any entity that can be encoded by the neural model (i.e., described with language). With no need to canonicalize context entities to link to a static knowledge graph, the knowledge model can be queried directly with complex compositional structures, and even full narrative contexts.

We use COMET to construct context-relevant knowledge graphs that can be reasoned over for commonsense question answering. Given a raw context, COMET, a neural representation of a knowledge base, generates commonsense inferences that provide world knowledge about the situation depicted in the context. These inferences can be used as additional context to score answer candidates or to generate additional inferences. By generating new inferences and connecting them to the raw context and answers, COMET dynamically constructs a knowledge graph of commonsense knowledge. The raw context is the root node, answer choices are leaf nodes and generated commonsense inferences provide reasoning paths between the context and answers. Using COMET-generated scores as factors weighting these paths, we propose new inference algorithms to reason over the generated graph and identify the most likely answers to questions about the situation.

We evaluate our approach on the SOCIALIQA (Sap et al., 2019b) benchmark, a question answering dataset for evaluating social commonsense, and the STORYCOMMONSENSE benchmark (Rashkin et al., 2018), a story understanding dataset. Empirical results show that our neuro-symbolic approach outperforms purely neural large-scale pretrained language models (Radford et al., 2018, 2019) and COMET models that evaluate QA examples directly without dynamically generating an intermediate symbolic commonsense knowledge graph (i.e., reasoning with COMET with no inference hops). However, our approach falls far short of state-of-the-art supervised methods that are trained on large amounts of task-relevant data.

4.2 Related Work

4.2.1 Question Answering with Knowledge Graphs

Previous work has explored integrating reasoning over static knowledge graphs for question answering and story understanding. In general, these approaches extract knowledge tuples from the static KG by linking canonicalized entities to nodes and performing multi-hop inference along relation paths to form full tuples that can be encoded by a downstream neural architecture (Mihaylov and Frank, 2018; Bauer et al., 2018; Weissenborn et al., 2017; Lin et al., 2019; Paul and Frank, 2019). Similar to our approach of discovering reasoning chains between contexts and answers, Paul and Frank (2019) extract multi-hop reasoning paths in ConceptNet between normalized entities from the context answer candidates, but can only discover paths

through nodes in the static knowledge graph. Finally, there exists works that also dynamically construct latent knowledge graphs (Das et al., 2019; Bosselut et al., 2018a), but these works presuppose a fixed set of entities that can be KG nodes and then approximate graph edges with neural transformations. In contrast, our algorithm can generate arbitrary nodes, thereby dynamically constructing a unique symbolic structure for any example that can be reasoned over with probabilistic inference.

4.2.2 Multi-hop Reading Comprehension

Similar in spirit to reasoning over knowledge graphs for question answering is work in multi-hop reading comprehension. Many datasets for learning to aggregate multiple facts without graph structure have been released in recent years (Weston et al., 2016; Welbl et al., 2018; Yang et al., 2018b; Talmor and Berant, 2018). Approaches designed for these resources generally use large-scale neural networks to attend over supporting facts across text (Zhong et al., 2019; Dhingra et al., 2018). Most similar to our work are approaches that construct real-time entity mention graphs as neural reasoning paths (Cao et al., 2018; Jiang et al., 2019; Jiang and Bansal, 2019; Fan et al., 2019). Our approach differs from these models in that we *generate* relevant supporting information rather than mining it from accompanying documents and conduct our study in a zero-shot setting with no additional training.

4.2.3 Automatic Commonsense Knowledge Graph Construction

Multi-hop reasoning over commonsense inferences requires construction of commonsense knowledge graphs and recent approaches have investigated how to mine commonsense knowledge from deep learning models. Previous work by Sap et al. (2019a), Li et al. (2016b), and Saito et al. (2018) highlighted in Chapter 3 investigated whether LSTM models could generate new tuples for the ATOMIC and ConceptNet knowledge graphs. Jastrzębski et al. (2018) built on these approaches for evaluating novel commonsense knowledge mined from Wikipedia. More recent work mapped candidate commonsense tuples to natural language with templates and used pretrained transformer language models to validate them (Davison et al., 2019; Petroni et al., 2019). Concurrently, other research has explored using pretrained language models and adapting them as generative knowledge graph constructors (Malaviya et al., 2020). In contrast to these works that augment static knowledge graphs, our approach focuses on constructing contextualized knowledge graphs *on demand*

to provide context-dependent commonsense for downstream inference.

4.3 Dynamic Knowledge Graph Construction for Question Answering

Our approach uses a knowledge model, COMET (introduced in Chapter 3), to dynamically construct a context-relevant commonsense knowledge graph about a presented situation. COMET is trained using transfer learning from large-scale pretrained language models (Radford et al., 2018) to knowledge graphs. When trained on the ATOMIC knowledge graph (Sap et al., 2019a), it learns to generate social commonsense inferences of situations depicted in text. Importantly, unlike static knowledge graphs (e.g., ConceptNet; Speer et al., 2017), which require canonicalizing input entities to link to the graph, COMET represents knowledge neurally, allowing it to generate commonsense for arbitrary input forms.

In Figure 4.1, for example, the context "Kai knew things were getting out of control and managed to keep his temper in check" is unlikely to be found in any existing knowledge graph. It describes a very specific situation. However, COMET can parse this full context and generate commonsense knowledge about Kai's reactions and motivations, such as "Kai stays calm" or "Kai wants to avoid trouble," as downstream inferences. We exploit this generalization property of knowledge models to dynamically construct knowledge graphs for presented situations that can be reasoned over to answer commonsense questions about them.

4.3.1 Constructing a Probabilistic Commonsense Knowledge Graph

Notation Formally, we assume a dataset of examples, each with an associated context c describing a situation, a question q asked about that situation, and a set of n possible answers $\mathcal{A} = \{a_1, ..., a_n\}$ to that question. Each answer is made up of multiple tokens $Y^a = \{y_1, ..., y_{|a|}\}$.

Generating Commonsense Inferences We generate commonsense inferences for a situational context c by concatenating the context with relation types from the ATOMIC knowledge graph and using COMET to produce candidates $\mathcal{G}$. Each candidate $g \in \mathcal{G}$ is associated with a score ϕ_g that approximates the model's confidence in the inference:

$$\phi_g = \frac{1}{|g|} \sum_{t=1}^{|g|} \log P(x_t | x_{<t}, c, r) \tag{4.1}$$

where x_t are the tokens of g, $|g|$ is the token length of g, r is an arbitrary commonsense relation type for which $\mathbb{COMET}$ can generate inferences, and:

$$P(x_t|x_{<t}, c, r) = \mathbb{COMET}(c, r, x_{<t}) \tag{4.2}$$

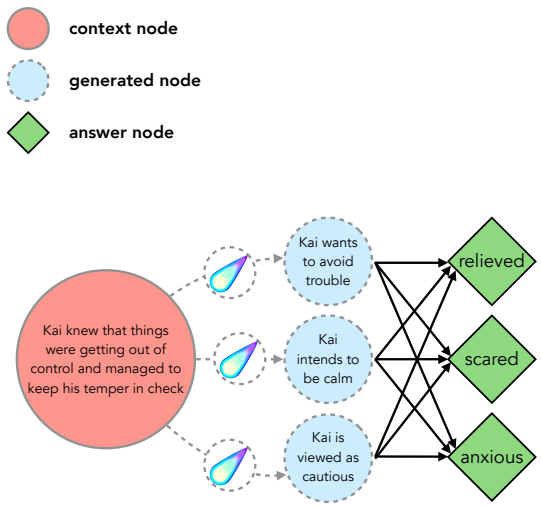
where the tokens of c and r are concatenated with the tokens $x_{<t}$ to be input to $\mathbb{COMET}$. Any generation $g \in \mathcal{G}$ conditioned on c can be seen as a 1-hop commonsense inference of c.

Using a Markov assumption, we can generalize this approach by conditioning on generated commonsense inferences to generate $\mathcal{G}^\ell$, a set of ℓ-hop inferences from c:

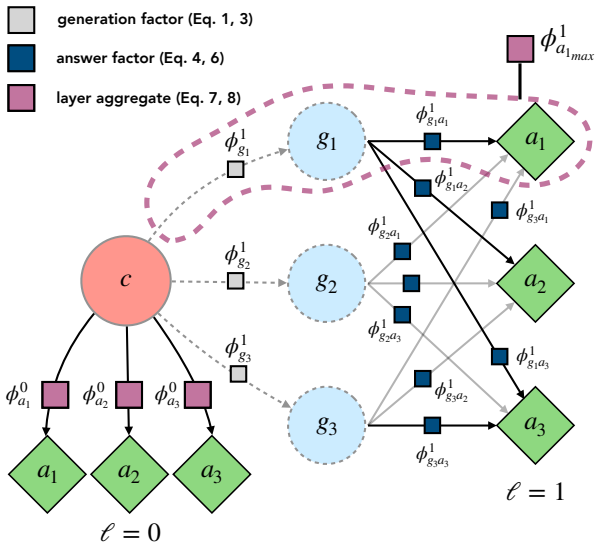
$$\phi_g^\ell = \phi_g^{\ell-1} + \frac{1}{|g^\ell|} \sum_{t=1}^{|g^\ell|} \log P(x_t|x_{<t}, g^{\ell-1}, r) \tag{4.3}$$

where ϕ_g^ℓ is a generation score for any $g^\ell \in \mathcal{G}^\ell$, $g^{\ell-1}$ is an arbitrary inference from $\mathcal{G}^{\ell-1}$, the set of inferences of the previous hop, and $\phi_g^{\ell-1}$ is the generation score of that seed inference. Using this approach, we can use $\mathbb{COMET}$ to construct a graph where commonsense inferences g are nodes. For an arbitrary node g^ℓ, its parent is the node from the previous level $\mathcal{G}^{\ell-1}$ that $\mathbb{COMET}$ conditions on to generate g^ℓ. The children of g^ℓ are nodes generated when $\mathbb{COMET}$ conditions on g^ℓ to generate new commonsense inferences. While using multiple $g^{\ell-1} \in \mathcal{G}^{\ell-1}$ to generate g^ℓ would provide richer subgraph structure on which to condition, the $\mathbb{COMET}$ model is not trained for this purpose and so we assume that the generation of each node g^ℓ is conditioned on a single parent node. We set $g^0 = c$ because the context is the root node of the graph, and $\phi_g^0 = 0$ because the original context c is deterministic.

Answers as Leaf Nodes The final step in constructing the knowledge graph is to connect the answer choices $a \in \mathcal{A}$ to the generated commonsense inferences. We initialize a node in the graph for each answer choice a and connect it as a child node to each commonsense inference in the graph: $g \in \mathcal{G}^\ell$ for $\ell \in [0, L)$ where L is the number of levels in the final graph. In Figure 4.2b, we see that the answer choices $\mathcal{A}$ = {*relieved, scared, anxious*} are connected to the root node and each generated commonsense inference in the $L = 2$ level graph.



(a) COMET receives the context c and generates new commonsense inferences to construct a local graph of knowledge about the situation (Section 4.3).



(b) Our inference algorithms reason over the graph by finding the best combination of commonsense paths to answer questions about the situation (Section 4.4).

Figure 4.2: Our approach consists of dynamically constructing a local commonsense knowledge graph about a presented situation. This graph can be used to reason about the different questions about the situation.

4.4 Reasoning over Probabilistic Knowledge Graphs

Being designed as a conditional language model, COMET can also be used to score candidate commonsense inferences. We use this property to score answer candidates $a \in \mathcal{A}$ conditioned on the generated commonsense inferences $g \in \mathcal{G}$ that are connected to them. The scores from COMET are used to initialize factor nodes between each generated commonsense inference (at all levels of the graph) and each answer choice. Using these scores, and scores between commonsense inferences (Eqs. 4.1, 4.3), as a set of factors, our generated knowledge graph implicitly encodes a factor graph that can be reasoned over to evaluate each answer candidate.

4.4.1 Computing Answer Scores

COMET is originally trained to maximize the conditional log-likelihood of the tokens of a target entity e_2 from a knowledge graph tuple (e_1, r, e_2). As a result, the knowledge model can measure the log-likelihood of a candidate entity e_2 given a source entity e_1 and relation r. For a given example, we treat each answer

candidate a as an e_2 candidate for $\mathbb{COMET}$, map the parent nodes of a (e.g., g nodes) to be equivalent to e_1, and set the question q as r, allowing $\mathbb{COMET}$ to evaluate each answer candidate according to its implicit knowledge representations. For each answer $a \in \mathcal{A}$, we define a factor based on each token's conditional log-likelihood as computed by $\mathbb{COMET}$:

$$\phi_{ga} = \frac{1}{|a|} \sum_{s=1}^{|a|} \log P(y_s | y_{<s}, g, q) \tag{4.4}$$

where y_s corresponds to the token in a at time step s, $y_{<s}$ is all the tokens preceding y_s in a, and $|a|$ is the total number of tokens making up a. In this way, for any QA example, we define a set of factor nodes ϕ_{ga} connecting the answer candidates $a \in \mathcal{A}$ to the commonsense inferences $g \in \mathcal{G}$ generated by $\mathbb{COMET}$ about the situational context c.

Overcoming Answer Priors Because certain answer candidates have a high probability of occurring for certain questions regardless of the context (e.g., *happy* is a common answer for questions about emotional reactions), we redefine ϕ_{ga} (Eq. 4.4) in terms of the point-wise mutual information between the commonsense path g and answer a:

$$\phi_{ga} \propto \text{PMI}(a, g | q) \tag{4.5}$$

$$\phi_{ga} = \frac{1}{|a|} \sum_{s=1}^{|a|} \Big(\log P(y_s | y_{<s}, g, q) - \log P(y_s | y_{<s}, q) \Big) \tag{4.6}$$

where $\log P(y_s | y_{<s}, q)$ is the log-likelihood of each token in the answer given only the question and previous answer tokens. We describe our approximation of this distribution in Section 4.5.2.

4.4.2 Inference

Each ϕ_g^ℓ scores a unique reasoning path at a particular depth ℓ in the graph. The composition $\gamma_g \phi_g^\ell + \gamma_{ga} \phi_{ga}^\ell$ can then be seen as scoring a path to a particular answer. To find the most likely answer, we marginalize

over all paths to the answers at layer ℓ:

$$\phi_a^\ell = f(\{\gamma_g \phi_g^\ell + \gamma_{ga} \phi_{ga}^\ell : g \in \mathcal{G}^\ell\}) \tag{4.7}$$

where ϕ_g^ℓ (Eq. 4.3) and ϕ_{ga}^ℓ (Eq. 4.4) are the *path* and *answer* score, respectively, for generation $g \in \mathcal{G}^\ell$. γ_g and γ_{ga} are hyperparameters balancing the contribution of both scores. Because the path and answer scores are log-probabilities, we set f as the LogSumExp, yielding Eq. 4.7 as a variable elimination over $g \in \mathcal{G}^\ell$.

As the number of levels L grows and the set of generated inferences $\mathcal{G}^\ell$ grows for each level $\ell \in (0, L)$, it can become intractable (and noisy) to compute ϕ_a^ℓ by marginalizing over all generated inferences, so we define an extremum estimator over the distribution of generated inferences $\mathcal{G}^\ell$:

$$\phi_{a_{max}}^\ell = \max_{g \in \mathcal{G}^\ell} \gamma_g \phi_g^\ell + \gamma_{ga} \phi_{ga}^\ell \tag{4.8}$$

At a high level, $\phi_{a_{max}}^\ell$ can be interpreted as approximating the likelihood of answer a given the reasoning path of $\{c \to g^1 \to \cdots \to g^\ell \to a\}$.

Once the answer scores at different levels in the graph are computed, $\{\phi_a^\ell\}_0^L$, the final score for each answer can be evaluated by averaging over the graph levels $\ell \in [0, L)$:

$$\log P(a|q, c) \propto \phi_a = \sum_{\ell=0}^{L} \beta^\ell \phi_a^\ell \tag{4.9}$$

$$\hat{a} = \arg\max_{a \in \mathcal{A}} \phi_a \tag{4.10}$$

where L is the number of generation hops made by the $\mathbb{COMET}$ model (i.e., the number of levels in the graph), ϕ_a^ℓ is the score that is propagated from each hop of the constructed knowledge graph, and β^ℓ is a hyperparameter scaling the contribution of each hop score. We note that ϕ_a^0 is the result from evaluating the answer candidates directly against the original context c, and that ϕ_a^ℓ is replaced by $\phi_{a_{max}}^\ell$ if the extremum estimator (Eq. 4.8) is used instead of variable elimination (Eq. 4.7).

4.5 Experimental Setup

We evaluate our approach in a zero-shot experimental setting. It is a well-studied phenomenon that neural methods trained on crowdsourced data often learn to shortcut reasoning to arrive at a correct answer (Gururangan et al., 2018; Li and Gauthier, 2017; Poliak et al., 2018), yielding representations that are brittle (Jia and Liang, 2017). We use a zero-shot setting to simulate the model having to reason about situations it has never encountered before, forcing it to construct reasoning graphs from explicit knowledge it can generate (e.g., knowledge learned by COMET), and precluding it from learning dataset-specific artifacts. As such, we do not use training data to update model parameters. Furthermore, any result presented on the test set does not have hyperparameters tuned on the development set. As additional analysis, we show ablations with scores from the development sets of these datasets.

4.5.1 Datasets and Processing

We evaluate our method on two datasets: SOCIALIQA (Sap et al., 2019b), a social commonsense question answering dataset, and STORYCOMMONSENSE (Rashkin et al., 2018), a text classification dataset for identifying the motivations and emotions of characters in short stories. For each dataset, we use the development and test sets to report performance.

Dataset	# dev	# test
SOCIALIQA	1952	2217
STORYCOMMONSENSE	202360	182160

Table 4.1: Dataset statistics for SOCIALIQA and STORYCOMMONSENSE

SOCIALIQA The SOCIALIQA dataset evaluates a model's ability to understand the social dynamics underlying situations in short text snippets. Each example in the dataset consists of a context, a question about that context, and three multiple choice answers. An example from the dataset is shown in Figure 4.3. When converting generated inferences to contexts for answer scoring (Eq. 4.4), we add a prefix that is specific to the inference type to the generated tokens (e.g., happy ⇒ Person is happy). The prefixes for each inference type can be found in Table B.1 in Appendix B. Additionally, we prune the list of generated inferences that can be used for reasoning with rules that are presented in Appendix B. For our main models and ablations, names that appear in contexts and answers are

anonymized.

STORYCOMMONSENSE The STORYCOMMONSENSE dataset consists of short 5-sentence stories with annotated motivations and emotional responses whose labels are drawn from classical theories of psychology (e.g., Plutchik's Wheel; Plutchik, 1980). To evaluate our method, we map the emotion classification task to a question answering task by posing an individual question for each emotion (*disgust, surprise, fear, anger, trust, anticipation, sadness, joy*) that must be predicted for each example. As the answer text to score, we use formulations of the words that make up the classification label (e.g., *disgusted, surprised, afraid, angry, trusting, excited, sad, happy*). The total number of QA pairs extracted is outlined in Table 4.1. As question representations q to give to COMET, we use the relations from ATOMIC (Sap et al., 2019a), the knowledge graph on which COMET is trained, that correspond to reactions to events: `xReact` and `oReact`. When computing ϕ_{ga} (Eq. 4.4, 4.6), we compute a score for $q =$ `xReact`, `oReact` and average them. Rules for pruning inferences are presented in Appendix B.

4.5.2 Experimental Settings

Hyperparameters We use most of the same hyperparameters to train the COMET model on the ATOMIC knowledge graph as in Chapter 3. However, instead of using GPT (Radford et al., 2018) as the pretrained language model that seeds COMET, we use GPT2-345M (Radford et al., 2019). We also freeze position embeddings so we can generalize to longer contexts than in ATOMIC. The number of levels in the graph L is set to 2, meaning we only do one hop of commonsense inferences. As we operate in the zero-shot setting, we do not tune inference hyperparameters. For the SOCIALIQA dataset, we set $\gamma_g = \gamma_{ga} = 1.0$ and $\beta^\ell = 1.0\ \forall \ell$. For STORYCOMMONSENSE, we use the same hyperparameters except that $\gamma_g = 0$. Unless stated otherwise, we use argmax greedy decoding to generate commonsense inferences from COMET.

Approximating scores We approximate the marginal distribution for the PMI calculation in Equation (4.6) using Equation (4.2), but set $c =$ "`PersonX`". Every training example in the ATOMIC knowledge graph on which COMET is trained begins with this token, so using it as the only token in the context essentially provides an output distribution that is only conditioned on the question q.

Question	Template
What will happen to Others?	The effect on others will be ___
How would Others feel as a result?	Others feel ___
What will Others want to do next?	After, others will want to ___
How would you describe CHARACTER?	CHARACTER is ___
What will happen to CHARACTER?	The effect on CHARACTER will be ___
What does CHARACTER need to do before this?	Before, CHARACTER needs to ___
Why did CHARACTER do this?	CHARACTER did this because ___
How would CHARACTER feel afterwards?	CHARACTER feels ___
What will CHARACTER want to do next?	After, CHARACTER will want to ___

Table 4.2: Templates used to convert question answering pairs from SOCIALIQA to a format that can be evaluated by the baseline pretrained language models: GPT, GPT2-117M, GPT2-345M, and GPT2-762M.

Predictions To predict an answer on the SOCIALIQA dataset, we use Equation (4.10). Making predictions for STORYCOMMONSENSE is more complicated as the task is originally a binary classification task for eight different dimensions (i.e., emotional responses). To make a prediction, we treat ϕ_a (Eq. 4.9) for each dimension independently and select an answer based on whether ϕ_a is above a dimension-specific threshold:

$$\hat{a} = \mathbb{1}[\phi_a^n > \kappa^n] \tag{4.11}$$

where ϕ_a^n is the score from the model for dimension n and κ^n is the threshold for that dimension. Normally, we could tune these decision thresholds on the development set, but this would violate the zero-shot setting. Instead, decision thresholds are chosen by producing a cumulative distribution function over model scores for the development set examples. Then, we select the threshold as the score at the percentile of the positive label distribution (i.e., if the *joy* emotion is present for 20% of examples, we set the score at the 20th percentile of the CDF as the threshold).

4.6 SOCIALIQA Study

Baselines As baselines in the SOCIALIQA study, we use state-of-the-art pretrained language models: GPT (Radford et al., 2018), and the GPT2-117M, GPT2-345M, and GPT2-762M models from Radford et al. (2019). To adapt these language models optimally to the QA task, question-answer pairs are automatically converted to a templated form. For example, a question such as "How does Alice feel after?" will be

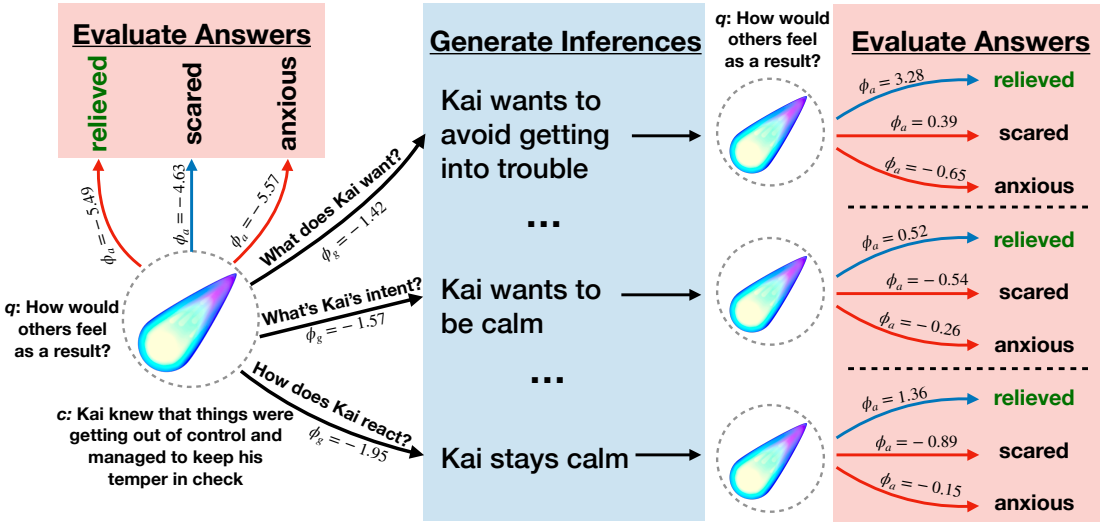


Figure 4.3: COMET receives the context c and question q. COMET can evaluate answer candidates directly or generate new commonsense inferences related to the context. By generating new inferences, it constructs a graph that can be reasoned over to select the best answer to the question. ϕ_g and ϕ_a correspond to the scores computed in Eqs. 4.1 and 4.6

replaced by the template "Alice feels." Answers are appended to the end of the template, which is then concatenated to the context, and the language models score the answer words conditioned on the context and template. Table 4.2 provides the template for each question variety. We also report the results of a model that solely uses ϕ_a^0 to select answers (i.e., answers are evaluated directly against the context with no dynamic graph construction) and call this model COMET - CA. Finally, for comparison, we report the result of supervised BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019) models, and the random baselines and human scores from Sap et al. (2019b).

Overall performance We report the main results of our SOCIALIQA study in Table 4.3. Our model achieves an absolute improvement of 10.1% over the top performing language model baseline, showing the effectiveness of using a neural representation of a commonsense knowledge base. Additionally, our approach of dynamically constructing a knowledge graph *on demand* (COMET - CGA) performs better than using the knowledge model to directly evaluate answers (COMET - CA) by 3.6%. Figure 4.3 shows an example demonstrating how the knowledge graph can help re-score answer options. The model initially selects *scared* as the most likely option when evaluating answer candidates directly, but changes its selection to *relieved* after reasoning over generated knowledge.

Model	Dev Acc.	Test Acc.
Random	33.3	33.3
GPT	41.8	41.7
GPT2 - 117M	40.7	41.5
GPT2 - 345M	41.5	42.5
GPT2 - 762M	42.5	42.4
COMET - CA	48.7	49.0
COMET - CGA	**50.1**	**52.6**
BERT-large (sup.)	66.0	63.5
RoBERTa-large (sup.)	78.1	77.0
Human	86.9	84.4

Table 4.3: Accuracy on the development and test sets of SOCIALIQA. COMET - CGA is our model.

We note, however, that the state-of-the-art performance of the supervised BERT and RoBERTa models are 63.5% and 77.0%, respectively, indicating there is still much room for improvement in developing comparable zero-shot approaches to QA. However, one point of interest is that the performance of training the BERT model with only 5000 training examples (rather than the full 30k) is close (54%) to COMET - CGA, indicating that knowledge models and joint neuro-symbolic solutions are already promising in low-data regimes.

Qualitative Analysis In Table 4.4, we present top reasoning paths for different examples from SOCIALIQA. The strength of our approach can be seen in the first example, where the correct answer, *drained*, is much more likely to be a feeling associated with wanting "to go home," a post-condition generated by COMET, rather than the original context, where it is implicit. This benefit to generating reasoning paths is also seen in the second example, where Quinn's *foolishness* is linked to "[getting] hurt." We note that COMET-CA, RoBERTa-large, and GPT2-345M all answer this question incorrectly, showing the benefit of explicit reasoning paths.

In the final two examples, we present uninteresting or failure cases. In the first, the model predicts that Alex will experience *joy* after COMET generates the inference that he will be "happy." While correct, this reasoning path is merely a synonym of *joy*, and so is not particularly interesting. In the final example, we show a case where the model selects an incorrect answer by reasoning through the wrong path. By recognizing that "Taylor wants to celebrate" as a likely post-condition of the context, the model selects an answer that is incorrect. An interesting secondary failure mode in this example is in the second generated path "Taylor wants to be home." While its most likely answer to the question is the correct one, it does not produce a path that is interpretable. In general, we find these cases to be more common in multi-sentence situations. The compositionality of the context makes it more challenging to generate directed inferences, and the factor nodes become less reliable in the graph. We observe that performance on multi-sentence contexts drops by $\sim$5%, highlighting the importance of expressive, well-calibrated neural knowledge bases.

Situation	Top Reasoning Paths	Answers
Jesse drove Ash to the airport and dropped them off at the airport with ease. How would Jesse feel afterwards?	**Jesse wants to go home**	**a) *drained* ✓** b) went to the ticket counter c) dropped me off at the airport
	Jesse wanted to be helpful	a) drained b) *went to the ticket counter* ✗ c) dropped me off at the airport
After jumping off the roof of his house Quinn had trouble breathing. How would you describe Quinn?	**Quinn gets hurt**	**a) *foolish* ✓** b) patient c) light-headed
	Quinn wants to get medical help	a) foolish b) patient c) *light-headed* ✗
Alex took notice of the children who were singing at the playground. What will happen to Alex?	**Alex is happy**	a) hurt the children **b) *joy* ✓** c) tell the children to stop
	Alex wants to go home	a) hurt the children b) joy c) *tell the children to stop* ✗
Taylor was close to winning the game. Taylor ran straight for home plate. What will Taylor want to do next?	**Taylor wants to celebrate**	a) try to get over that they did win **b) *celebrate the win* ✗** c) wanted to score
	Taylor wants to be home	a) try to get over that they did win b) celebrate the win c) *wanted to score* ✓

Table 4.4: Example contexts, questions, and answers from the SOCIALIQA dataset with corresponding paths generated by the COMET-CGA about the context. We **bold** the most likely path and *italicize* the most likely answer for each path. Incorrect answers selected by a path are highlighted in red ✗ and correct answers are highlighted in blue ✓. We only present a subset of the generated paths.

Tuning effects To evaluate how our results would have varied if we had tuned hyperparameters, we vary β^ℓ $\forall \ell \in L$ by increments of 0.1 between 0 and 1 and report the results in Figure 4.4. Regardless of the values of these hyperparameters, COMET - CGA was superior to the pretrained language models in every configuration, and better than COMET - CA (depicted by red line) 80% of the time. The black line indicates the performance of the purely zero-shot approach, which is one of the better

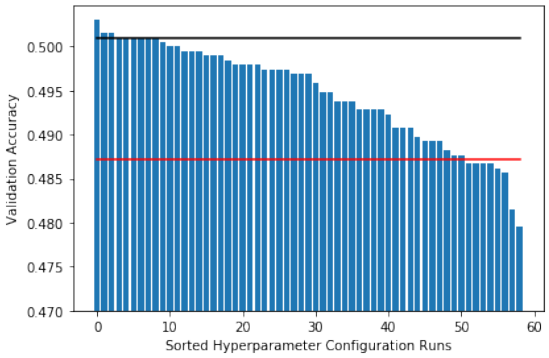


Figure 4.4: SOCIALIQA development set performance for different settings of β^ℓ.

performing configurations, though better results are possible by varying these values. The best performing configurations often have $\beta^1 \sim 1.5\beta^0$, highlighting the importance of the constructed graph.

Decoder	CGA$_{max}$	CGA
Argmax Decoding	**49.6**	50.1
Beam Search - 5	49.1	**49.5**
Beam Search - 10	49.1	50.0
Top-5 sampling	49.0	49.0
Top-10 sampling	49.4	49.3

Table 4.5: Effect of the decoding algorithm for generating commonsense inferences from COMET. All results are on the SOCIALIQA development set.

Effect of decoding strategy We also investigate whether the algorithm for generating commonsense inferences from COMET affects the utility of the generated facts. We present the effect of the following candidate generation schemes: argmax greedy decoding, beam search with beam size $b = 5, 10$ and top-k sampling (Fan et al., 2018; Holtzman et al., 2018) with k = 5, 10 . For each decoding method, we provide the inference algorithm with every candidate produced by each strategy (e.g., greedy decoding produces one candidate, top-10 sampling produces 10 candidates), yielding differently-sized commonsense knowledge graphs.

Our results in Table 4.5 show that the performance of our approach is not dependent on the decoding strategy used to generate the graph of commonsense knowledge. This result is promising as it shows that the reasoning procedure is robust to variability in the candidate generations. However, it also depicts the weaknesses of our inference algorithms for reasoning over the graph, as these approaches do not leverage larger candidate sets of commonsense knowledge to answer questions correctly. These results point to the need for future work in developing algorithms that can leverage diverse commonsense inference paths as we dynamically construct more expansive knowledge graphs.

Effect of seed language model Finally, we test the effect of the pretrained language model used to seed COMET. In addition to our main model that is trained on GPT2-345M, we train additional versions of the knowledge model from GPT, GPT2-117M, and GPT2-762M. We find that COMET is sensitive to the pretrained language model on which it is trained. In Table 4.6, we report the performance of training COMET on different seed language models.

We see that the COMET model trained on OpenAI GPT (Radford et al., 2018) has the best perfor-

Seed Language Model	CA Acc.	CGA Acc.
GPT (Radford et al., 2018)	**49.1**	49.1
GPT2 - 117M (Radford et al., 2019)	44.9	46.6
GPT2 - 345M (Radford et al., 2019)	48.7	**49.6**
GPT2 - 762M (Radford et al., 2019)	41.1	42.7

Table 4.6: Effect of pretrained language model used to seed COMET.

mance when directly answering the questions without generating an intermediate knowledge graph. However, training COMET on GPT2-345M (Radford et al., 2019) yields better results when an intermediate knowledge graph is constructed. Most importantly, however, regardless of which language model seeds COMET, generating commonsense inferences for reasoning always produces superior performance on SOCIALIQA compared to answering questions directly. Interestingly, there is a drastic performance drop when training COMET on GPT2-762M despite it being one of the better language models to evaluate directly on SOCIALIQA (as seen in Table 4.3). This result implies that the learned representations of GPT2-762M may not be as directly transferable as smaller versions of the model, though further analysis would be needed to confirm this hypothesis.

4.7 STORYCOMMONSENSE Study

Our second study is conducted on the STORYCOMMONSENSE dataset (Rashkin et al., 2018), which asks questions about the emotional states of characters in short 5-sentence stories. Eight questions are used, one for each of the original emotions in Plutchik's wheel (Plutchik, 1980): *disgust*, *surprise*, *fear*, *anger*, *trust*, *anticipation*, *sadness*, and *joy*. The answer to each question is a binary yes/no option. As mentioned in Section 4.5.2, we select the answer based on a threshold (Eq. 4.11) computed from the cumulative distribution over model scores.

Baselines As baselines, we report the performance of several models from Rashkin et al. (2018). Similar to our setup, these baselines can only access the sentence of the story for which the emotion classification must be made. These models use TF-IDF features (introduced in Chapter 2), Glove embeddings (Pennington et al., 2014), LSTMs (Hochreiter and Schmidhuber, 1997), or CNNs (Kim, 2014), respectively, to encode

the sentence. For each baseline, a multi-label linear classifier separately predicts each emotion label from this joint representation. As with SOCIALIQA, we also report the results of a baseline that only uses ϕ_a^0 to select answers (COMET - CA). Finally, we report the performance of a fine-tuned transformer model (GPT) that has access to the full story (including preceding context) to make its predictions.

Overall Performance Our results indicate that our zero-shot algorithm approaches the performance of the weaker supervised baselines for this task. This result is promising as no additional training is used to adapt the COMET model to generating contextual reasoning graphs for a classification task. Instead, we use the learned neural knowledge model to score the likelihood of tokens corresponding to emotional reactions for characters in the story. Importantly, again, we see consistent improvement from dynamically generating a contextualized commonsense knowledge graph of facts rather than directly evaluating the answer choices with COMET. Our full approach yields higher precision, F1, and accuracy than the COMET - CA baseline.

Model	P	R	F1
Random	10.4	50.0	17.2
Random (weighted)	12.3	11.8	12.0
Zero-shot	**No Training Data**		
COMET - CA	18.6	17.9	18.2
COMET - CGA	**20.6**	**19.6**	**20.1**
Tuned Hyperparameters	**No Training Data**		
COMET - CA	16.2	**60.3**	25.5
COMET - CGA	**19.0**	56.5	**28.4**
Supervised			
TF-IDF	20.1	24.1	21.9
Glove	15.2	30.6	20.3
LSTM	20.3	30.4	24.3
CNN	21.2	23.4	22.2
GPT	**41.6**	**50.2**	**45.5**

Table 4.7: Precision, Recall, F1 of emotion prediction on the STORYCOMMONSENSE dataset. The best model in the zero-shot, tuned, and supervised sections is **bolded**

To evaluate the quality of our untuned thresholds from Section 4.5.2 based on the CDF of the model's scores, we also report the results of our approach if we are allowed to tune the κ thresholds on 20% of the development data (the same amount used for validation in Rashkin et al., 2018). We see large gains in Recall from this process, causing our performance to exceed most supervised models. There is still much improvement that can be made, however, as new large-scale transformer models that are trained in a supervised manner perform considerably better on this task.

Situation	Top Reasoning Paths	Answers
Daniel was excited to get a remote control boat for his birthday. He asked his dad to drive him to the lake to try it out. *How does Daniel feel?*	His dad is helpful	disgusted, angry, sad, afraid, happy, **trusting ✓**, excited, surprised
	Daniel wants to try something new	disgusted, angry, sad, afraid, happy, trusting , **excited ✓**, surprised

Table 4.8: Example context, paths, and answers for our model on STORYCOMMONSENSE. We show which emotions are predicted through which path by **bolding** them. Correct predicted answers are highlighted in blue ✓.

Qualitative Analysis Qualitatively, we once again see the benefit of generating reasoning paths in Table 4.8. COMET-CGA is able to select the two correct answers through two different reasoning paths that it generates. These paths provide new information about the situation that was implicit in the context. By generating the commonsense inference that Daniel's "dad is helpful," it is easy to predict that Daniel is likely feeling *trusting*. None of the other emotions fit this commonsense inference. Meanwhile, by generating the inference that Daniel "wants to try something new," the answer *excited* is much easier to predict. These observations point to the idea that this framework could be especially promising in multi-label settings, where multiple questions must be considered about the same context.

4.8 Summary

In this chapter, we showed that a neural representation of a large-scale commonsense knowledge base (COMET) could be used to generate contextual commonsense knowledge graphs *on demand* for question answering. Our approach dynamically constructs a knowledge graph of commonsense inferences related to a presented context and conditions on it to evaluate answer options for a posed question. A novel inference algorithm reasons over the constructed weighted graph to select the most likely answer to a question. Our approach shows promising results at answering questions without end-task training on two datasets: SOCIALIQA and STORYCOMMONSENSE. Furthermore, our analysis indicates that dynamically generating a contextualized commonsense knowledge graph performs better than using COMET to directly answer questions, indicating the benefit of explicitly generating reasoning paths for commonsense understanding.

Our work is the first study on using powerful neural representations to generate symbolic reasoning structures that represent the underlying commonsense dynamics of situations. Importantly, these structures

could be used to reason to answer questions without using supervised labels, showing promise in low-data regimes. However, there are still clear areas of improvement, such as in constructing more expressive neural representations of knowledge bases, or exploring new graph construction and inference algorithms for reasoning about situations. A final future research direction, however, lies in whether we can encode the dynamics of reasoning implicitly in a neural model, as well, while avoiding the brittleness and interpretability issues that are typical of neural approaches.

Chapter 5

Simulating Action Dynamics with Neural Process Networks

Understanding narratives requires anticipating the causal effects of actions, even when they are not explicitly stated. In the previous chapter, we proposed a method that reasoned about these effects though graph construction. Our approach dynamically generated knowledge graphs and performed inference over paths that represented causal commonsense implications of the narrative presented. In this chapter, we introduce Neural Process Networks to understand narratives through neural simulation of action dynamics. Our work shows that it is possible to ground the internal mechanisms of neural networks to commonsense knowledge.

The resulting model is the first neural memory architecture that dynamically tracks entities by explicitly modeling semantic actions as state transformation functions. The model updates the states of the entities by executing learned action operators. Empirical results demonstrate that our proposed model can reason about the unstated causal effects of actions when formulating entity representations. This capability allows the model to provide more accurate contextual information for understanding and generating procedural text across two domains, recipes and short stories, all while offering interpretable internal representations.

The material in this chapter is adapted from Bosselut et al. (2018a) and Rashkin et al. (2018)

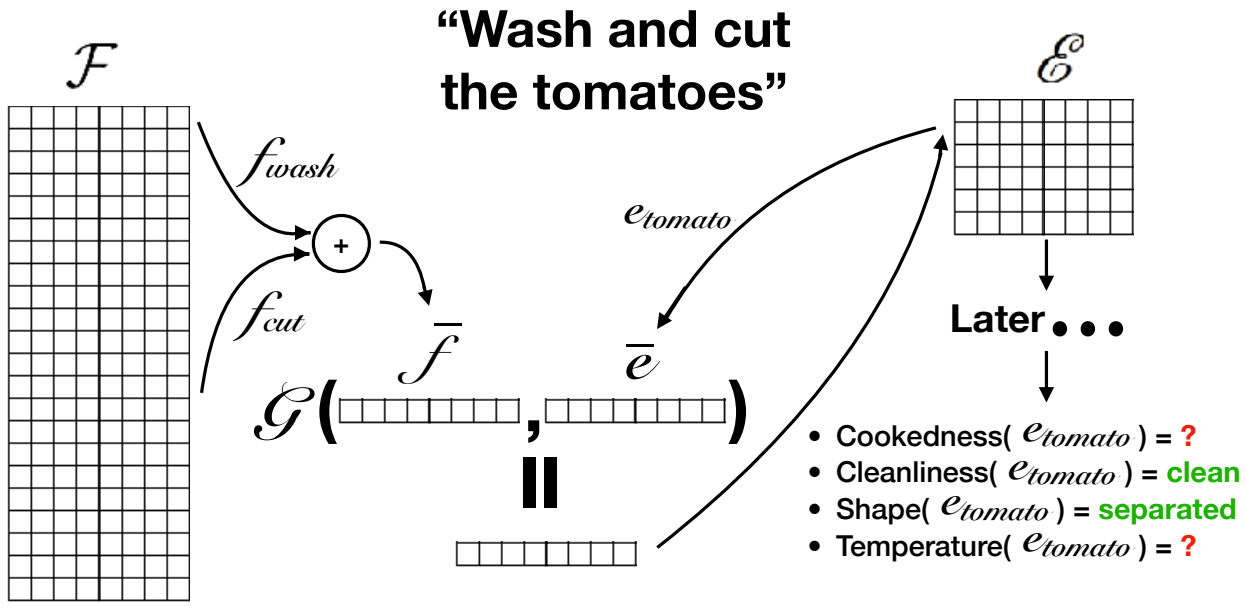


Figure 5.1: The *process* is a narrative of entity state changes induced by actions. In each sentence, entities are changed by the events described in the text. Our approach explicitly represents these actions that induce entity state changes, and used a dynamic memory to remember how entities have changed over the course of multiple steps.

5.1 Introduction

Understanding narratives, such as instructions or stories, requires anticipating the implicit causal effects of actions on entities. For example, given instructions such as “*add* blueberries to the muffin mix, then *bake* for one half hour,” an intelligent agent must be able to anticipate a number of entailed facts (e.g., the blueberries are now in the oven; their “temperature” will increase). While this common sense reasoning is trivial for humans, most natural language understanding algorithms do not have the capacity to reason about causal effects not mentioned directly in the surface strings (Levy et al., 2015; Jia and Liang, 2017; Lucy and Gauthier, 2017).

In this chapter, we introduce Neural Process Networks, a narrative understanding system that tracks common sense attributes through neural simulation of action dynamics. Our network models interpretation of natural language instructions as a *process* of actions and their cumulative effects on entities. More concretely, reading one sentence at a time, our model attentively selects what actions to execute on which entities, and remembers the state changes induced with a recurrent memory structure. In Figure 5.1, for ex-

ample, our model indexes the “tomato" embedding, selects the “wash” and “cut” functions and performs a computation that changes the “tomato” embedding so that it can reason about attributes such as its “SHAPE” and “CLEANLINESS”.

Our model contributes to a recent line of research that aims to model aspects of world state changes, such as language models and machine readers with explicit entity representations (Henaff et al., 2016; Yang et al., 2016; Ji et al., 2017), as well as other more general purpose memory network variants (Weston et al., 2014; Sukhbaatar et al., 2015; Hill et al., 2015; Seo et al., 2016). This *world-centric* modeling of language (i.e., understanding by simulation) abstracts away from the surface strings, complementing *text-centric* modeling of language, which focuses on syntactic and semantic labeling of surface words (i.e., understanding by labeling).

Unlike previous approaches, however, our model also learns explicit action representations as functional operators (See Figure 5.1). While representations of action semantics could be acquired through an embodied agent that can see and interact with the world (Oh et al., 2015), we propose to learn these representations from text. In particular, we require the model to be able to *explain* the causal effects of actions by predicting natural language attributes about entities such as “LOCATION” and “TEMPERATURE”. The model adjusts its representations of actions based on errors it makes in predicting the resultant state changes to attributes. This textual simulation allows us to model aspects of action causality that are not readily available in existing simulation environments. Indeed, most virtual environments offer limited aspects of the world – with a primary focus on spatial relations (Oh et al., 2015; Chiappa et al., 2017; Wahlstrom et al., 2015). They leave out various other dimensions of the world states that are implied by diverse everyday actions such as “dissolve” (change of “COMPOSITION”) and “wash” (change of “CLEANLINESS”).

Empirical results demonstrate that parametrizing explicit action embeddings provides an inductive bias that allows the neural process network to learn more informative context representations for understanding and generating narratives. In addition, our model offers more interpretable internal representations and can reason about the unstated causal effects of actions explained through natural language descriptors. Finally, an additional empirical setting on predicting emotional and motivational states of characters in short stories demonstrates the adaptability of our method to other domains.

5.2 Related Work

Recent studies in machine comprehension have used a neural memory component to store a running representation of processed text (Weston et al., 2014; Sukhbaatar et al., 2015; Hill et al., 2015; Seo et al., 2016). While these approaches map text to memory vectors using standard neural encoder approaches, our model, in contrast, directly interprets text in terms of the effects actions induce in entities, providing an inductive bias for learning how to represent stored memories. More recent work in machine comprehension (Henaff et al., 2016) and language modeling (Ji et al., 2017) also sought to couple memory representation with tracking entity states. Our work seeks to provide a relatively more structured representation of domain-specific action knowledge to provide an inductive bias to the reasoning process.

Neural Programmers (Neelakantan et al., 2015, 2016) have also used functions to simulate reasoning, by building a model to select rows in a database and applying operations on those selected rows. While their work explicitly defined the effect of a number of operations for those rows, we provide a framework for learning representations for a more expansive set of actions, allowing the model to learn representations for how actions change the state space.

Works on instructional language studied the task of building discrete graph representations of recipes using probabilistic models (Mori et al., 2012, 2014; Kiddon et al., 2015). We propose a complementary new model by integrating action and entity relations into the neural network architecture and also address the additional challenge of tracking the state changes of the entities.

Additional work in tracking states with visual or multimodal context has focused on 1) building graph representations for how entities change in goal-oriented domains (Si et al., 2011; Gao et al., 2016; Liu et al., 2016a) or 2) tracking visual state changes based on decisions taken by agents in environment simulators such as videos or games (Wahlstrom et al., 2015; Oh et al., 2015; Chiappa et al., 2017). Our work, in contrast, models state changes in embedding space using only text-based signals to map real-world actions to algebraic transformations.

Finally, recent work in modeling procedural text also focuses on tracking entity states. These works (Tandon et al., 2018; Dalvi et al., 2019; Du et al., 2019; Amini et al., 2020), which generally focus on text describing scientific processes (Dalvi et al., 2018), track the location and transitions of different entities in these processes. In contrast, our work focuses on the cooking domain, and tracks a larger set of state changes

than simply location. Additionally, our neural architecture is grounded to the dynamics of the cooking process, while these works use less strict inductive biases such as rule-based classification constraints (Tandon et al., 2018), multi-task learning to encourage causal consistency (Dalvi et al., 2019; Amini et al., 2020), or semi-supervised learning (Du et al., 2019).

5.3 Neural Process Network

The neural process network is an interpreter that reads in natural language sentences X_t, one at a time, and simulates the process of actions being applied to relevant entities through learned representations of actions and entities.

5.3.1 Overview and Notation

The main component of the neural process network is the simulation module (§5.3.5), a recurrent unit whose internals simulate the effects of actions being applied to entities. A set of A actions is known a priori and an embedding is initialized for each one, $\mathcal{F} = \{\mathbf{f}_1, ...\mathbf{f}_A\}$. Similarly, a set of I entities is known and an embedding is initialized for each one: $\mathcal{E} = \{\mathbf{e}_1, ...\mathbf{e}_I\}$. Each $\mathbf{e}_i$ can be considered to encode information about state attributes of that entity, which can be extracted by a set of state predictors (§5.3.6). As the model reads text, it *applies* action embeddings to the entity vectors, thereby changing the state information encoded about the entities.

For any document d, an initial list of entities I is known and $\mathcal{E}_I = \{\mathbf{e}_i | i \in I\} \subset \mathcal{E}$ entity state embeddings are initialized. As the neural process network reads a sentence from the document, it selects a subset of both $\mathcal{F}$ (§5.3.3) and $\mathcal{E}_I$ (§5.3.4) based on the actions performed and entities affected in the sentence. The entity state embeddings are changed by the action and the new embeddings are used to predict end states for a set of state changes (§5.3.6). The prediction error for end states is backpropagated to the action embeddings, allowing the neural process network to learn action representations that model the simulation of desired causal effects on entities. This process is broken down into five modules below. Unless explicitly defined, all $\mathbf{W}$ and $\mathbf{b}$ variables are parametrized linear projections and biases. We use the notation $\{\mathbf{e}_i\}_t$ when referring to the values of the entity embeddings before processing sentence X_t.

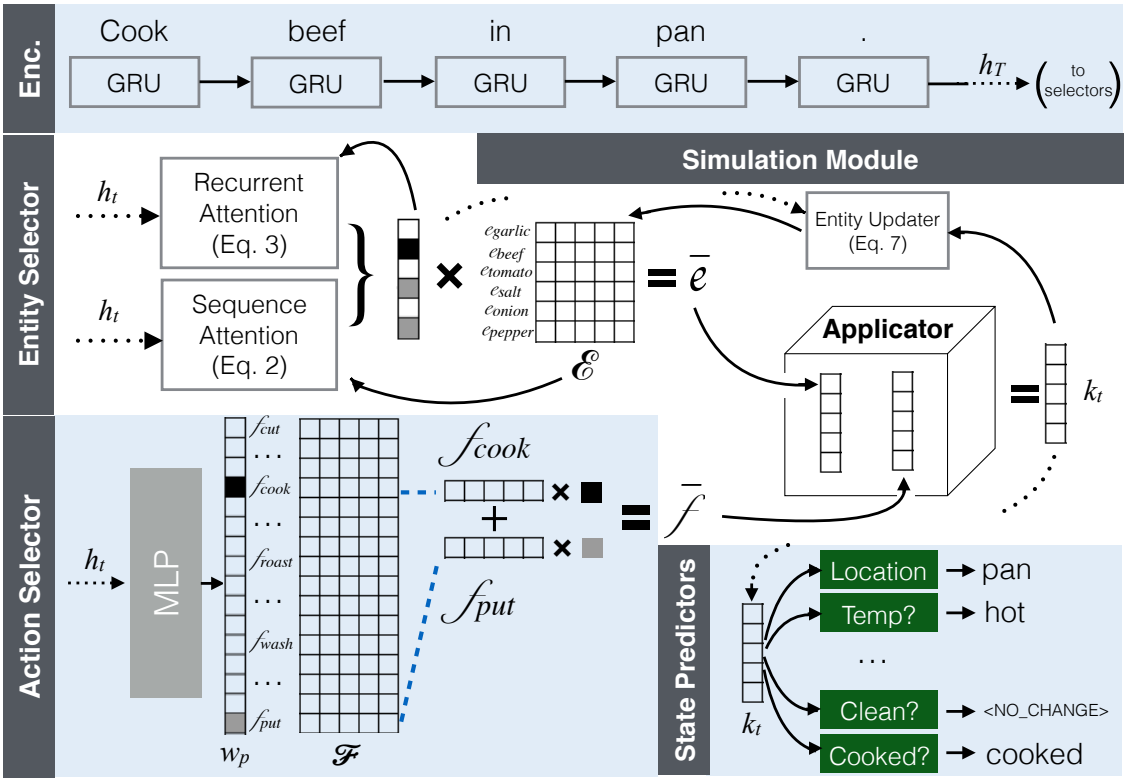


Figure 5.2: Model Summary. The sentence encoder converts a sentence to a vector representation, $\mathbf{h}_t$. The action selector and entity selector use the vector representation to choose the actions that are applied and the entities that are acted upon in the sentence. The simulation module indexes the action and entity state embeddings, and applies the transformation to the entities. The state predictors predict the new state of the entities if a state change has occurred. Equation references are provided in parentheses.

5.3.2 Sentence Encoder

Given a sentence X_t, a gated recurrent unit (Cho et al., 2014), introduced in Section 2.3.3, encodes each word and outputs its last hidden vector as a sentence encoding $\mathbf{h}_t$ (Sutskever et al., 2014).

5.3.3 Action Selector

Given $\mathbf{h}_t$ from the sentence encoder, the action selector (bottom left in Fig. 5.2) contextually determines which action(s) from $\mathcal{F}$ to execute. For example, if the input sentence is *"wash and cut beets"*, both $\mathbf{f}_{wash}$ and $\mathbf{f}_{cut}$ must be selected. To account for multiple actions, we make a soft selection over $\mathcal{F}$, yielding a

weighted sum of the selected action embeddings $\bar{\mathbf{f}}_t$:

$$\begin{aligned}\mathbf{w}_p &= \text{MLP}(\mathbf{h}_t) \\ \bar{\mathbf{w}}_p &= \frac{\mathbf{w}_p}{\sum_{j=1}^{A} \mathbf{w}_{p_j}} \\ \bar{\mathbf{f}}_t &= \bar{\mathbf{w}}_p^\intercal \mathcal{F}\end{aligned} \tag{5.1}$$

where MLP is a parametrized feed-forward network with a sigmoid activation and $\mathbf{w}_p \in \mathbb{R}^A$ is the attention distribution over A possible actions (§5.4.3). We compose the action embedding by taking the weighted average of the selected actions.

5.3.4 Entity Selector

Sentence Attention Given h_t from the sentence encoder, the entity selector chooses relevant entities using a soft attention mechanism:

$$\begin{aligned}\tilde{\mathbf{h}}_t &= \text{ReLU}(\mathbf{W}_1 \mathbf{h}_t + \mathbf{b}_1) \\ d_i &= \sigma(\mathbf{e}_{i_0}^\intercal \mathbf{W}_2 [\tilde{\mathbf{h}}_t; \mathbf{w}_p])\end{aligned} \tag{5.2}$$

where $\mathbf{W}_2$ is a bilinear mapping, $\mathbf{e}_{i_0}$ is a unique key for each entity (§5.3.5), and d_i is the attention weight for entity embedding $\mathbf{e}_i$. For example, in *"wash and cut beets and carrots"*, the model should select $\mathbf{e}_{beet}$ and $\mathbf{e}_{carrot}$.

Recurrent Attention While sentence attention would suffice if entities were always explicitly mentioned, natural language often elides arguments or uses referent pronouns. As such, the module must be able to consider entities mentioned in previous sentences. Using $\tilde{\mathbf{h}}_t$, the model computes a soft choice over whether to choose affected entities from this step's attention d_i or the previous step's attention distribution.

$$\begin{aligned}\mathbf{c} &= softmax(\mathbf{W}_3 \tilde{\mathbf{h}}_t + \mathbf{b}_3) \\ a_{i_t} &= \mathbf{c}^\intercal \begin{bmatrix} d_i \\ a_{i_{t-1}} \\ 0 \end{bmatrix}\end{aligned} \tag{5.3}$$

where $\mathbf{c} \in \mathbb{R}^3$ is the choice distribution, $a_{i_{t-1}}$ is the previous sentence's attention weight for each entity, a_{i_t} is the final attention for each entity, and 0 is an option to select no entity. Prior entity attentions can propagate forward for multiple steps.

5.3.5 Simulation Module

Entity Memory A unique state embedding $\mathbf{e}_i$ is initialized for every entity i in the document. A unique key to index each embedding $\mathbf{e}_{i_0}$ is set as the initial value of the embedding (Henaff et al., 2016; Miller et al., 2016). After the model reads X_t, it modifies $\{\mathbf{e}_i\}_t$ to reflect changes influenced by actions. At every time step, the entity memory receives the attention weights from the entity selector, normalizes them and computes a weighted average of the relevant entity state embeddings:

$$\alpha_i = \frac{a_i}{\sum_{j=1}^{I} a_j} \tag{5.4}$$

$$\bar{\mathbf{e}}_t = \sum_{i=1}^{I} \alpha_i \mathbf{e}_i \tag{5.5}$$

Applicator Given the action summary embedding $\bar{\mathbf{f}}_t$ and the entity summary embedding $\bar{\mathbf{e}}_t$, the applicator (middle right in Fig. 5.2) applies the selected actions to the selected entities, and outputs the new transformed entity embedding $\mathbf{k}_t$.

$$\mathbf{k}_t = \text{ReLU}(\bar{\mathbf{f}}_t \mathbf{W}_4 \bar{\mathbf{e}}_t + \mathbf{b}_4) \tag{5.6}$$

where $\mathbf{W}_4$ is a third order tensor projection. The vector $\mathbf{k}_t$ is the new representation of the entity $\bar{\mathbf{e}}_t$ after the applicator simulates the action being applied to it.

Entity Updater The entity updater interpolates the new transformed entity embedding $\mathbf{k}_t$ and the set of current entity embeddings $\{\mathbf{e}_i\}_t$:

$$\mathbf{e}_{i_{t+1}} = a_{i_t} \mathbf{k}_t + (1 - a_{i_t}) \mathbf{e}_{i_t} \tag{5.7}$$

yielding an updated set of entity embeddings $\{\mathbf{e}_i\}_{t+1}$. Each embedding is updated proportional to its entity's unnormalized attention a_i, allowing the model to completely overwrite the state embedding for any entity. For example, in the sentence *"mix the flour and water,"* the embeddings for $\mathbf{e}_{flour}$ and $\mathbf{e}_{water}$ must both be overwritten by $\mathbf{k}_t$ because they no longer exist outside of this new composition.

5.3.6 State Predictors

Given the new transformed entity embedding $\mathbf{k}_t$, the state predictor (bottom right in Fig. 5.2) predicts changes to the resulting entity embedding $\mathbf{k}_t$ along the following six dimensions: LOCATION, COOKEDNESS, TEMPERATURE, COMPOSITION, SHAPE, and CLEANLINESS. Discrete multi-class classifiers, one for each dimension, take in $\mathbf{k}_t$ and predict a unique end state for their corresponding state change type:

$$P(y_z|\mathbf{k}_t) = softmax(\mathbf{W}_z\mathbf{k}_t + \mathbf{b}_z) \tag{5.8}$$

For LOCATION changes, which require contextual information to predict the end state, $\mathbf{k}_t$ is concatenated with the original sentence representation $\mathbf{h}_t$ to predict the final state.

5.4 Experimental Setup

5.4.1 Dataset

For learning and evaluation, we use a subset of the Now You're Cooking dataset (Kiddon et al., 2016). We chose 65816 recipes for training, 175 recipes for development, and 700 recipes for testing. For the development and test sets, crowdsourced workers densely annotate actions, entities and state changes that occur in each sentence so that we can tune hyperparameters and evaluate on gold evaluation sets. Annotation details are provided in Appendix C.2.3.

5.4.2 Training

The neural process network is trained by jointly optimizing multiple losses for the action selector, entity selector, and state change predictors. Importantly, our training scheme uses weak supervision because dense annotations are prohibitively expensive to acquire at a very large scale. Thus, we heuristically extract verb mentions from each recipe step and assign a state change label based on the state changes induced by that action (§5.4.3). Entities are extracted similarly based on string matching between the instructions and the ingredient list. We use the following losses for training:

Action	State Change Types	End States
braise	COOKEDNESS; TEMPERATURE	COOKED; HOT
chill	TEMPERATURE	COLD
knead	SHAPE	MOLDED
wash	CLEANLINESS	CLEAN
dissolve	COMPOSITION	COMPOSED
refrigerate	TEMPERATURE; LOCATION	COLD; REFRIGERATOR
slice	SHAPE	SEPARATED

Table 5.1: Example actions, the state changes they induce, and the possible end states

Action Selection Loss Using noisy supervision, the action selector is trained to minimize the cross-entropy loss for each possible action, allowing multiple actions to be chosen at each step if multiple actions are mentioned in a sentence. The MLP in the action selector (Eq. 5.1) is pretrained.

Entity Selection Loss Similarly, to train the attentive entity selector, we minimize the binary cross-entropy loss of predicting whether each entity is affected in the sentence.

State Change Loss For each state change predictor, we minimize the negative loglikelihood of predicting the correct end state for each state change.

Coverage Loss An underlying assumption in many narratives is that all entities that are mentioned should be important to the narrative. We add a loss term that penalizes narratives whose combined attention weights for each entity does not sum to more than 1.

$$\mathcal{L}_{cover} = -\frac{1}{I}\sum_{i=1}^{I}\log\sum_{t=1}^{S} a_{i_t} \tag{5.9}$$

where a_{i_t} is the attention weight for a particular entity at sentence t and I is the number of entities in a document. $\sum_{t=1}^{S} a_{i_t}$ is upper bounded by 1. This is similar to the coverage penalty used in neural machine translation (Tu et al., 2016).

5.4.3 State Change Knowledge

We focus on physical action verbs in cooking recipes. We manually collect a set of 384 actions such as *cut, bake, boil, arrange*, and *place*, organizing their causal effects along the following predefined dimensions:

LOCATION, COOKEDNESS, TEMPERATURE, SHAPE, CLEANLINESS and COMPOSITION. The textual simulation operated by the model induces state changes along these dimensions by applying actions functions from the above set of 384. For example, *cut* entails a change in SHAPE, while *bake* entails a change in TEMPERATURE, COOKEDNESS, and even LOCATION. We annotate the state changes each action induces, as well as the end state of the action, using Amazon Mechanical Turk. The set of possible end states for a state change can range from 2 for binary state changes to more than 200. (See Appendix C.2 for details). Table 5.1 provides examples of annotations in this action lexicon.

5.4.4 Hyperparameters

Model Hyperparameters We use the following hyperparameters when initializing the neural process network. The hidden size of the instruction encoder is 100 and the embedding sizes of action functions and entities are 30. We use two instruction encoders, one for the entity selector, and one for the action selector. Word embeddings and entity embeddings are initialized with skipgram embeddings (Mikolov et al., 2013a,b) using a word2vec model trained on the training set recipes. We use a vocabulary size of 7358 for words, and 2996 for entities. We use dropout with a rate of 0.3 before any non-recurrent fully connected layers (Srivastava et al., 2014).

Training Hyperparameters We use the Adam optimizer (Kingma and Ba, 2015) with a learning rate of $\eta = 0.001$ and decay the learning rate by a factor of 0.1 if we see no improvement on the development set loss over three epochs. We stop training early if the development loss does not decrease for five epochs. The batch size is 64. Gradients with respect to the coverage loss (Eq. 5.9) are only backpropagated in steps where no entity is annotated as being selected. To account for the false negatives in the training data due to the heuristic generation of the labels, gradients with respect to the entity selection loss are zeroed when no entity label is present.

5.5 Cooking Recipe Understanding

We evaluate our model on a set of intrinsic tasks centered around tracking entities and state changes in recipes to show that the model can simulate preliminary dynamics of the recipe task. Additionally, we

provide a qualitative analysis of the internal components of our model.

5.5.1 Experimental Setup

In the tracking task, we evaluate the model's ability to identify which entities are selected and what changes have been made to them in every step. We break the tracking task into two separate evaluations, entity selection and end state prediction, and also investigate whether the model learns internal representations that approximate recipe dynamics.

Metrics In the entity selection test, we report the F1 score of choosing the correct entities in any step. A selected entity is defined as one whose attention weight a_i is greater than 50% (§5.3.4). Because entities may be harder to predict when they have been combined with other entities (e.g., the mixture may have a new name), we also report the recall for selecting combined (CR) and uncombined (UR) entities. In the end state prediction test, we report how often the model correctly predicts the state change performed in a recipe step and the resultant end state. This score is then scaled by the accuracy of predicting which entities were changed in that same step. We report the average F1 and accuracy across the six state change types.

Baselines We compare our models against two baselines. First, we compare against a GRU model that is trained to predict entities and state changes independently. This can be viewed as a bare minimum network with no action representations or recurrent entity memory. The second baseline is a Recurrent Entity Network (Henaff et al., 2016) with changes to fit our task. First, the model can tie memory cells to a subset of the full list of entities so that it only considers entities that are present in a particular recipe. Second, the entity distribution for writing to the memory cells is re-used when we query the memory cells. The normalized weighted average of the entity cells is used as the input to the state predictors. The unnormalized attention when writing to each cell is used to predict selected entities. Both baselines are trained with entity selection and state change losses (§5.4.2).

Ablations We report results on six ablations. First, we remove the recurrent attention (Eq. 5.3). The model only predicts entities using the current encoder hidden state. In the second ablation, the model is trained with no coverage penalty (Eq. 5.9). The third ablation prunes the connection from the action selector $\mathbf{w}_p$

Model	Entity Selection			State Change	
	F1	UR	CR	F1	ACC
2-layer LSTM Entity Recognizer	50.98	74.03	13.33	-	-
Adapted Gated Recurrent Unit	45.94	67.69	7.74	41.16	52.69
Adapted Recurrent Entity Network	48.57	71.88	9.87	42.32	53.47
- Recurrent Attention (Eq. 5.3)	48.91	72.32	12.67	42.14	50.48
- Coverage Loss (Eq. 5.9)	55.18	73.98	20.23	44.44	**55.20**
- Action Connections (Eq. 5.2)	54.85	73.54	20.03	44.05	54.81
- Action Selector Pretraining	54.91	73.87	20.30	44.28	55.00
+ Pretrained Action Embeddings	55.16	74.02	20.32	44.02	55.03
- Action Embedding Updates	53.79	70.77	18.60	44.27	55.02
Full Model	**55.39**	**74.88**	**20.45**	**44.65**	55.07

Table 5.2: Results for entity selection and state change selection

to the entity selector (Eq. 5.2). We also explore not pretraining the action selector. Finally, we look at two ablations where we intialize the action embeddings with vectors from a skipgram model. In the first, the model operates normally, and in the second, we do not allow gradients to backpropagate to the action embeddings, updating only the mapping tensor $\mathbf{W}_4$ instead (Eq. 5.6).

5.5.2 Results

Entity Selection As shown in Table 5.2, our full model outperforms all baselines at selecting entities, with an F1 score of 55.39%. The ablation study shows that the recurrent attention, coverage loss, action connections and action selector pretraining improve performance. Our success at predicting entities extends to both uncomposed entities, which are still in their raw forms (e.g., melt the *butter* $\rightarrow$ butter), and composed entities, in which all of the entities that make up a composition must be selected. For example, in a *Cooking lasagna* recipe, if the final step involves baking the prepared lasagna, the model must select all the entities that make up the lasagna (e.g., lasagna sheets, beef, tomato sauce). In Table 5.3, we provide examples of our model's ability to handle complex cases such as compositional entities (Ex. 1, 3), and elided arguments over long time windows (Ex. 2). We also provide examples where the model fails to select the correct entities because it does not identify the mapping between a reference construct such as "pizza" (Ex. 4) or "dough" (Ex. 5) and the set of entities that composes it, showcasing the difficulty of selecting the full set for a composed entity.

Good	X_{t-1}	Add tomato paste, broth, garlic, chili powder, cumin, chile peppers, and water.
	X_t	Bring to boil, then turn very low, cover and simmer until meat is tender.
	selected	meat, garlic, chili powder, tomato paste, cumin, chiles, beef broth, water
	correct	Same + [oil]
Good	X_{t-4}	Stir in oats, sugar, flour, corn syrup, milk, vanilla extract, and salt.
	X_{t-3}	Mix well.
	X_{t-2}	Drop by measuring teaspoonfuls onto cookie sheets.
	X_{t-1}	Bake 5 - 7 minutes.
	X_t	Let cool.
	selected	oats, sugar, flour, corn syrup, milk, vanilla extract, salt
	correct	oats, sugar, flour, corn syrup, milk, vanilla extract, salt
Good	X_{t-1}	In a large saucepan over low heat, melt marshmallows.
	X_t	Add sprinkles, cereal, and raisins, stir until well coated.
	selected	marshmallows, cereal, raisins
	correct	marshmallows, cereal, raisins, sprinkles
Bad	X_{t-3}	Ladle the barbecue sauce around the crust and spread.
	X_{t-2}	Add mozzarella, yellow cheddar, and monterey jack cheese.
	X_{t-1}	Next, add onion mixture and sliced chicken breast .
	X_t	Top pizza with jalapeno peppers.
	selected	jalapenos
	correct	crust, sauce, mozzarella, cheddar, monterey jack, white onion, chicken, jalapenos
Bad	X_{t-2}	Combine 1 cup flour, salt, and 1 tbsp sugar.
	X_{t-1}	Cut in butter until mixture is crumbly, then sprinkle with vinegar .
	X_t	Gather dough into a ball and press into bottom of 9 inch springform pan.
	selected	butter, vinegar
	correct	flour, salt, sugar, butter, vinegar

Table 5.3: Examples of the model selecting entities for sentence X_t. The previous sentences are provided as context in cases where they are relevant.

State Change Tracking In Table 5.2, we show that our full model outperforms competitive baselines such as Recurrent Entity Networks (Henaff et al., 2016) and jointly trained GRUs. While the ablation without the coverage loss shows higher accuracy, we attribute this to the fact that it predicts a smaller number of total state changes. Interestingly, initializing action embeddings with skipgram vectors and locking their values shows relatively high performance, indicating the potential gains in using powerful pretrained representations to represent actions.

Action	Nearest Neighbor Actions
cut	slice, split, snap, slash, carve, slit, chop
boil	cook, microwave, fry, steam, simmer
add	sprinkle, mix, reduce, splash, stir, dust
wash	rinse, scrub, refresh, soak, wipe, scale
mash	spread, puree, squeeze, liquefy, blend
place	ease, put, lace, arrange, leave
rinse	wash, refresh, soak, wipe, scrub, clean
warm	reheat, ignite, heat, light, crisp, preheat
steam	microwave, crisp, boil, parboil, heat
sprinkle	top, pat, add, dip, salt, season
grease	coat, rub, dribble, spray, smear, line

Table 5.4: Most similar actions based on cosine similarity of action embeddings

5.5.3 Analysis

Action Embeddings In our model, each action is assigned its own embedding $\mathbf{f} \in \mathcal{F}$, but many actions induce similar changes in the physical world (e.g., “cut" and “slice"). After training, we compute the pairwise cosine similarity d between each pair of action embeddings. In Table 5.4, we see that actions that perform similar functions are neighbors in embedding space, indicating the model has captured certain semantic properties of these actions. Learning action representations through the state changes they induce has allowed the model to cluster actions by their transformation functions.

Interestingly, we note the ways in which these learned representations differ from word embedding training approaches that only use textual context to learn representations (e.g., Mikolov et al., 2013a,b, word2vec). While word2vec embeddings do capture lexical semantics between these actions, the neural process network learns commonalities between actions that are not as extractable from context. When comparing actions such as “boil” and “bake, ” or “boil” and “pour,” for example, word embedding approaches are more likely to learn that “boil” and “pour” ($d(\texttt{boil}, \texttt{pour}) = 0.548$) are similar to one another than “boil” and “bake” ($d(\texttt{boil}, \texttt{bake}) = 0.329$). We hypothesize that the textual context surrounding “boil” and “pour” generally involves water, making their representations similar in terms of lexical context.

Conversely, the neural process network learns action embedding representations based on the state changes those actions induce as opposed to the local context windows around the mentions of the action in text. Therefore, the model encodes different semantics in the learned representation, which we find re-

flected in post-training similarity measurements ($d(\texttt{boil}, \texttt{pour}) = -0.119$) and ($d(\texttt{boil}, \texttt{bake}) = 0.687$).

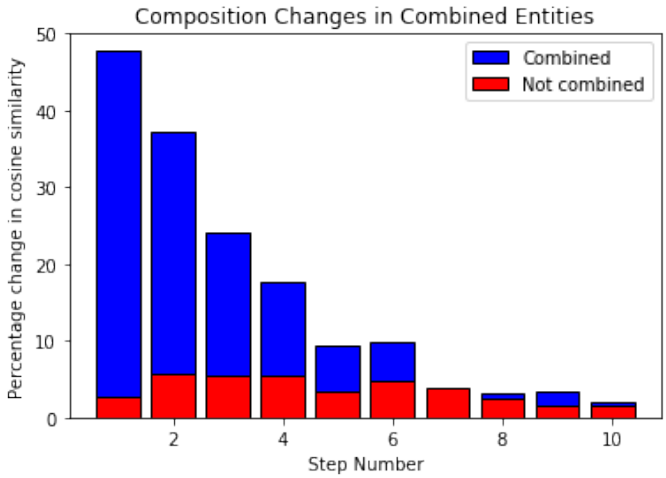


Figure 5.3: Change in cosine similarity of entity state embeddings

Entity Compositions When individual entities are combined into new constructs, our model averages their state embeddings (Eq. 5.5), applies an action embedding to them (Eq. 5.6), and writes them to memory (Eq. 5.7). The state embeddings of entities that are combined should be overwritten by the same new embedding. In Figure 5.3, we present the percentage increase in cosine similarity for state embeddings of entities that are combined in a sentence (blue) as opposed to the percentage increase for those that are not (red bars). While the soft attention mechanism for entity selection allows similarities to leak between entity embeddings, our system is generally able to model the compositionality patterns that result from entities being combined into new constructs. However, we see that this difference goes to near zero in later steps, showing the importance of methods that can maintain well-calibrated representations across multiple steps.

5.6 Cooking Recipe Generation

We evaluate the quality of the states encoded by our model on the extrinsic task of generating future steps in a recipe. The generation task tests whether our system can produce the next step in a recipe based on the previous steps that have been performed. The model is provided all of the previous steps as context.

5.6.1 Model Setup

To apply our model to the task of recipe step generation, we input the context sentences through the neural process network and record the entity state vectors once the entire context has been read (§5.3.5). These vectors can be viewed as a snapshot of the current state of the entities once the preceding context has been simulated inside the neural process network. We encode these vectors using a bidirectional GRU (Cho et al., 2014) and take the final time step hidden state as the representation of the entity states, $\mathbf{h}^e$. A different GRU encodes the context words in the same way (yielding $\mathbf{h}^c$) and the first hidden state input to the decoder is computed using the projection function:

$$\tilde{\mathbf{h}}_0 = \mathbf{W}_5(\mathbf{h}^e \odot \mathbf{h}^c) \tag{5.10}$$

where $\odot$ is the Hadamard product between the two encoder outputs. The hidden size of the context encoder, state vector encoder, and decoder are all 200. State vectors have dimensionality 30 (the same as in the neural process network). A dropout rate of 0.3 is used when training the decoder. Both encoders and the decoder are single layer. The learning rate is $\eta = 0.0003$ initially and is halved every 5 epochs. The model is trained with the Adam optimizer (Kingma and Ba, 2015) to minimize the negative loglikelihood of predicting the next word for the full sequence:

$$\mathcal{L}_{gen} = -\sum_{n=1}^{N} \log P(y_n | y_0, ..., y_{n-1}, \tilde{\mathbf{h}}_0) \tag{5.11}$$

where y_n are the tokens of the next step in the recipe.

5.6.2 Experimental Setup

Metrics We report the combined BLEU score and ROUGE score of the generated sequence relative to the reference sequence. Each candidate sequence has one reference sentence. Both metrics are computed at the corpus-level. Also reported are VF1, the F1 score for the overlap of the actions performed in the reference sequence and the verbs mentioned in the generated sequence, and SF1, the F1 score for the overlap of end states annotated in the reference sequence and predicted by the generated sequences. End states for the generated sequences are extracted using the lexicon from Section 5.4.3 based on the actions performed in

Model	BLEU	ROUGE-L	VF1	SF1
Vanilla Seq2Seq	2.81	33.00	16.17	40.21
Attentive Seq2Seq	2.83	33.18	16.97	41.43
EntNet Generator	2.30	32.71	17.53	42.43
NPN Generator	**3.74**	**35.64**	**20.12**	**43.40**

Table 5.5: Automatic metrics evaluation for recipe continuation generation

the sentence.

Baselines For the generation task, we use three baselines: a seq2seq model with no attention (Sutskever et al., 2014), an attentive seq2seq model (Bahdanau et al., 2014), and a similar variant as our NPN generator, except where the entity states have been computed by the Recurrent Entity Network (EntNet) baseline (§5.5). Implementation details for baselines can be found in Appendix C.1.

5.6.3 Results

Our results in Table 5.5 indicate that sequences generated using the neural process network entity states as additional input yield higher scores than competitive baselines. The entity states allow the model to predict next steps conditioned on a representation of the world being simulated by the neural process network. Additionally, the higher VF1 and SF1 scores indicate that the model is indeed using the extra information to better predict the actions that should follow the context provided. Example generations for each baseline from the development set are provided in Table 5.6, showing that the NPN generator can use information about ingredient states to reason about the most likely next step. The first and second examples are interesting as it shows that the NPN-aware model has learned to condition on entity state – knowing that raw butter will likely be melted or that a cooked flan must be refrigerated. The third example is also interesting because the model learns that cooked vegetables such as squash will sometimes be drained, even if it is not relevant to this recipe because the squash is steamed. The seq2seq and EntNet baselines, meanwhile, output reasonable sentences given the immediate context, but do not exhibit understanding of global patterns.

Context	Preheat oven to 425 degrees.
Reference	Melt butter in saucepan and mix in bourbon, thyme, pepper, and salt.
NPN	Melt butter in skillet.
Seq2seq	Lightly grease 4 x 8 baking pan with sunflower oil.
Attentive Seq2seq	Combine all ingredients and mix well.
EntNet	In a large bowl, combine flour, baking powder, baking soda, salt, and pepper.
Context	Pour egg mixture over caramelized sugar in cake pan. Place cake pan in large shallow baking dish. Bake for 55 minutes or until knife inserted into flan comes out clean.
Reference	Cover and chill at least 8 hours.
NPN	Refrigerate until ready to use.
Seq2seq	Serve at room temperature.
Attentive Seq2seq	Store in an airtight container.
EntNet	Store in an airtight container.
Context	Cut squash into large pieces and steam. Remove cooked squash from shells;
Reference	Measure 4 cups pulp and reserve remainder for another dish.
NPN	Drain.
Seq2seq	Mash pulp with a fork.
Attentive Seq2seq	Set aside.
EntNet	Set aside.

Table 5.6: Examples of the model generating sentences compared to baselines. The context and reference are provided first, followed by our model's generation and then the baseline generations

5.7 Short Story Understanding

To test whether the neural process network is adaptable to different domains with different world knowledge, we evaluate the performance of the model on various story understanding tasks introduced in Rashkin et al. (2018). These tasks are present in the same STORYCOMMONSENSE dataset used in Chapter 4.

5.7.1 Task

We train on the emotion classification task, as before, and on the motivation classification tasks. Motivation classification derives two distinct label sets from prominent classical theories of motivations: Maslow's hierarchy of needs (Maslow, 1943), and Reiss' 16 motives (Reiss, 2004). In each classification task, a model is given a line of the story (along with optional preceding context lines) and a character, and predicts the emotional reaction (or motivation) of the character to the event in the line of the story. A binary label is predicted for each emotion or motivation category.

5.7.2 Model Setup

Neural Process Network We use the same implementation as in Sections 5.5 and 5.6 with minor modifications. First, we replace the GRU-based encoders (§5.3.2) with the positionally-weighted bag-of-word encoders introduced by Henaff et al. (2016):

$$\mathbf{h}_t = \sum_n m_n \odot \mathbf{x}_n \tag{5.12}$$

where $\mathbf{x}_n$ is a word embedding at index n in a sentence indexed by t, m_n is a positional weight for that index in the sentence, and $\mathbf{h}_t$ is a sentence representation. Word embeddings of dimensionality 100 are initialized with pretrained Glove vectors (Pennington et al., 2014).

As action functions, we initialize a set of 100 vectors, but we do not ground the action function representations to real actions as in the Cooking domain. Instead, we treat these actions as latent. Actions are indexed in the same way as in Equation (5.1), except that we use a $softmax$ over action attentions, rather than a σ function.

Entities are selected as in Section 5.3.4, except that we do not use recurrent attention and we do not condition on the action function distribution (Eq. 5.1). Because compositions are less likely to occur in stories compared to recipes, we model the action functions being applied to each entity individually.

$$\tilde{\mathbf{e}}_i = \phi(\bar{\mathbf{f}}_t \mathbf{W}_3 \mathbf{e}_i + \mathbf{W}_4 \mathbf{h}_t) \tag{5.13}$$

where $\mathbf{e}_i$ is the value stored in memory cell i, $\bar{\mathbf{f}}_t$ is the representation of the selected actions, $\mathbf{W}_k$ are projection matrices, and ϕ is a PReLU non-linearity. Entity states are updated in the following way:

$$\mathbf{e}_i \leftarrow (1 - d_i)\mathbf{e}_i + d_i \odot \tilde{\mathbf{e}}_i \tag{5.14}$$

$$\mathbf{e}_i \leftarrow \frac{\mathbf{e}_i}{||\mathbf{e}_i||} \tag{5.15}$$

where d_i are the entity selection probabilities from Section 5.3.4. We initialize entity memory keys and entity memory values with the sum of the Glove vectors (Pennington et al., 2014) for all the words in the

character name. We use dropout of 0.5 between the encoder and dynamic memory.

Classifier For each character, the state classifiers predict a binary label $\hat{y}_z$ for every category $z \in \mathcal{Z}$ where $\mathcal{Z}$ is the set of category labels for each task (e.g., *disgust*, *surprise*, etc. in the emotional response categorization task). We use logistic regression as a classifier:

$$\hat{y}_z = \sigma(\mathbf{w}_z^\intercal \mathbf{e}_i + b_z) \tag{5.16}$$

where $\mathbf{w}_z$ and b_z are a label-specific set of weights and bias for classifying each label $z \in \mathcal{Z}$, and $\mathbf{e}_i$ is the memory cell value for a character in the story whose motivations or emotions we are predicting.

Training Hyperparameters We train the model with the Adam optimizer (Kingma and Ba, 2015) and a learning rate $\eta = 0.001$ and gradient clipping if the norm of the gradients exceeds 1.

5.7.3 Experimental Setup

Data As the STORYCOMMONSENSE dataset has no training set with classification labels (Rashkin et al., 2018), we split the development set D into two portions of 80% (D_1) and 20% (D_2). D_1 is used to train the neural process network and classifier. D_2 is used to tune hyperparameters. The model is trained to minimize the weighted binary cross entropy of predicting a class label y_z for each class z:

$$\mathcal{L}_{class} = \sum_{z=1}^{Z} \gamma_z y_z \log \hat{y}_z + (1 - \gamma_z)(1 - y_z) \log(1 - \hat{y}_z) \tag{5.17}$$

where Z is the number of labels in each of the three classifications tasks and γ_z is defined as:

$$\gamma_z = 1 - e^{-\sqrt{P(y_z)}} \tag{5.18}$$

where $P(y_z)$ is the marginal class probability of a positive label for z in the training set.

Pretraining The training set of the STORYCOMMONSENSE dataset contains free-text annotations of emotions and motivations for each character in each line of each story. We explore pretraining the neural process network on this larger dataset of information. To learn to generate these free-text explanations, we train a

single-layer LSTM (Hochreiter and Schmidhuber, 1997) decoder that receives an entity state vector e_i as its seed hidden and cell state. The decoder is trained to minimize the negative loglikelihood of predicting each word in the explanation of a character's state:

$$\mathcal{L}_{gen} = -\sum_{j=1}^{N} \log P(y_n | y_0, ..., y_{n-1}, \mathbf{e}_i) \tag{5.19}$$

where y_n are the tokens of the free-text annotation. When outputting words, we concatenate the original hidden state from the encoder, $\mathbf{e}_i$, to the output of the LSTM decoder before predicting a word. As before, word embeddings in the decoder are initialized with Glove vectors (Pennington et al., 2014). We train the decoder with $\eta = 0.0003$ as the learning rate.

Metrics For the state classification task, we report the micro-averaged precision (P), recall (R), and F1 score of the Plutchik, Maslow, and Reiss prediction tasks.

Baselines We report two sets of random baselines, one where each label is predicted with a coin flip, and one where we re-weight the probability of predicting the label by its class probability. We note that the class-weighted random baseline performs worse because the distribution of positive instances for each category is very uneven: macro-averaged positive class probabilities of 8.2%, 1.7%, and 9.9% per category for Maslow, Reiss, and Plutchik, respectively. We also report the performance of replacing the neural process network with a different encoder for the story: TF-IDF vectors, Glove vectors (Pennington et al., 2014), a convolutional neural network (CNN; Kim 2014), a LSTM (Hochreiter and Schmidhuber, 1997), or a recurrent entity network (EntNet; Henaff et al. 2016). Implementation details for these baselines can be found in Appendix C.1.3.

5.7.4 Results

Our results in Table 5.7 are mixed. While the neural process network shows strong performance in the emotion response prediction task, it performs much more poorly at generating motivations. We hypothesize that this is due to the inductive bias of the model, which is designed to predict the states of entities after actions are applied to them, thereby simulating the causal effects of those actions. Emotional responses are

Model	Motivation Classification						Emotion Classification		
	Maslow			Reiss			Plutchik		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Random	7.45	49.99	12.96	1.76	50.02	3.40	10.35	50.00	17.15
Random (Weighted)	8.10	8.89	8.48	2.25	2.40	2.32	12.28	11.79	12.03
TF-IDF	30.10	21.21	24.88	18.40	20.67	19.46	20.05	24.11	21.90
+ Entity Context	29.79	34.56	32.00	20.55	24.81	22.48	22.71	25.24	23.91
GloVe	25.15	29.70	27.24	16.65	18.83	17.67	15.19	30.56	20.29
+ Entity Context	27.02	37.00	31.23	16.99	26.08	20.58	19.47	**46.65**	27.48
LSTM	24.64	35.30	29.02	19.91	19.76	19.84	20.27	30.37	24.31
+ Entity Context	**31.29**	33.85	32.52	18.35	27.61	22.05	23.98	31.41	27.20
+ Explanation Pretraining	30.34	40.12	34.55	**21.38**	28.70	**24.51**	25.31	33.44	28.81
CNN (Kim, 2014)	26.21	31.09	28.44	20.30	23.24	21.67	21.15	23.36	22.20
+ Entity Context	27.47	41.01	32.09	18.89	31.22	23.54	24.32	30.76	27.16
+ Explanation Pretraining	29.30	44.18	**35.23**	17.87	**37.52**	24.21	24.47	38.87	30.04
EntNet (Henaff et al., 2016)	26.24	42.14	32.34	16.79	22.20	19.12	**26.22**	33.26	29.32
+ Explanation Pretraining	26.85	**44.78**	33.57	16.73	26.55	20.53	25.30	37.30	30.15
NPN	24.27	44.16	31.33	13.13	26.44	17.55	21.98	37.31	27.66
+ Explanation Pretraining	26.60	39.17	31.69	15.75	20.34	17.75	24.33	40.10	**30.29**

Table 5.7: State classification results on the STORYCOMMONSENSE dataset.

post-conditions of actions while motivations are pre-conditions, and are therefore less likely to be predictable from entity states sampled after an event changes the entity memory. We note that the recurrent entity network (EntNet), which has a similar memory for tracking entities also struggles at predicting motivations of characters.

For emotion state prediction, however, both the neural process network and recurrent entity network show superior performance over classical neural baselines, such as LSTMs or CNNs, highlighting the importance of explicit entity modeling. When pretrained on open-ended annotations, the neural process network exceed the performance of EntNet, highlighting the applicability of action-grounded transformations.

5.8 Summary

In this chapter, we explored how commonsense knowledge about the causal effects of actions could be represented by the parameters of a neural network. We introduced the Neural Process Network, which models a process of actions and their causal effects on entities by learning action transformations that change entity state representations. The model maintains a recurrent memory structure to track entity states and is trained to predict the state changes that entities undergo as a result of the events described in text. These

predictions provide an interface into the dynamics being simulated by the neural process network, yielding both an expressive neural model for representing procedural text dynamics, as well as an interpretable view into the internals of the network.

Our empirical results demonstrate that the model can learn the causal effects of action semantics in the cooking domain and track the dynamic state changes of entities, showing advantages over competitive baselines. Additionally, we show that representations of entity state formulated through simulation of causal effects of actions are better to condition on to seed downstream task models, such as recipe generators. Finally, a third study on predicting the emotional state of characters in short stories demonstrates that the neural process network can be adapted to other tasks that require predicting the post-conditions of latent events described by text. While these results are promising, there are promising future work opportunities in grounding neural networks to more general commonsense knowledge and reasoning principles, possibly using representational tools such as COMET, introduced in Chapter 3.

Chapter 6

Discourse-Aware Neural Rewards for Coherent Text Generation

Text generation models frequently ignore commonsense constraints of the world, often producing narratives that describe practically improbable (or impossible) real-world situations. In a vicious cycle, these improbable narratives lead to degenerate text, which exacerbates narrative coherence issues. In this chapter, we investigate the use of discourse-aware rewards with reinforcement learning to guide a model to generate long, coherent text that is grounded to world knowledge. In particular, we propose to learn neural rewards that model cross-sentence ordering as a means to approximate desired temporal semantics.

Empirical results demonstrate that a generator trained with the learned reward produces more coherent and less repetitive text than models trained with cross-entropy or with reinforcement learning with commonly used scores as rewards. This effect is compounded as the generated text increases in length, with human evaluators assessing a clear difference in quality among longer sequences when comparing our method to baselines. However, our approach is not without downsides, as using reinforcement learning with neural rewards requires careful tuning of hyperparameters to learn desired behavior.

The material in this chapter is adapted from Bosselut et al. (2018b)

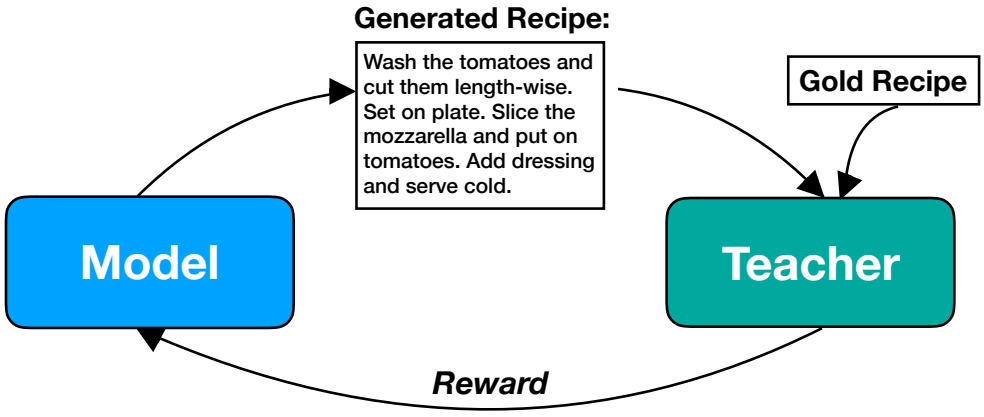


Figure 6.1: The generator is rewarded for imitating the discourse structure of the gold sequence.

6.1 Introduction

Defining an ideal loss function for training text generation models remains an open research question. Many existing approaches based on variants of recurrent neural networks (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) are trained using cross-entropy loss (Bahdanau et al., 2014; Vinyals et al., 2015; Xu et al., 2015; Rush et al., 2015), often augmented with additional terms for topic coverage or task-specific supervision (Kiddon et al., 2016; Yang et al., 2016).

Training with cross-entropy, however, does not always correlate well with achieving high scores on commonly used evaluation measures such as ROUGE (Lin, 2004), BLEU (Papineni et al., 2002), or CIDEr (Vedantam et al., 2015). Another current line of research therefore explores training generation models that directly optimize the target evaluation measure (Wu et al., 2016; Ranzato et al., 2015; Paulus et al., 2018; Rennie et al., 2017) using reinforcement learning methods such as the REINFORCE algorithm (Williams, 1992).

Importantly, most automatic measures are based on local n-gram patterns, providing only a limited and myopic perspective of overall text quality. As a result, while models trained to directly optimize these measures can yield improvements on the same measures, they may not lead to better quality in terms of overall coherence or discourse structure. Indeed, recent studies have reported cases where commonly used measures do not align well with desired aspects of generation quality (Rennie et al., 2017; Li et al., 2016a).

The challenge, however, is to define a global score that can measure the complex aspects of text quality

beyond local n-gram patterns. In this chapter, we investigate *learning* neural rewards and their use in a reinforcement learning regime with a specific focus on learning more discourse-aware and coherent text generation. Our approach shares the spirit of the work of Lowe et al. (2017), where neural scores were learned to approximate human judgments of dialogue quality. The key difference is that our rewards can be fully automatically constructed without requiring human judgments and can be trained in an unsupervised manner.

More specifically, we propose a neural reward learning scheme that is trained to capture cross-sentence ordering structure as a means to approximate the desired discourse structure in documents. While we acknowledge that discourse modeling is a broad challenge that is not addressed solely by modeling sentence ordering, we use sentence ordering as a proxy for these characteristics. In our framework, a learned *teacher* function computes rewards for the underlying text generator (see Figure 6.1), which is trained using self-critical reinforcement learning (Rennie et al., 2017). We also present a new method for distributing sentence-level rewards for more accurate credit assignment.

We test our approach on the task of generating cooking recipes, and evaluate using automatic overlap metrics that measure discourse structure. We also provide human judgments that yield comprehensive insights into the model behavior induced by the learned neural rewards. Empirical results demonstrate that a generator trained with the discourse-aware rewards produces text that is more coherent and less repetitive than models trained with cross-entropy or reinforcement learning with other commonly used scores.

6.2 Related Work

The field of neural text generation has received considerable attention in tasks such as image captioning (Vinyals et al., 2015; Xu et al., 2015), summarization (Rush et al., 2015; See et al., 2017; Hoang et al., 2019), machine translation (Bahdanau et al., 2014), and recipe generation (Kiddon et al., 2016). While these works have focused on developing new neural architectures that introduce structural biases for easier *learning*, our work uses a simple architecture and focuses on improving the optimization of the learner (i.e., better *teaching*).

The importance of better teaching for RNN generators was outlined in Bengio et al. (2015), which showed that *exposure* bias from a misaligned train and test setup limited the capabilities of sequence-to-

sequence models. This limitation had been addressed in previous work by augmenting training data with examples generated by pretrained models to make models robust to their own errors (Daumé III et al., 2009; Ross et al., 2011).

More recent work on training RNNs for generation has used sequence scores such as ROUGE (Paulus et al., 2018; Celikyilmaz et al., 2018), CIDEr (Rennie et al., 2017; Pasunuru and Bansal, 2017), BLEU (Ranzato et al., 2015) and mixtures of them (Liu et al., 2017) as a global reward to train a policy with the REINFORCE algorithm (Williams, 1992). In contrast, our work uses a neural teacher to reward a model for capturing discourse semantics.

Most similar to our work is work on using neural and embedding rewards to improve dialogue (Li et al., 2016a), image captioning (Ren et al., 2017), simplification (Zhang and Lapata, 2017), and paraphrase generation (Li et al., 2017). While these works use single-sentence similarity rewards for short generation tasks, our work designs teachers to reward long-range ordering patterns.

Finally, our teachers can be seen as rewarding generators that approximate script patterns in recipes. Previous work in learning script knowledge (Schank and Abelson, 1975) has focused on extracting scripts from long texts (Chambers and Jurafsky, 2009; Pichotta and Mooney, 2016), with some of that work focusing on recipes (Kiddon et al., 2015; Mori et al., 2014, 2012). Our teachers implicitly learn this script knowledge and reward recipe generators for exhibiting it.

6.3 Neural Teachers

Recent work in image captioning (Rennie et al., 2017), machine translation (Wu et al., 2016), and summarization (Paulus et al., 2018) has investigated using policy gradient methods to fine-tune neural generation models using automatic measures such as CIDEr as the reward. However, because most existing automatic measures focus on local n-gram patterns, fine-tuning on those measures may yield deteriorated text despite increased automatic scores, especially for tasks that require long coherent generation (§6.7.1).

Since writing out a scoring term that quantifies the quality of discourse coherence is an open research question, we take inspiration from previous research that *learns* the overall ordering structure of a document as an approximation of the discourse structure (Barzilay and Lapata, 2005, 2008; Barzilay and Lee, 2004; Li and Hovy, 2014), and propose two neural teachers that can learn to score an ordered sequence of sentences.

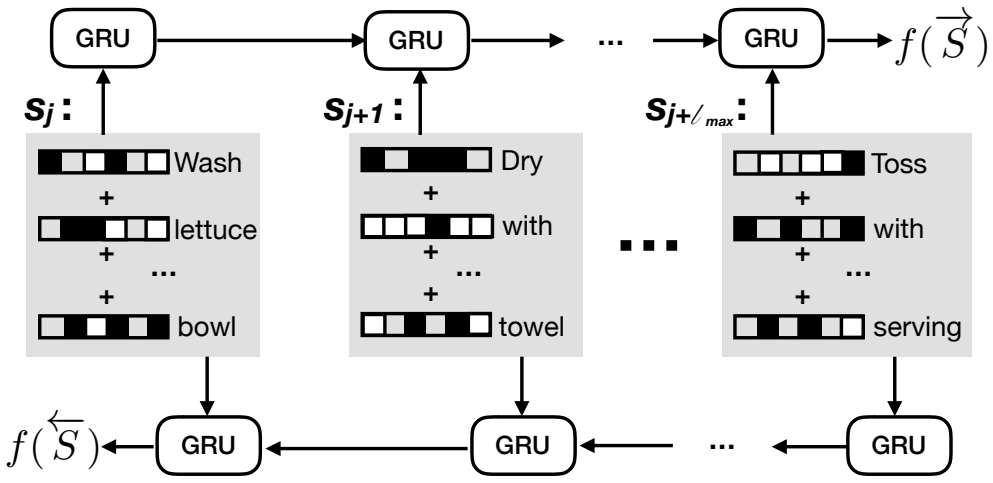


Figure 6.2: The teacher encodes the sentences of the document in the forward and reverse order.

The scores from these neural teachers are then used to formulate rewards (§6.5.2) that guide coherent long text generation systems in a policy gradient reinforcement learning setup. Notably, the neural teachers are trained offline on gold sequences in an unsupervised manner prior to training the generator. They are not trained jointly with the generator and their parameters are fixed during policy learning.

6.3.1 Notation

We define a document of N sentences as $S = \{s_0, ..., s_N\}$ where each sentence S_j has L_j words.

6.3.2 Absolute Order Teacher

The first teacher explored is motivated by work on deep semantic similarity models (Huang et al., 2013), which approximated the similarity between queries and documents in information retrieval tasks. We extend this approach to modeling temporal patterns by training a sentence encoder to minimize the similarity between a sequence encoded in its forward order, and the same sequence encoded in the reverse order (see Figure 6.2).

To focus the teacher on discourse structure, we design the encoder to capture *sentence order*, instead of

word order. Words in each sentence s_j are encoded using a bag of words:

$$\mathbf{s}_j = \sum_{i=1}^{L_j} \mathbf{x}_{ij} \tag{6.1}$$

where $\mathbf{x}_{ij}$ is a word embedding and $\mathbf{s}_j$ is a sentence embedding. Each $\mathbf{s}_j$ is passed to a gated recurrent unit (GRU) and the final output of the hidden unit is used as the representation for the full document:

$$\mathbf{h}_j = \text{GRU}(\mathbf{s}_j, \mathbf{h}_{j-1}) \tag{6.2}$$

$$f(S) = \mathbf{h}_N \tag{6.3}$$

where $f(S)$ is the representation of the sentences of the document and $\mathbf{h}_N$ is the final output vector of the GRU. To capture properties of temporal coherence among document sentences, the teacher is trained to minimize $\mathcal{L}_{abs}$, the cosine similarity between the sentence embedding from reading the sentences in the forward order, $\overrightarrow{S}$, and from reading the sentences in the reverse order, $\overleftarrow{S}$:

$$\mathcal{L}_{abs} = \frac{\langle f(\overrightarrow{S}), f(\overleftarrow{S}) \rangle}{\|f(\overrightarrow{S})\| \|f(\overleftarrow{S})\|} \tag{6.4}$$

Intuitively, by parametrizing only relations between *sentences* (with the GRU layer) and not those between *words*, the teacher only captures sentence ordering properties. When training the neural generator (§6.5), we use this learned teacher to generate a reward that judges the generated sequence's ordering similarity to the gold sequence.

6.3.3 Relative Order Teacher

While the absolute ordering teacher evaluates the temporal coherence of the entire generation, we may want our teacher to be able to judge finer-grained patterns between sentences. In recipes, for example, where sentences correspond to process steps, the teacher should capture implicit script knowledge (Schank and Abelson, 1975) among groups of sentences. Consequently, the teacher should reward sentences individually for how they fit with surrounding sentences.

In many current approaches for using policy gradient methods to optimize a model with respect to a

global score, each sentence receives the same reward. This framework assumes each sentence is equally responsible for the reward gathered by the full sequence, allowing potentially appropriate subsequences to be incorrectly penalized. We design the relative order teacher to address this issue.

The relative order teacher is trained in the same way as the absolute order model. A bag of words embedding is computed for each sentence in the gold sequence. Subsequences of the gold document that have ℓ sentences are selected where $\ell \in (\ell_{min}, \ell_{max})$. For a subsequence beginning at sentence j, the model computes:

$$f(S_{j:j+\ell}) = \text{GRU}(\mathbf{s}_{j+\ell}, \mathbf{h}_{j+\ell-1}) \tag{6.5}$$

where $f(S_{j:j+\ell})$ is the encoded representation of sentences $\{s_j, ...s_{j+\ell}\}$ and $\mathbf{h}_{j-1}$ would be initialized as a vector of zeros. The relative ordering teacher is trained to minimize $\mathcal{L}_{rel}$, the cosine similarity between gold orders of subsequences:

$$\mathcal{L}_{rel} = \frac{\langle f(\overrightarrow{S}_{j:j+\ell}), f(\overleftarrow{S}_{j:j+\ell}) \rangle}{\|f(\overrightarrow{S}_{j:j+\ell})\| \|f(\overleftarrow{S}_{j:j+\ell})\|} \tag{6.6}$$

where the arrow above S signifies the order in which the sentences are processed. The relative ordering teacher learns to identify local sentence patterns among ordered sentences, thereby learning how to reward sequences that are temporally coherent.

6.4 Generator Architecture

In the task of recipe generation, the model is given a title of a recipe such as *"Cheese Sandwich"* and a list of ingredients (e.g., cheese, bread, etc.) and must generate the full multi-sentence recipe text. Similar to data to document generation tasks, the model must generate a full long-form text from sparse input signal, filling in missing information on its own (Wiseman et al., 2017).

6.4.1 Notation

Using the same notation as Kiddon et al. (2016), we are given a set of recipe title words $\{g_1, ..., g_m\}$ (e.g., { "cheese", "sandwich" }) and a list of ingredients $\mathcal{E} = \{e_1, ..., e_I\}$ where each e can be a single- or multi-

word ingredient phrase (e.g., "onions" or "onions, chopped"). In the following paragraphs, all $\mathbf{W}$ variables are projection matrices and all $\mathbf{b}$ variables are bias vectors.

6.4.2 Encoder

We use a modification of the baseline encoder of Kiddon et al. (2016). First, the title words are encoded as a bag of embeddings, $\mathbf{h}^g$. Second, each ingredient phrase is encoded as a bag of embeddings vector, $\mathbf{e}_i$. The ingredient embeddings $\mathbf{e}_i$ are inputs to a bidirectional gated recurrent unit, which yields an output vector $\mathbf{h}^e$. The final encoder output is the concatenation of these two representations, $[\mathbf{h}^g, \mathbf{h}^e]$.

6.4.3 Decoder

The decoder is a separate gated recurrent unit that receives $[\mathbf{h}^g, \mathbf{h}^e]$ from the encoder to initialize its hidden state $\mathbf{h}_0^d$ and must generate a full recipe word by word. At each time step, the model receives an input token embedding, $\mathbf{x}_t$, as well as the output from the encoder $[\mathbf{h}^g, \mathbf{h}^e]$:

$$a_t = \sigma(\mathbf{W}_1 \mathbf{h}_{t-1}^d + \mathbf{W}_2 \mathbf{x}_t + \mathbf{b}_1) \tag{6.7}$$

$$\mathbf{z}_t = a_t \mathbf{h}_0^d \tag{6.8}$$

$$\tilde{\mathbf{x}}_t = [\mathbf{x}_t, \mathbf{z}_t] \tag{6.9}$$

where $\tilde{x}_t$ is the input to the recurrent unit at every time step. The recipe generator is pretrained to minimize the negative loglikelihood of predicting the next token in the recipe:

$$L_{mle} = -\sum_{t=1}^{T} \log P(x_t | x_0, ..., x_{t-1}, \mathbf{h}^g, \mathbf{h}^e) \tag{6.10}$$

where $\mathbf{h}^g$ and $\mathbf{h}^e$ are the encoded representation of the title and ingredients from Section 6.4.2, respectively, and T is the number of words in the gold recipe.

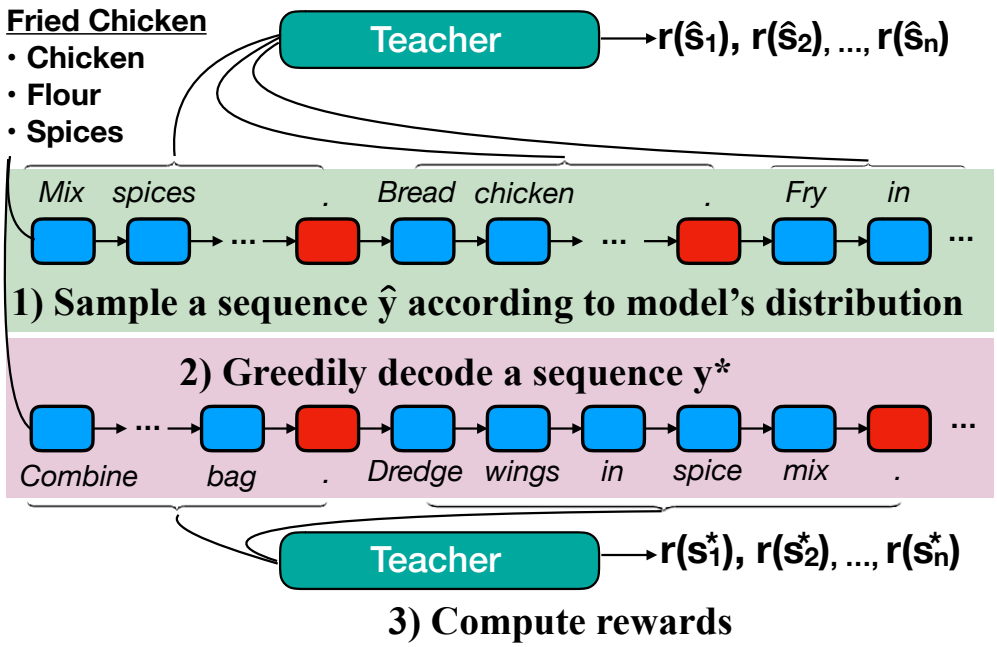


Figure 6.3: In our learning framework, our model generates a recipe by randomly sampling from the output distribution over its vocabulary. The generated recipe sentences are then passed to the learned teacher function (§6.3), which returns a reward for each token in the recipe. Simultaneously, a baseline recipe (i.e., an approximation of the "best" sequence the model can produce) is argmax decoded using the same initial ingredients and recipe title. The teacher computes rewards for the baseline sequence too. If the reward of the randomly sampled recipe is greater than that of the baseline recipe, the model learns to increase the likelihood of generating the sampled recipe in the future.

6.5 Policy Learning

Training a recipe generation model using maximum likelihood estimation produces generations that are locally coherent, but lack understanding of domain knowledge. By using a teacher that rewards the model for capturing cooking recipe discourse semantics, the model learns a policy that produces generations that better model the underlying recipe process. We learn a policy using the self-critical approach of Rennie et al. (2017).

6.5.1 Self-critical Sequence Training

In self-critical sequence training, outlined in Figure 6.3, the model learns by being rewarded for sampling sequences that receive more reward than a greedily decoded sequence. For each training example, a sequence $\hat{y}$ is generated by sampling from the model's distribution $P(\hat{y}_t|\hat{y}_0, ..., \hat{y}_{t-1}, \mathbf{h}^g, \mathbf{h}^e)$ at each time step t. Once

the sequence is generated, the teacher produces a reward $r(\hat{y}_t)$ for each token in the sequence. A second sequence y^* is generated by argmax decoding from the distribution $P(y_t^*|y_0^*, ..., y_{t-1}^*, \mathbf{h}^g, \mathbf{h}^e)$ at each time step t. The model is trained to minimize:

$$\mathcal{L}_{rl} = -\sum_{t=1}^{T}(r(\hat{y}_t) - r(y_t^*)) \log P(\hat{y}_t|\hat{y}_0, ..., \hat{y}_{t-1}, \mathbf{h}^g, \mathbf{h}^e) \tag{6.11}$$

where $r(y_t^*)$ is the reward produced by the teacher for tokens of the greedily decoded sequence. Because $r(y^*)$ can be viewed as a *baseline* reward that sampled sequences should receive more than, the model learns to generate sequences that receive more reward from the teacher than the best sequence that can be greedily decoded from the current policy. This approach allows the model to *explore* sequences that yield higher reward than the current best policy.

6.5.2 Rewards

As we decode a sequence $y = \{y_0..., y_t\}$, we track a sentence index that is the number of sentence delimiter tokens (e.g., "."} generated by the model. The model then implicitly decodes a set of generated sentences, $\hat{S} = \{\hat{s}_0, ..., \hat{s}_n\}$. These sentences are provided to the teachers defined in Section 6.3, which compute a score for the generated sequence. We explain the procedure for producing a token reward $r(y_t)$ from these scores below.

Absolute Order Once a sequence has been generated, the absolute order teacher computes a reward for y in the following way:

$$r_{abs}(y) = \frac{\langle f(\hat{S}), f(\overrightarrow{S})\rangle}{\|f(\hat{S})\|\|f(\overrightarrow{S})\|} - \frac{\langle f(\hat{S}), f(\overleftarrow{S})\rangle}{\|f(\hat{S})\|\|f(\overleftarrow{S})\|} \tag{6.12}$$

where $\overrightarrow{S}$ is the forward-ordered corresponding gold sequence and $\overleftarrow{S}$ is the reverse-ordered gold sequence. Both terms in the reward computation are variations of the loss function on which the absolute order teacher was trained (Equation 6.4). This reward compares the generated sequence to both sentence orders of the gold sequence, and rewards generations that are more similar to the forward order of the gold sequence. Because the cosine similarity terms in Equation (6.12) are bounded in $[-1, 1]$, the model receives additional reward for generating sequences that are different from the reverse-ordered gold sequence.

Relative Order Similarly, the relative order reward is generated by the relative order teacher (§6.3.3), which evaluates subsequences of sentences, rather than the whole sequence. For a sentence s_j, the reward is computed as:

$$r_{rel}(s_j) = \frac{1}{L} \sum_{\ell=\ell_{min}}^{\ell_{max}} \left(\frac{\langle f(\hat{S}_{j-\ell:j}), f(\overrightarrow{S}_{j-\ell:j}) \rangle}{\|f(\hat{S}_{j-\ell:j})\| \|f(\overrightarrow{S}_{j-\ell:j})\|} - \frac{\langle f(\hat{S}_{j-\ell:j}), f(\overleftarrow{S}_{j-\ell:j}) \rangle}{\|f(\hat{S}_{j-\ell:j})\| \|f(\overleftarrow{S}_{j-\ell:j})\|} \right) \tag{6.13}$$

where ℓ_{min} and ℓ_{max} define the window of sentences to include in the computation of the reward. Similar to the absolute order teacher, the relative order teacher produces scores bounded in $[-1, 1]$, giving the model additional reward for generating sequences that are different from the reverse-ordered gold subsequences.

Credit Assignment When rewarding tokens with the absolute ordering teacher, each generated token receives the same sequence-level reward from the absolute order teacher:

$$r(y_t) = r_{abs}(y) \tag{6.14}$$

The relative order teacher, meanwhile, computes rewards for sentences based on their imitation of nearby sentences in the gold recipe. Rather than combining all rewards from the teacher to compute a full sequence reward, sentences should only be rewarded for their own quality. Each token in a sentence corresponds to a position in the full sequence. When relative order rewards are computed by the teacher, the correct sentence reward is indexed for each token. Consequently, when training with a relative order teacher, words only receive rewards for the sentences they belong to:

$$r(y_t) = \sum_{j=1}^{|S|} \mathbb{1}(y_t \in \hat{s}_j) r_{rel}(\hat{s}_j) \tag{6.15}$$

where $|S|$ is the number of sentences in the generated recipe, and $\mathbb{1}$ is an indicator variable identifying word y_t belonging to sentence s_j.

6.5.3 Mixed Training

As the model learns parameters to optimize the amount of reward it receives from the teacher, it is not explicity encouraged to produce fluent generations. The model quickly learns to generate simple sequences

Title: Chili Grits
Ingredients: boiling water, butter, shredded cheddar cheese, jalapenos, eggs, chicken cream of soup, salt
Generated Recipe: Here .

Figure 6.4: Recipe generated from a self-critical model with no mixed training

that *exploit* the teacher for high rewards despite being incoherent recipes (e.g., Figure 6.4). Consequently, it is possible that generated sequences are no longer readable (Pasunuru and Bansal, 2017; Paulus et al., 2018). To remedy this effect, the model optimizes a mixed objective that balances learning the discourse-focused policy while maintaining the generator's language model:

$$\mathcal{L}_{mix} = \gamma \mathcal{L}_{rl} + (1 - \gamma)\mathcal{L}_{mle} \tag{6.16}$$

where L_{mle} is the objective from Equation (6.10), L_{rl} is the objective from either Equation (6.11), and γ is a hyperparameter in [0, 1].

6.6 Experimental Setup

6.6.1 Datasets

We use the Now You're Cooking dataset with the same training/test/development splits from Kiddon et al. (2016). For training, we use 109567 recipes with 1000 recipes set aside for both development and test. Recipes range from 3 to 814 tokens with an average of 103 tokens. They range 1 to 94 sentences with an average of 9.8 sentences. Each recipe is batched based on the number of tokens and number of ingredients it has.

6.6.2 Training

Teacher Models The teachers are trained before the recipe generator and their parameters are fixed during generation. We tune hyperparameters on the development set. To train the relative order teacher, we sample 20 subsequences from each recipe of $\ell_{min} = 3$ to $\ell_{max} = 6$ sentences. The hidden size of the reward generator is 100, the word embeddings have dimensionality 100. We use dropout with a rate of 0.3 between the bag of words layers and the recurrent layers.

Recipe Generator We pretrain a recipe generator using a variant of the encoder-decoder baseline from Kiddon et al. (2016). We use a hidden size of 256 for the encoder and 256 for the decoder. We initialize three different sets of embeddings for the recipe titles, ingredient lists, and text, each of size 256. All models are trained with a dropout rate of 0.3 and are based on a single-layer GRU. We use a temperature coefficient of $\beta = 2$ to make the output word distribution more peaky (Kiddon et al., 2016), allowing for more controlled exploration during self-critical learning. We use scheduled sampling with a linear decay schedule of 5% every 5 epochs up to a max of 50%. We use a learning rate of $\eta = 0.0003$ and train with the Adam optimizer. We use a minibatch size of 32 for training.

Policy Learning We train a different model for three different teacher-provided rewards: absolute ordering (AO), relative ordering (RO) and a joint reward of relative ordering and BLEU-4 (RO + B4), where the full-sequence BLEU-4 reward and the sentence-level relative ordering reward are summed at each time step. The best model for the absolute and relative ordering rewards are the ones that receive the highest average reward on the development set. The best model for the mixed reward was chosen as the one that achieved the highest average geometric mean of BLEU-4 reward and average relative ordering reward for each generated sequence y in the development set:

$$\bar{r} = \frac{r_{b4}(y)}{T} \sum_{t=1}^{T} r_{RO}(y_t) \tag{6.17}$$

where r_{b4} is the BLEU-4 score of the whole generated sequence, and r_{RO} is computed using Equation (6.15). Our best models use $\gamma = 0.97$ when training with the mixed objective from Equation (6.16).

We use the same model hyperparameters as during pretraining, but re-initialize the Adam optimizer, use $\eta = 3 \times 10^{-5}$ as the learning rate, and do not train with scheduled sampling.

6.6.3 Baselines

As baselines, we report results for a model trained only with cross-entropy loss (MLE) and for re-implemented versions of models from Rennie et al. (2017) and Paulus et al. (2018). These baselines achieved state of the art results in image captioning and document summarization tasks. We found, however, that their high γ (1 and 0.9984, respectively) led to low fluency, resulting in reduced performance on word-level scores. To

control for this effect, we trained additional versions of each baseline with different values for γ and report the best performing configurations (see Table 6.1).

Baseline Selection For each baseline we trained, we report the score of the γ setting that achieved the highest score for the metric on which it was trained. For example, for baselines trained with ROUGE-L reward, we report the results for the model trained with the value of γ that scored the highest ROUGE-L score on the development set. For the models trained with the CIDEr reward, we select the model with value of γ that achieved the highest CIDEr score on the development set. We do the same for models trained with BLEU-1 and BLEU-4 rewards. The values of γ yielding the best performance on the development set were 0.97 for the BLEU-1, ROUGE-L, and CIDEr-trained models, and 0.99 for the BLEU-4 trained baseline. For each baseline, the best model is chosen by selecting the checkpoint that achieves the highest reward (or lowest loss for the MLE model) for the metric it was trained on.

6.7 Experimental Results

6.7.1 Overlap Metrics

Scores We compute the example-level BLEU-1, BLEU-4, and ROUGE-L (R-L) scores for all recipes in the test set. A generated recipe, however, must be coherent at both the *word-level*, linking words and phrases sensibly, and the *world-level*, describing events that are grounded in real-world actions. Because n-gram scores do not evaluate if a generated recipe models this latent process, we also report these scores on the *action* and *state change* sequence described in the recipe. These words depict a *simulated* world where actions are taken and state changes are induced. A generated recipe should follow the sequence of actions taken in the gold recipe, and induce the same state changes as those in the gold recipe.

We use the state change lexicon from Chapter 5 to map recipe words to ordered sequences of actions and state changes. Each entry in the lexicon contains an action in the cooking domain as well as the state changes that result from that action in the set of {LOCATION, COMPOSITION, COOKEDNESS, TEMPERATURE, SHAPE, CLEANLINESS}.

Action sequences are formed by mapping lemmas of words in generated sequences to entries in the lexicon. We compare these event sequences to the gold event sequences using the same scores as for words

Model	BLEU-1	BLEU-4	R-L	AB1	AB4	AR-L	SCB1	SCB4	SCR-L
Cross-entropy (MLE)	26.86	4.74	28.86	31.23	4.83	28.51	51.92	26.35	50.21
BLEU-4 (Rennie et al., 2017)	7.75	1.38	13.93	5.69	0.84	10.37	10.76	5.05	20.87
CIDEr (Rennie et al., 2017)	12.67	1.90	21.20	14.61	1.79	21.70	26.07	12.30	41.65
ROUGE-L (Paulus et al., 2018)	29.00	4.86	29.10	33.49	4.73	28.11	56.86	27.83	51.26
BLEU-1 ($\gamma = 0.97$)	**31.16**	**5.60**	29.53	32.28	5.09	29.34	52.63	25.43	51.58
BLEU-4 ($\gamma = 0.99$)	30.56	5.42	29.16	32.53	4.99	28.99	53.48	26.35	51.02
CIDEr ($\gamma = 0.97$)	29.60	5.10	28.79	33.93	4.81	28.41	57.00	27.55	50.57
ROUGE-L ($\gamma = 0.97$)	26.88	4.66	29.49	31.85	5.01	29.25	53.84	26.77	51.88
Absolute Ordering (AO)	23.70	4.25	28.43	28.22	4.44	27.88	47.93	24.47	50.15
Relative Ordering (RO)	27.75	4.88	29.60	34.37	**5.60**	**29.36**	58.31	29.14	**53.08**
Relative Ordering + BLEU-4	29.58	5.26	**29.78**	**35.13**	5.55	29.33	**59.13**	**29.19**	52.46

Table 6.1: Evaluation results for generated sequences by models and baselines. We **bold** the top performing result. The second to fourth columns list word-level scores. Columns AB1, AB4, and AR-L list action-level scores (§6.7.1). Columns SCB1, SCB4, and SCR-L list state change level scores (§6.7.1).

– BLEU-1, BLEU-4, and ROUGE-L. Intuitively, these scores can be seen as evaluating the following: whether the generated recipe depicts the same actions (AB1), subsequences of consecutive actions (AB4), and full action sequence (AR-L) as the gold recipe.

State change sequences are more coarse-grained than action sequences, and are formed by mapping actions to their state changes in the lexicon from Bosselut et al. (2018a). These scores evaluate whether the generated recipe implies the same induced state changes (SCB1), subsequences of consecutive state changes (SCB4), and global state change order (SCR-L) as the gold recipe.

Results Our results in Table 6.1 show that models optimized on word overlap metrics achieve the greatest improvements for those scores. Optimizing scores such as BLEU-1 encourages the model to output words and phrases that overlap often with reference sequences, but that may not describe main events in the recipe process.

When examining models trained using a neural teacher, we see that the model optimized with the absolute ordering reward performs worse than most baselines for every word-level score. The relative ordering model, however, raises every word-level score above the cross-entropy baseline, indicating the importance of fine-grained credit assignment at the sentence-level. The model trained with mixed rewards from the teacher and BLEU-4 achieves even higher scores, showing the benefits of training with diverse rewards.

MLE	RO + B4	Tie		BLEU-1	RO + B4	Tie
0.330	**0.447**	0.223	Fluency	**0.387**	0.373	0.240
0.350	**0.440**	0.210	Ingredient Use	0.327	**0.363**	0.310
0.347	**0.430**	0.223	Title Completion	0.353	**0.377**	0.270
0.377	**0.453**	0.170	Action Order	**0.410**	0.403	0.187

Table 6.2: Human evaluation measuring proportion of winners. The left part of the table compares recipes from the **MLE** baseline with recipes generated from the **RO + B4** model. The right part of the table compares recipes generated from the **BLEU-1** baseline with recipes generated from the **RO + B4** model.

When evaluating these metrics for the action and state change sequence, the models trained with feedback from the relative ordering teacher show large improvement over the baselines, indicating that the models exhibit more understanding of the latent process underlying the task. While optimizing word-level scores teaches the generator to output common sequences of words, the relative ordering reward teaches the model to focus on learning co-occurrences between recipe events.

6.7.2 Human Evaluation

We perform a human evaluation on 100 recipes sampled from the test set to evaluate our model on four aspects of recipe quality: fluency, ingredient use, title completion, and action ordering. For a particular recipe title and ingredient, three judges from Amazon Mechanical Turk are shown a pair of generated recipe steps, each produced by a different model, and asked to select the recipe that is better according to the four aspects. For ingredient use, judges are asked to select the recipe that uses more of the ingredients correctly. For title completion, we ask judges to select the recipe that best completes the dish described in the recipe title. Finally, for action ordering, judges are asked to choose the recipe that better links subtasks in the recipes.

Models We use generated recipes from the Relative Ordering + BLEU-4 model (**RO + B4**) and compare it to to recipes generated from two baselines: the cross-entropy model (**MLE**), and the **BLEU-1** model, which achieved the best scores on several word-level metrics (§6.7.1).

Results We report results in Table 6.2. In the left part of the table, we show the results of the **RO+B4** model being compared to the **MLE** model. Our model outperforms the cross-entropy baseline (**MLE**),

MLE	RO + B4	Tie		BLEU-1	RO + B4	Tie
0.317	**0.425**	0.258	Fluency	**0.391**	0.383	0.225
0.342	**0.458**	0.200	Ingredient Use	0.267	**0.392**	0.342
0.358	**0.450**	0.192	Title Completion	0.325	**0.418**	0.258
0.367	**0.483**	0.150	Action Order	0.433	**0.442**	0.125

Table 6.3: Proportion of winners for **long** generated recipes. The left part of the table compares recipes from the **MLE** baseline with recipes generated from the **RO + B4** model. The right part of the table compares recipes generated from the **BLEU-1** baseline with recipes generated from the **RO + B4** model.

consistently being preferred on aggregate for every question. When compared to the **BLEU-1** baseline (in the right part of the table), workers preferred the **BLEU-1** baseline for the fluency and action order questions, while preferring recipes generated by the teacher-trained model for the ingredient use and title ordering questions. Upon further analysis, we see that the strength of the **BLEU-1** model depends on the length of the original reference sequence. The precision-guided objective of the BLEU-1 reward causes the model to prefer generating short sequences. To evaluate this effect, in Table 6.3, we aggregate evaluation scores only for recipes where the gold recipe was originally longer than 100 words. The performance of the teacher-trained model rises compared to the **BLEU-1** model for every question, showing that modeling discourse structure as a learned reward improves global coherence in long generated texts.

Title **Ings**	Royale Casserole Broccoli, chopped; butter; flour; salt; black pepper; nutmeg; milk; cheese shredded swiss; lemon juice; cooked cubed turkey; mushrooms; grated parmesan cheese; biscuits, refrigerated
MLE	Cook broccoli according to package directions. Drain. Melt butter in skillet. Add flour, stirring constantly. Cook, stirring constantly, until thickened. Add milk, salt, pepper, and nutmeg. Cook over low heat, stirring constantly, until thickened. Add cheese, salt, pepper and pimento. Cook and stir till thickened. Add cheese, stirring until melted. Add **broccoli and mushrooms**. Cook over low heat for 5 minutes. Add cheese. Cook and stir until cheese melts. Add cheese. Cook and stir until cheese melts. Add cheese. Cook and stir until cheese melts. Add cheese and stir until melted. Serve over hot noodles.
RO	Melt butter in saucepan. Add flour and cook, stirring constantly, until thickened. Add milk, salt, pepper, and nutmeg. Cook, stirring constantly, until thickened. Add cheese, salt and pepper. Cook over low heat, stirring constantly, until mixture is thickened and smooth. Remove from heat. Stir in cheese. Spoon into **greased casserole**. Top with cheese. **Bake in 350 f oven for 30 minutes.** Serves 6.
Gold	Preheat oven to 375. Melt butter in saucepan. Blend in flour, salt, pepper, and nutmeg; cook 1-2 minutes. Gradually add milk; cook, stirring, until slightly thickened. Stir in frozen vegetables. Remove from heat; stir in cheese until melted. Add lemon juice, turkey, mushrooms and broccoli. Pour mixture into a lightly greased baking dish; sprinkle with parmesan and top with biscuits. Bake 20 minutes, or until biscuits are golden brown.

Table 6.4: Example recipe generation from our model and comparative baselines. **Boxed** spans indicate recipe events missed by another model's generation. Red spans indicate superfluous events. The **Ings** row lists the ingredients (separated by semicolons) provided to make the dish in the title.

Title **Ings**	Wine Sauerkraut butter; dry white wine; 16 oz drained sauerkraut
MLE	In a saucepan, combine soup, wine, and butter. Heat, uncovered, in microwave oven 2 minutes or until mixture is heated through.
RO	Melt butter in skillet. **Add sauerkraut** and wine; heat to boiling. Reduce heat; cover and simmer 15 minutes. Add wine and heat to boiling; reduce heat. Cover and cook 10 minutes or until mixture is tender. Serve on rice.
Gold	Melt butter in 10-inch skillet over low heat; add sauerkraut. Cover and cook, stirring occasionally, 30 minutes; add wine. Heat to boiling; reduce heat. Cover and simmer until liquid is absorbed, about 45 minutes. Follow directions except simmer until liquid is absorbed, about 1 hour.

Table 6.5: Example recipe generations from our model and comparative baselines. **Boxed** spans indicate recipe events missed by another model's generation. Red spans indicate superfluous events. The **Ings** row lists the ingredients (separated by semicolons) provided to make the dish in the title.

Title **Ings**	Strawberry Glazed Pie fresh strawberries; granulated sugar; salt; cornstarch; lemon juice; baked, cooled pie shell
MLE	Combine sugar, cornstarch and salt in top of a double boiler. Cook and stir over medium heat until mixture thickens and boils. Boil and stir 1 minute. Remove from heat and stir in lemon juice. Cool. Add strawberries and lemon juice. Pour into baked pie shell. Top with whipped cream. Serve immediately.
RO	Combine sugar, cornstarch, and salt in a saucepan; cook over medium heat, stirring constantly, until mixture boils and thickens. Remove from heat; stir in lemon juice. Cool. Add strawberries; mix well. Pour into baked pie shell. Bake in preheated 325-degree oven for 10 minutes. Cool. Garnish with whipped cream.
Gold	Wash, drain thoroughly, and hull strawberries. Arrange about 3 cups of whole berries over bottom of baked pastry shell. Crush remaining berries in a saucepan. In a bowl, mix sugar, salt and cornstarch; stir into crushed berries. Heat slowly, stirring constantly, until mixture comes to a boil and thickens. Remove from heat and stir in lemon juice. Cool, then spoon over berries in pie shell chill until glaze is set. Garnish with whipped cream.

Table 6.6: Example recipe generations from our model and comparative baselines. **Boxed** spans indicate recipe events missed by another model's generation. Red spans indicate superfluous events. The **Ings** row lists the ingredients (separated by semicolons) provided to make the dish in the title.

6.7.3 Insights

Qualitative Analysis In Tables 6.4, 6.5, 6.6, we see the effect that the neural teacher has on the recipe generator. The teacher rewards behavior that more closely imitates the actions in the gold recipe. In the example in Table 6.4, the generator learns to complete the actions of placing the mixture into a *greased casserole* and then *baking* it, which the MLE model misses. The teacher also discourages repetitive phrases, as they provide no increase in reward during training. One weakness of our teacher models, however, is that they encourage common temporal patterns, such as in the example in Table 6.6, where the generator mentions *baking the pie*. The model recognizes pies are generally supposed to be baked, even if it is not appropriate for that particular recipe.

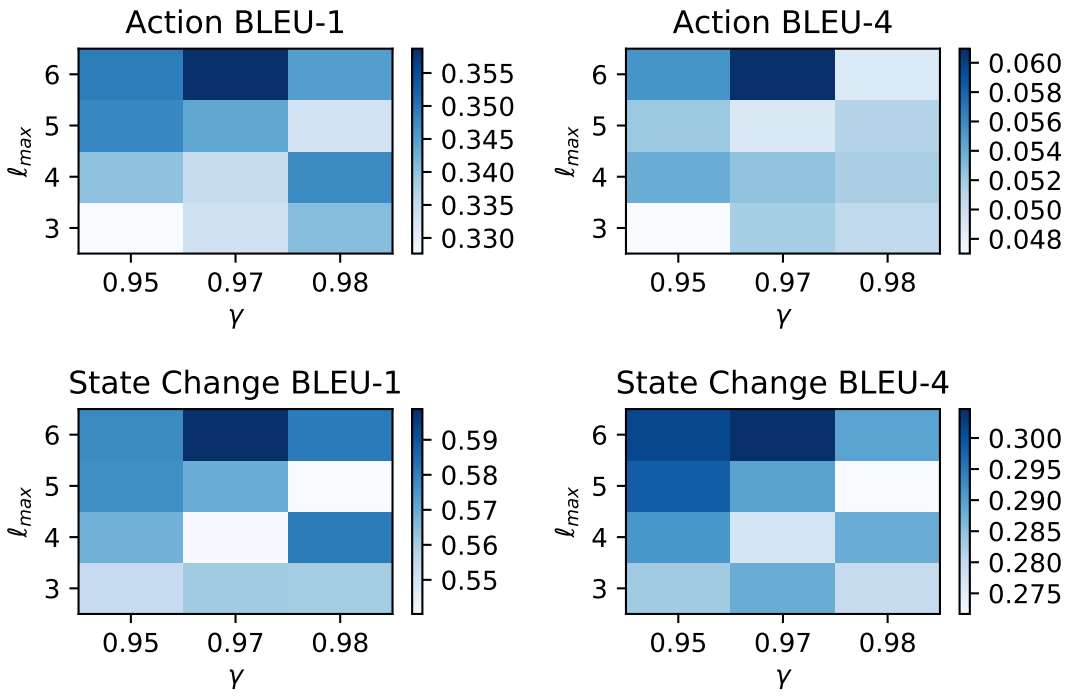


Figure 6.5: Action and State Change BLEU Metrics for different initializations of ℓ_{max} and γ

Teacher Feedback Frequency We design the reward functions in Eq. 6.12 and Eq. 6.13 to require two passes through the teacher, one comparing the generated sequence to the forward gold sequence, and one comparing it to the reverse gold sequence. With no teacher comparison to the reverse-ordered sequence, the generator learns to *exploit* the teacher for reward with very simple sequences such as “Serve." and “Here’s direction." When comparing with both orders, however, this effect is dampened, hinting at the importance of ensembling feedback from multiple sources for robust reward production. Another solution to this effect was mixing policy learning and maximum likelihood learning (Eq. 6.16) as the underlying language model of the generator did not deteriorate.

Impact of ℓ_{max} and γ Two hyperparameters to tune when training with teacher models are the mixed loss coefficient γ, which balances MLE learning with policy learning, and [ℓ_{min}, ℓ_{max}], the number of sentences to consider when computing the relative order reward. We fix $\ell_{min} = 3$, and vary $\ell_{max} \in [3, 6]$ and $\gamma \in \{0.95, 0.97, 0.98\}$. Figure 6.5 shows the importance of tuning γ. A low γ will not allow the teacher to guide the model’s learning, while a high γ causes the language model to deteriorate. Interestingly, a

higher ℓ_{max} leads to better performance on global coherence scores, implying that relative order rewards conditioned on more sentences allow the model to learn longer-range context co-occurrences.

6.8 Summary

In this chapter, we explore how to integrate temporal commonsense knowledge into neural text generators through rewarded interaction. We introduce the *absolute ordering* and *relative ordering* teachers, which score a sequence's adherence to discourse structure and temporal coherence in long text. The teachers are used to compute rewards in a self-critical reinforcement learning framework, allowing a recipe generator to be rewarded for producing sequences that exhibit the temporal semantics of the cooking domain. Empirical results demonstrate that our teacher-trained generator better models the latent event sequences of cooking recipes than competitive baselines. Our human evaluation shows that this improvement is mainly due to maintaining semantic coherence in longer recipes where neural generators produce more degenerate text (Holtzman et al., 2020).

Chapter 7

Conclusions & Future Work

7.1 Conclusions

In recent years, commonsense knowledge representation and reasoning has regained prominence as a research direction, particularly as deep learning has emerged as a powerful tool for solving natural language understanding and generation problems. In this dissertation, we explored the interplay between natural language understanding algorithms and commonsense reasoning. Specifically, we investigated how to endow question answering, state tracking, and text generation systems with the ability to use commonsense knowledge to reason about unseen implications of written language. These studies focused on two sub-problems of endowing machines with commonsense knowledge: how to construct large-scale commonsense knowledge representations that machines can easily and robustly access, and how to integrate available commonsense knowledge into downstream models that require it.

Our first study, presented in Chapter 3, developed a solution for the first of these problems, and proposed a novel framework, COMET, for representing commonsense knowledge graphs with neural models. Our framework allows us to learn *neural representations of knowledge bases*, which can produce commonsense knowledge on-demand for any entity, event, or concept that can be described by natural language. Furthermore, COMET can be initialized using large-scale language models trained on sizable textual corpora. As a result, it learns to transfer the representations learned from text to hypothesizing knowledge through language generation, allowing it to produce high-quality knowledge about concepts it is queried

about even if it has never seen examples of these concepts in a knowledge base. Our empirical studies across two diverse knowledge bases demonstrated that COMET can generate tuples of high-quality, novel commonsense knowledge on-demand.

We used the resultant approach in our second study in Chapter 4, where COMET was tasked with dynamically generating commonsense inferences about presented situations in text. The generated inferences formed a local graph of contextual commonsense knowledge about the situation, with edge scores approximating the confidence of each commonsense inference with respect to the depicted situation. To reason over this graph, we developed a novel inference algorithm that aggregated different paths in the graph to answer given questions about the situation. Our empirical studies showed that commonsense knowledge models could be used out of the box to provide interpretations of situations with no additional training, and correctly answer questions about those situations.

In Chapter 5, we showed how commonsense knowledge could be directly integrated into the parameters of a neural architecture, and how reasoning over that knowledge could be achieved through the network's operations. Our model, the neural process network, encoded semantic actions as unique functions and used these action functions to change dynamic state representations of entities. Entity transformations could solely be caused by having action functions applied to them, and the function representations were trained to induce correct state change predictions downstream. As a result, the neural process network could be used to formulate commonsense-grounded representations of entities that reflected the state changes they had undergone at any point in time. Empirical studies showed that these entity representations were more helpful inputs for downstream tasks such as generating future steps in the narrative compared to using purely text-derived representations. Furthermore, our analysis showed that the internal dynamics of the network reflected the inductive biases we were trying to integrate into the network.

Finally, in Chapter 6, we presented a structure-agnostic approach to integrating commonsense knowledge into neural models. We defined a new training algorithm that used reinforcement learning with neural knowledge-grounded rewards to guide a text generator to produce coherent text. Using recipes as a testbed, we pretrained teacher networks on a large corpus of cooking recipes to understand knowledge about the temporal order of recipe steps. The resulting teacher network could score how well generated recipes depicted implicit event chains that matched the expected procedural dynamics of cooking recipes (e.g., raw

ingredients are mixed into a dough before they are put in the oven). Then, the teacher network could evaluate recipes produced by the generator and use its score to reward the generator. Using reinforcement learning, the generator learned to produce recipes that were more fluent, and that respected the temporal semantics of the cooking domain more consistently. Importantly, as the length of the recipes increases, this improvement became more pronounced compared to baseline models with no knowledge integration.

7.2 Future Research Plans

Our studies on representing and reasoning about commonsense knowledge showed the potential for augmenting NLP systems with broader capabilities that would allow them to understand new situations in text. Below, we outline future directions for addressing new challenges arising from these successes, as well as new application settings for integrating commonsense reasoning capacities into NLP systems.

7.2.1 Expressive Knowledge Representations

Our work on COMET showed the potential for using neural representations of knowledge bases as opposed to static structures of explicit knowledge. The ability to learn a neural knowledge base that generalizes from a limited number of examples implies that much knowledge is already encoded in the initial language model's parameters, and that fine-tuning on knowledge tuples is a way of learning an interface to this knowledge. The continued performance improvements on common benchmarks (Wang et al., 2018, 2019) due to scaling up the width and parameter counts of transformer language models (Shoeybi et al., 2019; Raffel et al., 2019), thereby increasing their capacity to learn knowledge, seem to support this conclusion, as well. However, it is clear that continuing to scale up knowledge has its limits, and that combining neural knowledge bases jointly with information retrieval is an area of future benefits (Guu et al., 2020; Lewis et al., 2020). Our future work will explore this idea further, focusing on formalizing the types of knowledge that are more suited to certain representations (e.g., a neural knowledge base, or a more traditional static database).

7.2.2 Robust Reasoning with Commonsense Knowledge

Our initial studies in injecting natural language understanding systems with commonsense knowledge (Chapters 4, 5) showed how commonsense reasoning capabilities allow systems to adapt more robustly to new,

unseen situations. However, there is still considerable room for improvement, particularly as we develop agents for more general domains with less curated, in-domain training data (e.g., conversation, story generation, creativity). Current approaches for addressing the commonsense deficit are promising, but continue to rely on fixed ontologies, such as ConceptNet (Speer et al., 2017), that are difficult to scale to the situational diversity required by these tasks.

We will continue to explore new ways of connecting neural commonsense representation models to challenging reasoning tasks with limited data. Current studies have already explored how knowledge stored in neural representations of knowledge bases can be used in a variety of downstream tasks such as sarcasm generation (Chakrabarty et al., 2020), medical dialogue (Kearns et al., 2020), and reading comprehension (Liu et al., 2020). Their adoption in these areas is a result of their ability to produce on-demand knowledge that can be reasoned over using probabilistic methods, deep learning, rules, or ensembles of them. Our initial work in this space will specifically focus on designing short story generation systems that can use commonsense knowledge to drive stories forward. Neural generation models are generally challenged in the task of story generation because narrative structure is difficult to discover using standard data-driven approaches (Chapter 6). By providing commonsense knowledge about event relationships as an additional input, we will explore whether more interesting and coherent narratives can be generated by automated storytelling agents.

7.2.3 Evaluation of Commonsense Reasoning

Designing evaluation settings that directly test commonsense reasoning problems remains a challenge (Sakaguchi et al., 2020; Bhagavatula et al., 2020). Our own work in constructing datasets for recipe state tracking (Bosselut et al., 2018a), emotional state tracking (Rashkin et al., 2018), multimodal knowledge extraction (Bosselut et al., 2016), and counterfactual reasoning (Qin et al., 2019; Tandon et al., 2019) has demonstrated the importance of careful design when testing new hypotheses related to commonsense knowledge understanding and reasoning. Our future work in designing reasoning systems will also focus on constructing new datasets and metrics that target key elements of commonsense understanding. Importantly, we will prioritize building testbeds that are less sensitive to models that learn to exploit annotation artifacts (Gururangan et al., 2018), dataset biases (Zhao et al., 2017), and simple surface patterns (Lucy and Gauthier, 2017).

7.2.4 Foundations of Commonsense Knowledge

Commonsense knowledge is often described as the everyday facts that *all* humans generally know. However, cultural, linguistic, historical, geographical, and societal differences between populations can affect the expected knowledge humans have and how that knowledge is expressed. For NLP agents to interact successfully with users across populations, they must understand these differences and how they contribute to a different foundation for commonsense. Our work will focus on developing commonsense knowledge learning algorithms that generalize to diverse societal norms, yielding reasoning systems that are more inclusive and equitable. In initial studies, we will work to create multi-lingual commonsense representations that can be deployed to natural language understanding applications in multiple languages using pretrained language models produced for those languages (Martin et al., 2019; Cañete et al., 2020; Polignano et al., 2019).

7.2.5 Connecting Commonsense Knowledge to Real-World Applications

There is a divide between the end tasks used by NLP researchers to test new approaches and the applications developed by NLP practitioners. A future direction of our research is to tie improvements in commonsense understanding to user applications. For example, a system might be able to understand today that inclement weather can cause slowdowns in traffic. However, we are still far away from a system that can automatically set an alarm 30 minutes earlier to ensure a user is not late to work. While intelligent agents that can understand and reason about commonsense are the end-goal, we will focus on how to deploy intermediate commonsense modeling capabilities to present systems. In initial studies, our research will explore whether a phone application can learn commonsense to better schedule and change a user's calendar when certain events are triggered.

Bibliography

Aida Amini, Antoine Bosselut, Bhavana Dalvi Mishra, Yejin Choi, and Hannaneh Hajishirzi. 2020. Procedural reading comprehension with attribute-aware context flow. *arXiv preprint arXiv:2003.13878*.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*.

Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *ICLR 2015*.

Regina Barzilay and Mirella Lapata. 2005. Modeling local coherence: An entity-based approach. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*.

Regina Barzilay and Mirella Lapata. 2008. Modeling local coherence: An entity-based approach. *Computational Linguistics*, 34(1):1–34.

Regina Barzilay and Lillian Lee. 2004. Catching the drift: Probabilistic content models, with applications to generation and summarization. In *Proceedings of the 2004 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Lisa Bauer, Yicheng Wang, and Mohit Bansal. 2018. Commonsense for generative multi-hop question answering tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *ArXiv*, abs/1711.03902.

Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Scott Yih, and Yejin Choi. 2020. Abductive commonsense reasoning. *ArXiv*, abs/1908.05739.

Olivier Bodenreider. 2004. The unified medical language system (umls): Integrating biomedical terminology. *Nucleic acids research*, 32:D267–70.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA. ACM.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795.

Antoine Bosselut, Jianfu Chen, David Warren, Hannaneh Hajishirzi, and Yejin Choi. 2016. Learning Prototypical Event Structure from Photo Albums. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Antoine Bosselut and Yejin Choi. 2019. Dynamic Knowledge Graph Construction for Zero-shot Commonsense Question Answering. *ArXiv*, abs/1911.03876.

Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018a. Simulating Action Dynamics with Neural Process Networks. In *Proceedings of the 6th International Conference for Learning Representations (ICLR)*.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Çelikyilmaz, and Yejin Choi. 2019. COMET: Commonsense Transformers for Automatic Knowledge Graph Construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Antoine Bosselut, Asli Çelikyilmaz, Xiaodong He, Jianfeng Gao, Po-Sen Huang, and Yejin Choi. 2018b. Discourse-Aware Neural Rewards for Coherent Text Generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Tom B. Brown, Benjamin Pickman Mann, Nick Ryder, Melanie Subbiah, Jean Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, G. Krüger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric J Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2018. Question answering by reasoning across documents with graph convolutional networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, Jr., and Tom M. Mitchell. 2010. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 1306–1313. AAAI Press.

José Cañete, Gabriel Chaperon, Rodrigo Fuentes, and Jorge Pérez. 2020. Spanish pre-trained bert model and evaluation data. In *PML4DC at ICLR 2020*.

Asli Celikyilmaz, Antoine Bosselut, Xiaodong He, and Yejin Choi. 2018. Deep communicating agents for

abstractive summarization. In *Proceedings of the 16th Annual Meeting of the North American Association for Computational Linguistics (NAACL)*.

Tuhin Chakrabarty, Debanjan Ghosh, Smaranda Muresan, and Nanyun Peng. 2020. R3: Reverse, Retrieve, and Rank for Sarcasm Generation with Commonsense Knowledge. *ArXiv*, abs/2004.13248.

Nathanael Chambers and Dan Jurafsky. 2009. Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 602–610. Association for Computational Linguistics.

Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. 2017. Recurrent environment simulators. *CoRR*, abs/1704.02254.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Lei Cui, Furu Wei, and Ming Zhou. 2018. Neural open information extraction. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. 2018. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, New Orleans, Louisiana. Association for Computational Linguistics.

Bhavana Dalvi, Niket Tandon, Antoine Bosselut, Wen-tau Yih, and Peter Clark. 2019. Everything happens for a reason: Discovering the purpose of actions in procedural text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on*

Natural Language Processing (EMNLP-IJCNLP), pages 4496–4505, Hong Kong, China. Association for Computational Linguistics.

Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. 2019. Building dynamic knowledge graphs from text using machine reading comprehension. In *Proceedings of the 7th International Conference on Learning Representations*.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning Journal (MLJ)*.

Joe Davison, Joshua Feldman, and Alexander Rush. 2019. Commonsense knowledge mining from pretrained models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1173–1178, Hong Kong, China. Association for Computational Linguistics.

Tim Dettmers, Minervini Pasquale, Stenetorp Pontus, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32nd Conference on Artificial Intelligence (AAAI)*, pages 1811–1818, New Orleans, USA.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Bhuwan Dhingra, Qiao Jin, Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2018. Neural models for reasoning over multiple mentions using coreference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA. ACM.

Xinya Du, Bhavana Dalvi, Niket Tandon, Antoine Bosselut, Wen tau Yih, Peter Clark, and Claire Cardie.

2019. Be consistent! improving procedural text comprehension using label consistency. In *Proceedings of the 17th Annual Meeting of the North American Association for Computational Linguistics (NAACL)*.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam. 2011. Open information extraction: The second generation. In *IJCAI*.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1535–1545. Association for Computational Linguistics.

Angela Fan, Claire Gardent, Chloé Braud, and Antoine Bordes. 2019. Using local knowledge graph construction to scale seq2seq models to multi-document inputs. *ArXiv*, abs/1910.08435.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

James Fan, David A. Ferrucci, David Gondek, and Aditya Kalyanpur. 2010. Prismatic: Inducing knowledge from a large scale lexicalized relation resource. In *Proceedings of the 2010 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Qiaozi Gao, Malcolm Doering, Shaohua Yang, and Joyce Y Chai. 2016. Physical causality of action verbs in grounded language understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1, pages 1814–1824.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *J. Artif. Intell. Res.*, 57:345–420.

Andrew S. Gordon and Jerry R. Hobbs. 2017. *A Formal Theory of Commonsense Psychology: How People Think People Think*. Cambridge University Press.

Jonathan Gordon and Benjamin Van Durme. 2013. Reporting bias and knowledge acquisition. In *Proceedings of the 2013 workshop on Automated knowledge base construction*, pages 25–30. ACM.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New Orleans, Louisiana. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. *ArXiv*, abs/2002.08909.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2016. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2015. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*.

Andrew Hoang, Antoine Bosselut, Asli Çelikyilmaz, and Yejin Choi. 2019. Efficient adaptation of pretrained transformers for abstractive summarization. *arXiv*, abs/1906.00138.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28 – 61. Artificial Intelligence, Wikipedia and Semi-Structured Resources.

Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649, Melbourne, Australia. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *Proceedings of the 8th International Conference for Learning Representations (ICLR)*.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. ACM.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Stanislaw Jastrzębski, Dzmitry Bahdanau, Seyedarian Hosseini, Michael Noukhovitch, Yoshua Bengio, and Jackie Cheung. 2018. Commonsense mining as knowledge base completion? a study on the impact of novelty. In *Proceedings of the Workshop on Generalization in the Age of Deep Learning*, pages 8–16, New Orleans, Louisiana. Association for Computational Linguistics.

Yangfeng Ji, Chenhao Tan, Sebastian Martschat, Yejin Choi, and Noah A Smith. 2017. Dynamic entity representations in neural language models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1831–1840.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yichen Jiang and Mohit Bansal. 2019. Self-assembling modular networks for interpretable multi-hop reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Yichen Jiang, N. Joshi, Yen-Chun Chen, and Mohit Bansal. 2019. Explore, propose, and assemble: An interpretable model for multi-hop reading comprehension. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

William R. Kearns, Neha Kaura, Myra Divina, Cuong Vo, Dong Si, Teresa Ward, and Weichao Yuwen. 2020. A wizard-of-oz interface and persona-based methodology for collecting health counseling dialog. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, page 1–9, New York, NY, USA. Association for Computing Machinery.

Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 982–992.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (Proceedings of the Conference on Empirical Methods in Natural Language Processing)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942.

Douglas B Lenat. 1995. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.

Hector J. Levesque. 2017. *Common Sense, the Turing Test, and the Quest for Real AI*, 1st edition. The MIT Press.

Omer Levy, Steffen Remus, Christian Biemann, and Ido Dagan. 2015. Do supervised distributional methods really learn lexical inference relations? In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke S. Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *CoNLL*.

Patrick A. Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401.

Jiwei Li and Eduard H Hovy. 2014. A model of coherence based on distributed sentence representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016a. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Lucy Li and Jon Gauthier. 2017. Are distributional representations ready for the real world? evaluating word vectors for grounded perceptual meaning. *ArXiv*, abs/1705.11168.

Xiang Li, Aynaz Taheri, Lifu Tu, and Kevin Gimpel. 2016b. Commonsense knowledge base completion. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. 2017. Paraphrase generation with deep reinforcement learning. *arXiv preprint arXiv:1711.00279*.

Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. Kagnet: Knowledge-aware graph networks for commonsense reasoning. *ArXiv*, abs/1909.02151.

Chin-Yew Lin. 2004. ROUGE: a package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain.

Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors. 2018. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Brussels, Belgium.

Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, editors. 2019. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Florence, Italy.

Changsong Liu, Shaohua Yang, Sari Saba-Sadiya, Nishant Shukla, Yunzhong He, Song-Chun Zhu, and Joyce Y Chai. 2016a. Jointly learning grounded task structures from language instruction and visual demonstration. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016b. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*.

Qiao Liu, Liuyi Jiang, Minghao Han, Yao Liu, and Zhiguang Qin. 2016c. Hierarchical random walk inference in knowledge graphs. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 445–454, New York, NY, USA. Association for Computing Machinery.

Siqi Liu, Zhenhai Zhu, Ning Ye, Sergio Guadarrama, and Kevin Murphy. 2017. Improved image captioning via policy gradient optimization of spider. *Proceedings of the 2017 IEEE International Conference on Computer Vision*.

Ye Liu, Tao Yang, Zeyu You, Wei Fan, and Philip S. Yu. 2020. Commonsense evidence generation and injection in reading comprehension. *ArXiv*, abs/2005.05240.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ryan Lowe, Michael Noseworthy, Iulian Serban, Nicolas Angelard-Gontier, Yoshua Bengio, and Joelle Pineau. 2017. Towards an automatic turing test: Learning to evaluate dialogue responses. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Li Lucy and Jon Gauthier. 2017. Are distributional representations ready for the real world? evaluating word vectors for grounded perceptual meaning. *CoRR*, abs/1705.11168.

H. P. Luhn. 1957. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317.

Chaitanya Malaviya, Chandra Bhagavatula, Antoine Bosselut, and Yejin Choi. 2020. Commonsense knowledge base completion with structural and semantic context. In *AAAI*.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, 'Eric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. 2019. Camembert: a tasty french language model. *ArXiv*, abs/1911.03894.

Abraham H Maslow. 1943. A theory of human motivation. *Psychol. Rev.*, 50(4):370.

Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in*

Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL '12, pages 523–534, Stroudsburg, PA, USA. Association for Computational Linguistics.

John W. Mccarthy. 1960. Programs with common sense.

Tzvetan Mihaylov and Anette Frank. 2018. Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*.

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. 2014. Flow graph corpus from recipe texts. In *LREC*, pages 2370–2377.

Shinsuke Mori, Tetsuro Sasada, Yoko Yamakata, and Koichiro Yoshino. 2012. A machine learning approach to recipe text processing. In *Proceedings of the 1st Cooking with Computer Workshop*, pages 29–34.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. 2011. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 227–236, New York, NY, USA. ACM.

Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. 2012. Patty: A taxonomy of relational patterns with semantic types. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1135–1145. Association for Computational Linguistics.

Arvind Neelakantan, Quoc V. Le, Martín Abadi, Andrew McCallum, and Dario Amodei. 2016. Learning a natural language interface with neural programmer. *CoRR*, abs/1611.08945.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2015. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834.

Preksha Nema and Mitesh M Khapra. 2018. Towards a better metric for evaluating question generation systems. *arXiv preprint arXiv:1808.10192*.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016. STransE: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 460–466, San Diego, California. Association for Computational Linguistics.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104:11–33.

Feng Niu. 2012. *Web-scale Knowledge-base Construction via Statistical Inference and Learning*. Ph.D. thesis, Madison, WI, USA. AAI3524067.

Eric W Noreen. 1989. *Computer intensive methods for hypothesis testing: An introduction.* Wiley, NY.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. 2015. Action-conditional video prediction using deep networks in atari games. In *NIPS*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Ramakanth Pasunuru and Mohit Bansal. 2017. Reinforced video captioning with entailment rewards. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Debjit Paul and Anette Frank. 2019. Ranking and selecting multi-hop knowledge paths to better predict human needs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *Proceedings of the 6th International Conference for Learning Representations*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (Proceedings of the Conference on Empirical Methods in Natural Language Processing)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matthew Gardner, Christopher Clark, Kenton Lee, and Luke S. Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Karl Pichotta and Raymond J. Mooney. 2016. Using sentence-level lstm language models for script inference. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Robert Plutchik. 1980. A general psychoevolutionary theory of emotion. *Theories of emotion*, 1(3-31):4.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis only baselines in natural language inference. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 180–191, New Orleans, Louisiana. Association for Computational Linguistics.

Marco Polignano, Pierpaolo Basile, Marco de Gemmis, Giovanni Semeraro, and Valerio Basile. 2019. AlBERTo: Italian BERT Language Understanding Model for NLP Challenging Tasks Based on Tweets. In *Proceedings of the Sixth Italian Conference on Computational Linguistics (CLiC-it 2019)*, volume 2481. CEUR.

Lianhui Qin, Antoine Bosselut, Ari Holtzman, Chandra Bhagavatula, Elizabeth Clark, and Yejin Choi. 2019. Counterfactual Story Reasoning and Generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/language_understanding_paper.pdf*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. In *Proceedings of the 4th International Conference for Learning Representations*.

Hannah Rashkin, Antoine Bosselut, Maarten Sap, Kevin Knight, and Yejin Choi. 2018. Modeling naive psychology of characters in simple commonsense stories. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Steven Reiss. 2004. Multifaceted nature of intrinsic motivation: The theory of 16 basic desires. *Rev. Gen. Psychol.*, 8(3):179.

Zhou Ren, Xiaoyu Wang, Ning Zhang, Xutao Lv, and Li-Jia Li. 2017. Deep reinforcement learning-based

image captioning with embedding reward. *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 74–84, Atlanta, Georgia. Association for Computational Linguistics.

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Itsumi Saito, Kyosuke Nishida, Hisako Asano, and Junji Tomita. 2018. Commonsense knowledge base completion and generation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 141–150.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI*.

Tim Salimans and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909.

Maarten Sap, Ronan LeBras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A Smith, and Yejin Choi. 2019a. Atomic: An atlas of machine commonsense for if-then reasoning. In *AAAI*.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019b. Social iqa: Commonsense reasoning about social interactions. *ArXiv*, abs/1904.09728.

Roger C Schank and Robert P Abelson. 1975. *Scripts, plans, and knowledge*. Yale University.

Abigale See, Peter J. Liu, and Christopher Manning. 2017. Gettothepoint: Summarization with pointer-generatornetworks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Query-reduction networks for question answering. *arXiv preprint arXiv:1606.04582*.

Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. 2019. The woman worked as a babysitter: On biases in language generation. In *EMNLP/IJCNLP*.

Mohammad Shoeybi, Mostofa Ali Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *ArXiv*, abs/1909.08053.

Zhangzhang Si, Mingtao Pei, Benjamin Yao, and Song-Chun Zhu. 2011. Unsupervised learning of event and-or grammar and semantics from video. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 41–48. IEEE.

Stephen Soderland, Brendan Roof, Bo Qin, Shi Xu, Mausam, and Oren Etzioni. 2010. Adapting open information extraction to domain-specific relations. *AI Magazine*, 31:93–102.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge.

In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA. ACM.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Yi Chern Tan and L. Elisa Celis. 2019. Assessing social and intersectional biases in contextualized word representations. In *NeurIPS*.

Niket Tandon, Bhavana Dalvi, Joel Grus, Wen-tau Yih, Antoine Bosselut, and Peter Clark. 2018. Reasoning about actions and state changes by injecting commonsense knowledge. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 57–66, Brussels, Belgium. Association for Computational Linguistics.

Niket Tandon, Bhavana Dalvi Mishra, Keisuke Sakaguchi, Antoine Bosselut, and Peter Clark. 2019. Wiqa: A dataset for "what if..." reasoning over procedural text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Lisbon, Portugal. Association for Computational Linguistics.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *Proceedings of the Association for Computational Linguistics*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. CIDEr: Consensus-based image description evaluation. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the 2015 IEEE Conference on Computer Cision and Pattern Recognition*.

Niklas Wahlstrom, Thomas B. Schön, and Marc Peter Deisenroth. 2015. From pixels to torques: Policy learning with deep dynamical models. *CoRR*, abs/1502.02251.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia. Association for Computational Linguistics.

Dirk Weissenborn, Tom'avs Kovcisk'y, and Chris Dyer. 2017. Dynamic integration of background knowledge in neural nlu systems. *CoRR*, abs/1706.02596.

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. 2018. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6:287–302.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the International Conference for Learning Representations*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916*.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4).

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of The 32nd International Conference on Machine Learning*.

An Yang, Kai Liu, Jing Liu, Yajuan Lyu, and Sujian Li. 2018a. Adaptations of ROUGE and BLEU to better evaluate machine reading comprehension task. In *Proceedings of the Workshop on Machine Reading for Question Answering*, pages 98–104, Melbourne, Australia. Association for Computational Linguistics.

Bishan Yang, Wen tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019.

Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018b. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. 2016. Reference-aware language models. *CoRR*, abs/1611.01628.

Xingxing Zhang and Mirella Lapata. 2017. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Ryan Cotterell, Vicente Ordonez, and Kai-Wei Chang. 2019. Gender bias in contextualized word embeddings. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 629–634, Minneapolis, Minnesota. Association for Computational Linguistics.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. 2017. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. *ArXiv*, abs/1707.09457.

Victor Zhong, Caiming Xiong, Nitish Shirish Keskar, and Richard Socher. 2019. Coarse-grain fine-grain coattention network for multi-evidence question answering. In *Proceedings of the International Conference for Learning Representations*.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Appendix A

Experimental Settings for COMET

A.1 Training Details of Baseline Models

A.1.1 ConceptNet Baseline

We train the ConceptNet baseline with a learning rate of $\eta = 0.0001$ for 100k minibatches. Early stopping is used with the validation loss. Similarly to Saito et al. (2018), we use 200-dimensional hidden states and 200-dimensional word embeddings. We use a single-layer bidirectional LSTM (Hochreiter and Schmidhuber, 1997) to encode the seed phrase and a single-layer unidirectional LSTM to decode the target phrase. Relation embeddings are concatenated with the word embeddings of the decoder before being input to the decoder LSTM. We set the dropout rate to 0.2 before the output projection layer and after the word embedding layers. We outline the following differences between our re-implementation of the model of Saito et al. (2018) and their original implementation and the reason for the change.

1. We use Glove (Pennington et al., 2014) embeddings rather than fastText embeddings (Bojanowski et al., 2017) to initialize word embeddings. Because the model indicated that 200-dimensional word embeddings were used, we could not use the pretrained embeddings provided by the fastText group.[1] In Saito et al. (2018), the authors described training their fastText embeddings on Wikipedia. With no reference to the precise corpus used, we opted to use Glove embeddings to initialize the word

The material in this appendix is adapted from Bosselut et al. (2019)

[1] https://fasttext.cc/

embeddings of the encoder and decoder instead.

2. We use the Adam optimizer with learning rate of $\eta = 0.0001$, rather than SGD with a learning rate of $\eta = 1.0$. We originally trained a model using both approaches, but found that the Adam-trained model achieved a lower development set perplexity. We also do not use weight decay, as this seemed to lower development set performance too.

3. We do not train the generation model jointly with the completion model. We only train an individual generator. The results of Saito et al. (2018) did not show a significant difference in generation performance between the two on the ConceptNet dataset.

4. We train a second baseline (LSTM - s) that does not learn to produce relations in both directions (i.e., $sr \rightarrow o$ and $or \rightarrow s$). Instead, it only learns parameters that can produce relations in the forward direction ($sr \rightarrow o$)

5. We do not decay the learning rate because it was unclear from the original work what the exact learning rate schedule was.

A.2 Additional Seed Knowledge Graph Details

In Table A.1, we show the relations of the ATOMIC knowledge graph, their definitions, and example generations for each of them given the same seed event: "Person X puts Person X's trust in Person Y"

Relation	Description	Example Completion:
		Person X puts Person X's trust in Person Y
oEffect	The effect the event has on others besides Person X	is considered trustworthy is believed gains Person X's loyalty
oReact	The reaction of others besides Person X to the event	trusted honored trustworthy
oWant	What others besides Person X may want to do after the event	work with Person X partner with Person X to help Person X
xAttr	How Person X might be perceived given their part in the event	faithful hopeful trusting
xEffect	The effect that the event would have on Person X	gets relieved stays faithful Is betrayed
xIntent	The reason why Person X would cause the event	to be trusting his or her help/guidance/advice to be friends
xNeed	What Person X might need to do before the event	to be friends with Person Y to have heard a lot of good things about Person Y to get to know Person Y
xReact	The reaction that Person X would have to the event	trusting safe, not alone understood
xWant	What Person X may want to do after the event	to rely on Person Y to go into business with Person Y to make sure that their heart feeling is right

Table A.1: Definitions of the relations in ATOMIC. Events in ATOMIC center around the personal situations of a central figure, Person X, with potentially more participants.

A.3 Additional Experiments

In addition to the training setups for knowledge graph construction outlined in Chapter 3, we explore various multi-task and hierarchical learning setups on top of the taxonomy of commonsense relations given by Sap et al. (2019a), which group together along various axes (e.g., related to agent/theme, related to causes/effects, etc.).

Organization	Description	Relations
PERSON X/Y	The training set is split into relations for the subjects of the event (Person X) and relations for other participants in the event	T_1 = {xAttr, xEffect, xIntent, xNeed, xReact, xWant} T_2 = {oEffect, oReact, oWant}
PRE/POST	Event preconditions are jointly trained (i.e., intentions, needs). Event postconditions are jointly trained.	T_1 = {xIntent, xNeed} T_2 = {oEffect, oReact, oWant, xEffect, xReact, xWant}
(IN)VOLUN	Involuntary relations are trained jointly, such as reactions and effects. Voluntary relations are trained jointly, such as needs, wants, and intents.	T_1 = {oWant, xIntent, xNeed, xWant} T_2 = {oEffect, oReact, xAttr, xEffect, xReact}
FULL	The training set is made up of all relations and the model is trained jointly on all of them	T_1 = {oEffect, oReact, oWant, xAttr, xEffect, xIntent, xNeed, xReact, xWant}

Table A.2: Multi-relation training setups. Following Sap et al. (2019a), the xAttr relation is not included in the PRE/POST training configuration

A.3.1 Multi-relation Training

For the ATOMIC corpus, we experiment with multiple multi-task training setups, similar to Sap et al. (2019a). First, we train an individual model for each relation type (oReact, oEffect, etc.), which we denote as COMET - 9LM in the Table A.2. We also experiment with various information-sharing dataset configurations that organize different relations across common dimensions. We outline these dimensions and the makeup of each split in Table A.2. For ConceptNet, all models are always trained on all relation types jointly. Results on automatic evaluation metrics are provided in Table A.4. Because there did not seem to be significant differences between these performances and that of COMET - FULL, we did not run additional

experiments on these ablations.

Meta-Token	Description	Relations
<X>	Appended to relations that describe an attribute of Person X	xAttr, xEffect, xIntent, xNeed, xReact, xWant
<Y>	Appended to relations that describes an attribute of a participant that is not Person X	oEffect, oReact, oWant
<Pre>	Appended to relations that correspond to pre-conditions of the event	xIntent, xNeed
<Post>	Appended to relations that correspond to post-conditions of the event	oEffect, oReact, oWant, xEffect, xReact, xWant
<Voluntary>	Appended to relations that correspond to voluntary dimensions of the situation	oWant, xIntent, xNeed, xWant
<Involuntary>	Appended to relations that correspond to involuntary dimensions of the situation	oEffect, oReact, xAttr, xEffect, xReact

Table A.3: Category hierarchy meta-tokens, along with their descriptions and their corresponding relations

Model	PPL[3]	BLEU-2	N/T sro[4]	N/T o	N/U o
COMET- 9LM	11.72	14.89	100.00	9.45	49.89
COMET- (IN)VOLUN	11.38	14.99	100.00	8.60	48.36
COMET- PERSONX/Y	11.30	15.21	100.00	9.12	49.59
COMET- PRE/POST	11.35	14.88	100.00	9.86	51.86
COMET- FULL (- pretrain)	15.42	13.88	100.00	7.25	45.71
COMET- FULL	11.14	15.10	100.00	9.71	51.20
COMET- FULL (+ hierarchy meta-tokens)	**10.98**	**15.27**	100.00	**10.03**	**51.97**

Table A.4: Automatic evaluations of quality and novelty for generations of ATOMIC commonsense that are trained with the training set split along different relation types. The training splits are outlined in Table A.2.

A.3.2 Concept Hierarchy Training

Leveraging the prior knowledge that certain relation types in the ATOMIC knowledge graph are linked to each other, we explore providing these group identities as additional tokens in the relation. For example, when generating the completion of a xReact relation, the model would receive as input the following meta-tokens: <xReact>, <X>, <POST>, <Involuntary> – thereby providing common context with other relations that are part of the same groupings (e.g., generating a phrase for a xWant relation would receive the <X> and <POST> tokens as input, but not <Involuntary>). Depending on the relation for a particular training example (e.g., xReact), a set of meta-tokens are appended to the relation tokens, X^r,

Model	oEffect	oReact	oWant	xAttr	xEffect	xIntent	xNeed	xReact	xWant	Total
COMET	**29.02**	37.68	**44.48**	**57.48**	**55.50**	**68.32**	**64.24**	**76.18**	75.16	**56.45**
COMET (+ hierarchy meta-tokens)	28.46	**38.96**	43.64	51.90	50.84	63.00	63.98	66.20	**75.82**	53.64

Table A.5: Human score of generations of ATOMIC commonsense for the regular COMET model and the COMET + category meta tokens

that provide hierarchical relational information, allowing the model to share information across relation types. We provide a more in-depth description of the category hierarchy training combinations in Table A.3. Results on human evaluation metrics are provided in Table A.5. Because the model with the hierarchical meta-tokens performed worse than the regular COMET, we did not run additional experiments on these ablations.

Appendix B

Experimental Settings for Dynamically Constructing Knowledge Graphs

B.1 Rules for Pruning Generation Sets

We use the following rules to prune the set of commonsense inferences generated by COMET as it constructs a graph of commonsense knowledge:

1. Any generation that is "none" is pruned
2. Any generation that is identical to a previous generation from the same inputs, but has added punctuation is pruned (e.g., to go to the mall vs. to go to the mall.)
3. Any generation that has the phrase "PersonY" for the following relations is removed: `oEffect, oReact, oWant`. These generations are untrustworthy as they are often impossible to resolve with an actual person in the context
4. Any generation for the following relations that does not have a token that is a verb is removed: `xEffect, oEffect`
5. In multiple candidate settings (i.e., beam search, top-k sampling), if one of the candidates is "none," we prune all candidates with less likely scores

The material in this appendix is adapted from Bosselut et al. (2019)

6. For the STORYCOMMONSENSE dataset, we only generate inferences along the following ATOMIC relations: xReact, oReact, xEffect, oEffect, xIntent. The reason for pruning xWant, oWant, xNeed, xAttr inferences is that emotional reactions for these dimensions could be irrelevant to the context. For example, the emotional reaction to *getting into a car accident* is different from *needing to own a car* to do this. Emotional reactions to the kept relations are more likely to be fidelitous to the original context.

B.2 Evaluation Prefixes

To better model the conditional distribution on which the COMET model was trained, we append the following prefixes to COMET-generated inferences when using them in Equation (4.4) to compute factor nodes between them and answer nodes:

Relation	Prefix
xWant	PersonX wants
xReact	PersonX is
xNeed	PersonX needs
xIntent	PersonX wants
xAttr	PersonX is
xEffect	PersonX
oReact	PersonX is
oEffect	PersonX
oWant	PersonX wants

Table B.1: Prefixes appended to COMET-produced commonsense inferences for the evaluation step (Eq. 4.4)

Appendix C

Experimental Settings for Neural Process Networks

C.1 Training Details of Baseline Models

C.1.1 Cooking Recipe Understanding Baselines

Joint Gated Recurrent Unit The hidden state of the GRU is 100. We use a dropout with a rate of 0.3 before any non-recurrent fully connected layers. We use the Adam optimizer with a learning rate of $\eta = .001$ and decay by a factor of 0.1 if we see no improvement on validation loss over a single epoch. We stop training early if the development loss does not decrease for five epochs. The batch size is 64. We use encoders, one for the entity selector, and one for the state change predictors. Word embeddings are initialized with skipgram embeddings using a word2vec model trained on the training set. We use a vocabulary size of 7358 for words.

Recurrent Entity Networks (EntNet) Memory cells are tied to the entities in the document. For a recipe with 12 ingredients, 12 entity cells are initialized. All hyperparameters are the same as the in the bAbI task from Henaff et al. (2016). We initialize the learning rate at $\eta = 0.01$ and halve it every 25 epochs. Entity cells and word embeddings are 100 dimensional. The encoder is a multiplicative mask initialized the same

The material in this appendix is adapted from Bosselut et al. (2018a) and Rashkin et al. (2018)

as in (Henaff et al., 2016). Intermediate supervision from the weak labels is provided to help predict entities. A separate encoder is used for computing the attention over memory cells and the content to write to the memory. Dropout of 0.3 is used in the encoders. The batch size is 64. We use a vocabulary size of 7358 for words, and 2996 for entities.

C.1.2 Cooking Recipe Generation Baselines

Seq2seq The encoder and decoder are both single-layer GRUs with hidden size 200. We use dropout with probability 0.3 in the decoder. We train with the Adam optimizer starting with a learning rate of $\eta = 0.0003$ that is halved every 5 epochs. The encoder is bidirectional. The model is trained to minimize the negative loglikelihood of predicting the next word.

Attentive Seq2seq The encoder is the same as in the seq2seq baseline. A multiplicative attention between the decoder hidden state and the context vectors is used to compute the attention over the context at every decoder time step. The model is trained with the same learning rate, learning schedule and loss function as the seq2seq baseline.

EntNet Generator The model is trained in the same way as the NPN generator model in Section 5.6, except that the state representations used as input are produced by the EntNet baseline described in Section 5.5 and Appendix C.1.1.

C.1.3 Short Story Understanding Baselines

The following baselines are given a sentence t from a story X_t containing N words $\{x_0, x_1, \ldots, x_N\}$ from vocabulary V, a character in that story $e_i \in \mathcal{E}$ where $\mathcal{E}$ is the set of characters in the story, and (optionally) the preceding sentences in the story $C^t = \{X_0 \ldots, X_{t-1}\}$ containing words from vocabulary V. A representation for a character's psychological state after the events of sentence X_t is encoded as:

$$\mathbf{h}_t^e = \text{Encoder}(X_t, C_t[e_i]) \tag{C.1}$$

where $C_t[e_i]$ corresponds to the concatenated subset of sentences in C_t where e_i appears. Moving forward,

we drop t from subscripts for simplicity. Unless specified, $\mathbf{h}^e$ is computed by encoding separate vector representations for the sentence ($X \rightarrow \mathbf{h}^x$) and character-specific context ($C[e_i] \rightarrow \mathbf{h}^c$) and concatenating these encodings ($\mathbf{h}^e = [\mathbf{h}^c; \mathbf{h}^x]$).

All classification baselines are trained with the Adam Optimizer (Kingma and Ba, 2015) with a learning rate of $\eta = 0.001$ and gradient clipping if the norm of the gradients exceeds 1. We regularize with dropout layers (Srivastava et al., 2014) whose probabilities are specific to each model. All models are trained with word embeddings of dimensionality 100 that are initialized with pretrained Glove vectors (Pennington et al., 2014).

TF-IDF We learn a TD-IDF model on the full training corpus of Mostafazadeh et al. (2016) (excluding the stories in our development/test sets). To encode the sentence, we extract TF-IDF features for its words, yielding $\mathbf{v} \in \mathcal{R}^V$. A projection and non-linearity is applied to these features to yield $\mathbf{h}^x$:

$$\mathbf{h}^x = \phi(\mathbf{W}\mathbf{v} + \mathbf{b}) \tag{C.2}$$

where $\mathbf{W} \in \mathcal{R}^{d \times V}$. The character vector $\mathbf{h}^c$ is encoded in the same way on sentences in the context pertaining to the character.

GloVe We extract pretrained Glove vectors (Pennington et al., 2014) for each word in V. The word embeddings $\mathbf{x}_n$ for a word $x_n \in X$ are max-pooled, yielding embedding $\mathbf{v} \in \mathcal{R}^H$, where H is the dimensionality of the Glove vectors. Using this max-pooled representation, $\mathbf{h}^x$ and $\mathbf{h}^c$ are extracted in the same manner as for TF-IDF features (Equation C.2).

CNN We implement a CNN text categorization model using the same configuration as Kim (2014) to encode the sentence words. A sentence is represented as a matrix, $\mathbf{X} \in \mathcal{R}^{M \times d}$ where each row is a word embedding $\mathbf{x}_n$ for a word $x_n \in X$.

$$\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N] \tag{C.3}$$

$$\mathbf{h}^x = \text{CNN}(\mathbf{X}) \tag{C.4}$$

where CNN represents the categorization model from (Kim, 2014). The character vector $\mathbf{h}^c$ is encoded in the same way with a separate CNN. We use kernels of size 3, 4, and 5 and use 100 kernels of each size. We add a dropout layer of 0.5 between the word embedding layer and the convolutional kernel layers. We include a dropout layer of 0.5 before the logistic regression classifier.

LSTM A two-layer bi-LSTM (Hochreiter and Schmidhuber, 1997) encodes the sentence words and concatenates the final time step hidden states from both directions to yield $\mathbf{h}^x$. The character vector $\mathbf{h}^c$ is encoded the same way. The hidden states of the LSTM have dimensionality 100. We add dropout layers of 0.5 in between the word embedding layer and the LSTM and between LSTM layers. We include a dropout layer of 0.5 before the logistic regression classifier.

EntNet We use the "tied" recurrent entity network from Henaff et al. (2016). A memory cell is initialized for each of the I characters in the story, $\mathcal{E} = \{\mathbf{e}_0, \ldots, \mathbf{e}_I\}$. In an abuse of notation, we use $\mathcal{E}$ to signify both the list of characters in a story e_i and the set of memory vectors that represent them $\mathbf{e}_i$. The recurrent entity network reads documents one sentence at a time and updates each $\mathbf{e}_i \in \mathcal{E}$ after reading each sentence. Unlike the previous encoders, all sentences of the context C_t are given to the recurrent entity network along with the sentence X_t. The model learns to distribute encoded information to the correct memory cells. The representation passed to the downstream model is:

$$\mathbf{h}^e_t = \{\mathbf{e}_i\}_t \tag{C.5}$$

where $\{\mathbf{e}_i\}_t$ is the memory vector in the cell corresponding to $\mathbf{e}_i$ after reading X_t. All equations from the input encoder and dynamic memory are identical to those of Henaff et al. (2016). The input encoder computes a positionally weighted average of all the words in a sentence:

$$\mathbf{h}^x = \sum_n m_n \odot \mathbf{x}_n \tag{C.6}$$

where $\mathbf{x}_n$ is a word embedding at index n in a sentence, m_n is a positional weight for that index in the sentence, and $\mathbf{h}^x$ is a sentence representation (we note that $\mathbf{h}^x \rightarrow \mathbf{h}^x_t$ for sentence X_t). The dynamic memory is updated in the following way:

$$g_i = \sigma(\mathbf{e}_i^\intercal \mathbf{h}^x + \mathbf{e}_{i_0}^\intercal \mathbf{h}^x) \tag{C.7}$$

$$\tilde{\mathbf{e}}_i = \phi(\mathbf{U}\mathbf{e}_i + \mathbf{V}\mathbf{e}_{i_0} + \mathbf{W}\mathbf{h}^x) \tag{C.8}$$

$$\mathbf{e}_i \leftarrow \mathbf{e}_i + g_i \odot \tilde{\mathbf{e}}_i \tag{C.9}$$

$$\mathbf{e}_i \leftarrow \frac{\mathbf{e}_i}{||\mathbf{e}_i||} \tag{C.10}$$

where $\mathbf{e}_i$ is the value stored in memory cell i, $\mathbf{e}_{i_0}$ is a key corresponding to memory cell i, ϕ is a PReLU non-linearity, and $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$ are projection matrices (we deviate from the typical notation of defining projection matrices with $\mathbf{W}$ to match the notation in Henaff et al. 2016). We initialize entity memory keys and entity memory values with the sum of the Glove vectors for all the words in the character name. Entity keys $\mathbf{e}_{i_0}$ are locked during training. We use dropout of 0.3 between the encoder and dynamic memory.

C.2 Annotations

C.2.1 Annotating State Changes

We provide workers with a verb, its definition, an illustrative image of the action, and a set of sentences where the verb is mentioned. Workers are provided a checklist of the six state change types and instructed to identify which of them the verb causes. They are free to identify multiple changes. Seven workers annotate each verb and we assign a state change based on majority vote. Of the set of 384 verbs extracted, only 342 have a state change type identified with them. Of those, 74 entail multiple state change types.

C.2.2 Annotating End States

We give workers a verb, a state change type, and an example with the verb and ask them to provide an end state for the ingredient the verb is applied to in the example. We then use the answers to manually aggregate a set of end states for each state change type. These end states are used as labels when the model predicting state changes. For example, a LOCATION change might lead to an end state of "pan," "pot", or "oven." End states for each state change type are provided in Table C.1.

State Change Type	End States
TEMPERATURE	`hot; cold; room`
COMPOSITION	`composed; not composed`
CLEANLINESS	`clean; dirty; dry`
COOKEDNESS	`cooked; raw`
SHAPE	`molded; hit; deformed; separated`
LOCATION	`pan; pot; cupboard; screen; scale; garbage`, 260 more

Table C.1: End states for each state change type

C.2.3 Annotating Development and Test Sets

Annotators are instructed to note any entities that undergo one of the six state changes in each step, as well as to identify new combinations of ingredients that are created. For example, the sentence "Cut the tomatoes and add to the onions" would involve a SHAPE change for the tomatoes and a combination created from the "tomatoes" and "onions." In a separate task, three workers are asked to identify the actions performed in every sentence of the development and test set recipes. If an action receives a majority vote that it is performed, it is included in the annotations.