Artificial Neural Networks with Dynamic Connections

Dimitrios C. Gklezakos

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

Doctor of Philosophy

University of Washington

2022

Reading Committee:
Rajesh P. N. Rao, Chair
Bing W. Brunton
Sewoong Oh

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Artificial Neural Networks with Dynamic Connections

Dimitrios C. Gklezakos

Chair of the Supervisory Committee:
Rajesh P. N. Rao
Computer Science and Engineering

While highly successful in many different domains, most artificial neural networks suffer from a severe limitation ; they use the same parameters for different inputs. Different examples can have significantly different characteristics and can require different treatment from the model. This work investigates how to alleviate this issue using the recently introduced concept of "hypernetworks", neural networks that generate other neural networks. The first part of this thesis discusses how these models can be used to learn dynamic features from unlabeled image and video data. The second part introduces Active Predictive Coding Networks, models that hierarchically reconstruct images using neural sub-programs. The final part is dedicated to applications of hypernetworks to generating custom policies from contextual inputs in reinforcement learning settings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First I would like to express my deepest gratitude to my advisor Rajesh P. N. Rao for his trust, support and guidance throughout this journey. I feel truly privileged to have had the chance to work with him. Because of him, I am coming out of this program as a better researcher and a better person.

I would also like to thank the other members of my committee: Bing W. Brunton, Sewoong Oh and Amy Orsborn for their thought-provoking questions, time and involvement in this process.

I thank my collaborators Rishi Jha and Preston Jiang for their enthusiastic involvement in parts of this work. The discussions, intellectual stimulation and exchange of ideas between us are sincerely appreciated and will be missed.

My experience in this PhD program would not have been the same without my beloved friends and colleagues Raymond Cheng, Makrand Sinha, Brandon Holt and Daniel Butler.

Finally, I am thankful for the chance to work and interact with the members of the Neural Systems Lab: Koosha Khalvati, Zoe Steine-Hanson, Ares Fischer, Matt Bryan, Vishwas Sathish, Ellie Strandquist, Sam Sun, Sat Singh and Prashant Rangarajan.

My work has been supported by the following grants: NSF grant no. EEC-1028725, CRCNS/NIMH grant no. 1R01MH112166-01, Defense Advanced Research Projects Agency (DARPA) Contract No. HR001120C0021, National Science Foundation (NSF) Grant #EEC-1028725, a Weill Neurohub Investigator grant and a grant from the Templeton World Charity Foundation.

DEDICATION

To my parents Michalis Gklezakos and Vasiliki Gklezakou and my sister Triantafyllia Gklezakou. From them I have received love that is enough for more than a few lifetimes.

Chapter 1

INTRODUCTION

Artificial Neural Networks (ANNs), in their many different forms, hold state-of-the-art results in a vast range of artificial intelligence tasks. A severe limitation of most of these models is that their parameters are the same across inputs that can have very different characteristics. As an example, the features that are useful in distinguishing between different breeds of dogs will probably significantly differ from those required to classify different species of plants. At the same time models for different tasks can share some commonalities. It would make sense in this scenario to "spawn" a model that is customized to handle the specific (sub)task at hand. This approach resembles that of object-oriented programming in software engineering ; classes inherit similarities from their super-class while extending it in different ways.

In this thesis I will discuss a class of neural networks that generate parameters for other neural networks [27]. Apart from providing a framework for an object-oriented approach to ANNs, these "hypernetworks" have several other advantages over traditional models. Initially hypernetworks were introduced as a relaxed form of weight sharing that can compress feedforward ANNs but also enhance recurrent neural networks (RNNs) [27]. They have been used to learn functional representations of data for various data modalities such as images [39, 14], shapes [48, 7] or point clouds [63]. Hypernetworks have also been used in continual learning for supervised tasks [71] and model-based [33, 72] or model-free reinforcement learning scenarios [16].

In this chapter I will provide some technical background on how static and dynamic hypernetworks work. I will also present some recent theoretical results that highlight the advantage of hypernetworks in conditioning neural networks to contextual inputs. Finally I

will go over the organizational structure of the rest of this thesis.

1.1 Hypernetworks

The term "hypernetwork" was initially introduced in [27]. The authors define hypernetworks as small neural networks that generate weights for a larger neural network called the "primary". The primary network is then used to solve a task by mapping the input to the desired output. The initial motivation behind hypernetworks was to introduce a form of relaxed weight sharing. Recurrent neural networks (RNNs) can be viewed as deep feed-forward ANNs that use the same synaptic weights for all layers. Modern convolutional neural networks (CNNs) can be very deep, with some utilizing hundreds or even thousands of layers [28]. Hypernetworks can be viewed as a trade-off between these two ends of the weight-sharing spectrum. Smaller hypernetworks require few parameters but are not very expressive in terms of the generated weights, whereas larger ones can generate diverse sets of weights but require more parameters. Subsequent research into hypernetworks has revealed other significant advantages of the approach.

1.1.1 Static Hypernetworks

For simplicity let's assume the primary network f consists of D fully connected layers each with its own synaptic weight matrix W and bias b. Denote these parameters collectively by $\theta = \{W^{(j)}, b^{(j)}\}_{j=1}^{D}$. Then $\theta = H(z)$ is generated by feeding an embedding z through a neural (hyper)network H. These parameters are plugged into the primary network, which is then evaluated on the task. The output of the primary network on input x is $y = f(x; \theta)$ where $f(; \theta)$ denotes a neural network parametrized by θ. The hypernetwork is trained end-to-end via back-propagation, with gradients flowing through the loss function and the generated parameters. The set of trainable parameters consists of the parameters of H together with the embedding z. For a diagram of this basic hypernetwork format see Figure 1.1.

This architecture allows for a quite flexible form of weight sharing. For example instead of having a single z generate all the parameters, there could be an embedding z_j for every

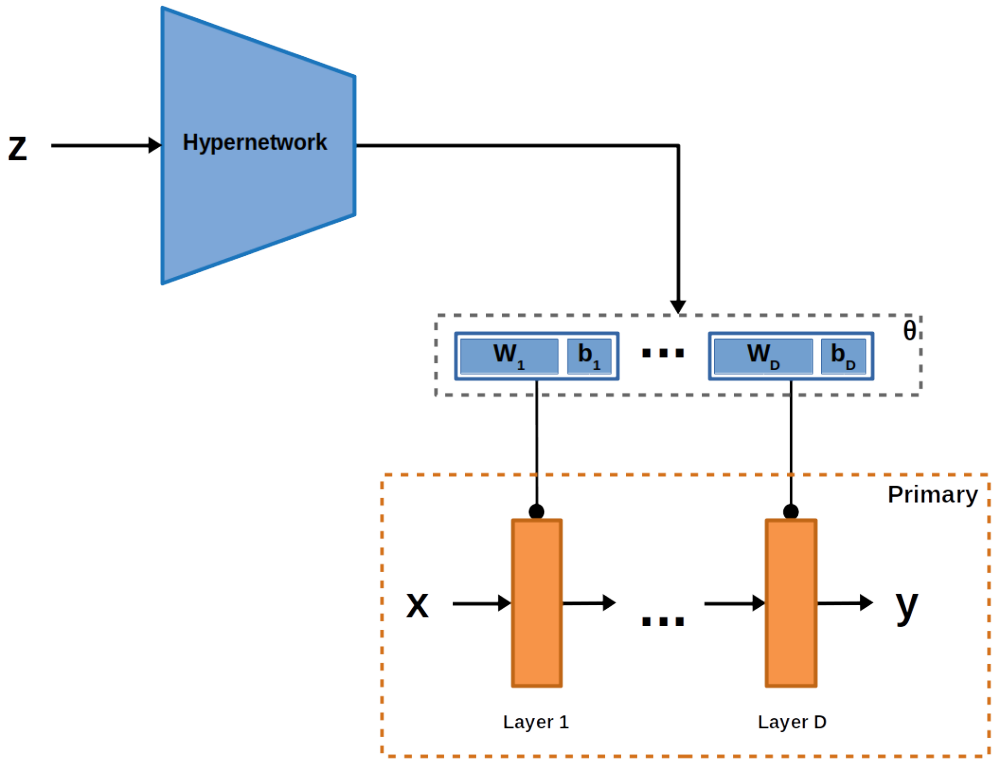


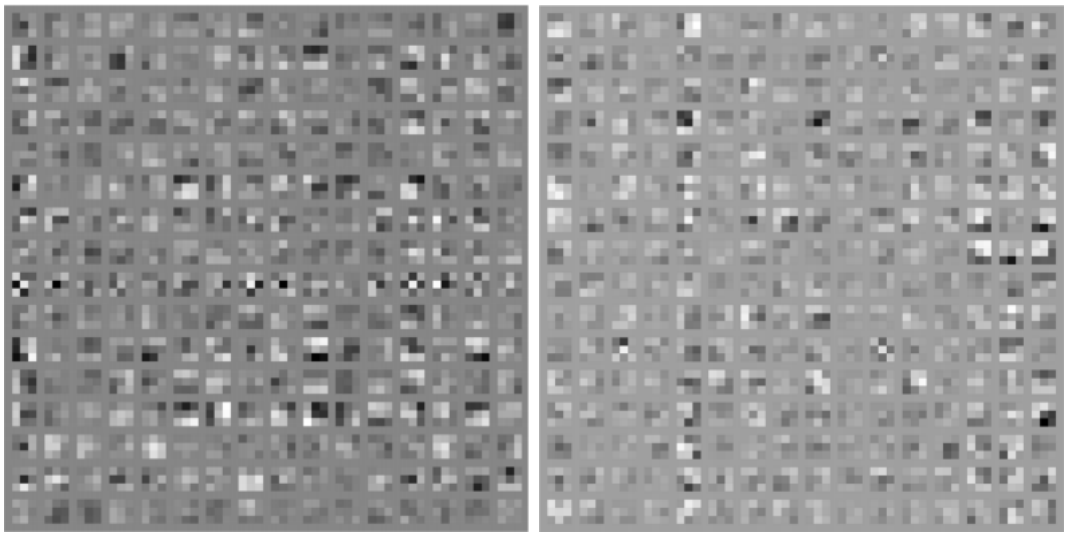
Figure 1.1: **Hypernetwork Architecture:** A learned embedding vector z together with the hypernetwork, generate the primary network.

layer of the primary. Layers with different sizes can be handled via "chunking" [27, 71] where each layer is generated by splicing together chunks of fixed size, with each chunk generated via a different embedding vector. The embedding z can also directly depend on the input x or part of it [39, 72].

The authors in [27] applied a hypernetwork to generate convolutional kernels for a convolutional neural network (CNN) that classifies instances from the CIFAR-10 dataset [42]. The primary is formulated as a Wide Residual Network (WRN) [28, 74]. The hypernetwork consists of just two linear layers and chunking is used.

For the architecture of the primary, WRN-40-2 from [74] was chosen. WRN-40-2 achieved 94.34% accuracy on CIFAR-10 with $2.236M$ parameters. The hypernetwork version (Hyper-

WRN-40-2) achieves 92.77% accuracy with only $148K$ parameters. Weight sharing via hypernetwork costs a 1.57% decrease in performance while reducing the number of parameters by at least an order of magnitude. Figure 1.2 shows a comparison of learned (WRN-40-2) vs generated (Hyper-WRN-40-2) kernels .



(a) WRN-40-2 (b) Hyper-WRN-40-2

Figure 1.2: 3×3 **WRN kernels:** (a) original model (b) generated via hypernetwork. Figure taken from [27].

1.1.2 Dynamic Hypernetworks

Static hypernetworks can be viewed as a form of model compression that reduces the number of total parameters for deep CNNs. Weight-sharing comes with a price ; a small drop in accuracy. Recurrent Neural Networks (RNNs) can be regarded as deep feedforward networks where the same layer is used over and over again. Since RNNs share the same weights at every time step it is natural to wonder whether we could *increase* their accuracy if we relaxed this strict form of weight-sharing. The basic RNN version is given by:

$$h_t = \phi\left(W_h h_{t-1} + W_x x_t + b\right) \tag{1.1}$$

Where h_t is the recurrent hidden state and x_t the input at time t, while $\{W_h, W_x, b\}$ are the learnable parameters. Suppose we could generate these parameters as before via another network. Then the dynamic parameters would be a function of some embedding. Instead of learning fixed embeddings we can generate them through another RNN. The HyperRNN [27] maintains its own hidden state $\hat{h}_t$ and generates dynamic parameters for the primary denoted by $\{W_h(\hat{h}_t), W_x(\hat{h}_t), b(\hat{h}_t)\}$. The input to the HyperRNN is $\hat{x}_t = [h_{t-1}, x_t]$ [1]. The hidden state $\hat{h}_t$ is updated via an equation similar to Equation 1.1.

In [27], the authors apply this concept to LSTMs [32] to derive a HyperLSTM version. Perhaps the most interesting application of the HyperLSTM is that of a generative model for handwriting. The authors adjust the handwriting prediction model introduced in [25]. The model represents the $\Delta x, \Delta y$ coordinates of the pen stroke as a Gaussian Mixture Model (GMM). The parameters for the GMM are generated via the LSTM [5]. The authors convert this model to use a HyperLSTM, improving the log-likelihood score from -1055 to -1162 nats.

1.1.3 Modularity of Hypernetworks

Perhaps the most intriguing quality of this framework is that it provides a novel way of "conditioning" the primary f to contextual information z, that is possibly outside of the domain of the input x. Suppose we have two inputs x and I and we are interested in approximating a function $y(x, I)$. We can view x as the standard input of the ANN, whereas I as a contextual input that y is conditioned on and can potentially be similar across many different instances of x. In [48] I is a collection of 2D images of a 3D object, x a 3D point and $y(x, I)$ indicates whether this point resides within the boundaries of the object. I will discuss another example in Chapter 4 where I is a map of a space and y a network that navigates an agent to a specific goal location within that space.

The traditional approach to the above scenario is the embedding one ; the conditioning input is passed through an embedding ANN e to produce the embedding $e(I)$. Then it is

[1] The input to the primary concatenated with the hidden state of the primary.

concatenated with x and passed through a primary network q to compute $q([x, e(I)])$. The second approach is the hypernetwork one. Input I is passed through a hypernetwork f to generate the parameters θ_I for a primary network g to compute $g_{\theta_I}(x)$.

In [18] the authors introduce the concept of "modularity". Intuitively we could fit a different primary network g_I for every different I. Each such primary would require a certain number of parameters to approximate the target $y_I = y(., I)$ to a given error ϵ in the worst case scenario. We say that an approach is modular if the size of its primary whose weights are given by either q (embedding) or $\theta_I = f(I)$ (hypernetwork) matches that number of parameters. This means that the model efficiently conditions q or g on I to compute y_I.

The authors provide the first theoretical foundation for the hypernetwork approach by proving its modularity property. They compare it to the embedding approach and show that the hypernetwork requires orders of magnitude fewer parameters to approximate the target function y to the same quality. The result suggests that the hypernetwork approach will likely outperform the embedding one whenever a model needs to be conditioned on contextual information with this structure.

1.1.4 Related Work

There are various other techniques for dynamically adjusting ANNs and RNNs based on the input. The most relevant example would be that of attention. Attention allows a model to only focus on the part of the input that is most relevant. In [1] the authors use 'fast weights" that depend on the input to attend to the recent past in the context of a recurrent neural network. In [34] Spatial Transformer Networks apply affine transformations to normalize the input via cropping, centering, scaling etc. In [15] the same module is used to sequentially attend to the various relevant parts of an image. Perhaps the most well-known attention approach is that of Transformers [70] which have been very successful in natural language processing, with recent applications to computer vision [12]. Transformers allow models for sequential data to draw context from any position in the sequence.

A different technique is that of dynamic routing for capsules [58]. Dynamic routing

allows the output of a capsule [2] from one layer to be routed only to the capsule in the next layer that is the most appropriate to process it. The model uses some form of iterative optimization such as Expectation-Maximization [31] to determine the routing parameters. A different approach is that of [8], where the authors propose a composite convolutional layer that maintains a set of kernels and chooses which one to apply based on the current input. They demonstrate that the technique allows for an increase in model complexity without significant increase in the depth or width of the architecture. While it is not explicitly stated in the paper, this approach falls into the hypernetwork category. Finally hypernetworks share some similarities with Fast Weight Programmers introduced in [61, 60].

1.2 Organization

The rest of this thesis is organized as follows:

- **Chapter 2:** In this chapter I will present two hypernetwork-based approaches for enhancing sparse coding models. In the first part I will introduce the Transformational Bilinear Sparse Coding (TBSC) framework, a hypernetwork that learns dynamic yet interpretable features from short unlabeled video sequences. The chapter follows with a discussion on Dynamic Predictive Coding, an approach that uses hypernetworks to better predict sparse codes of future video frames.

- **Chapter 3:** In this chapter I will introduce Active Predictive Coding Networks (APCNs). APCNs are models that parse images hierarchically via structured sequences of glimpses, using neural sub-programs generated by hypernetworks. APCNs combine predictive coding, hypernetworks and reinforcement learning.

- **Chapter 4:** In this chapter I will introduce Hyper-Universal Policy Approximators (HUPAs). HUPAs are models that efficiently condition agent policies to contextual information.

[2] A capsule is a complex type of artificial neuron that "encapsulates" not only a feature activation but also its pose (location, orientation, scale etc.).

Chapter 2

LEARNING FROM UNLABELED VIDEOS USING DYNAMIC SPARSE CONNECTIONS

In this section I will discuss two novel extensions of sparse coding ; a model that can learn interpretable features from unlabeled data. In Section 2.1 I will go over the background for sparse coding. In Section 2.2 I will introduce Transformational Bilinear Sparse Coding (TSBC) [21], a sparse coding model with dynamic connections that allows for the discovery of steerable, yet interpretable features from small unlabeled video segments. In Section 2.3 I will discuss how dynamic connections can be applied to the temporal domain to enhance sparse coding models for video prediction.

2.1 Sparse Coding

Sparse coding was initially introduced in the context of modeling image patches [53]. Each such patch is reconstructed via a linear combination of a collection of features called the dictionary:

$$I_i = \sum_k r_{ik} U_k + n \tag{2.1}$$

where I_i is an image patch, n a Gaussian noise vector, $\{U_k\}_{k=1}^K$ a dictionary of features that are common across all inputs and $\{r_{ik}\}_{k=1}^K$ a set of activation coefficients, specific to I_i, that linearly combine these features. One important detail is that the dictionary is over-complete ; the number of atoms in the dictionary is typically larger than the dimension of I_i. Over-completeness would result in an under-constrained model and is counter-balanced by the requirement that r is sparse i.e. for any given input only a small number of features are active. Typically this constraint is expressed by penalizing the ℓ_1 norm of r.

The model is trained using the mean-squared-error (MSE) loss:

$$L(r, U) = \frac{1}{N} \sum_{i=1}^{N} \left\| I_i - \sum_k r_{ik} U_k \right\|_2^2 + \lambda \sum_{i=1}^{N} \|r_i\|_1 \tag{2.2}$$

The most common algorithms for optimizing this model are the Iterative Soft-Thresholding Algorithm (ISTA) and its version with momentum (FISTA) [3]. When applied to natural images, sparse coding learns features that resemble Gabor filters, similar to those in the V1 area of the mammalian cortex (Figure 2.1). Sparse coding has been extended to various convolutional versions to handle larger images [29].

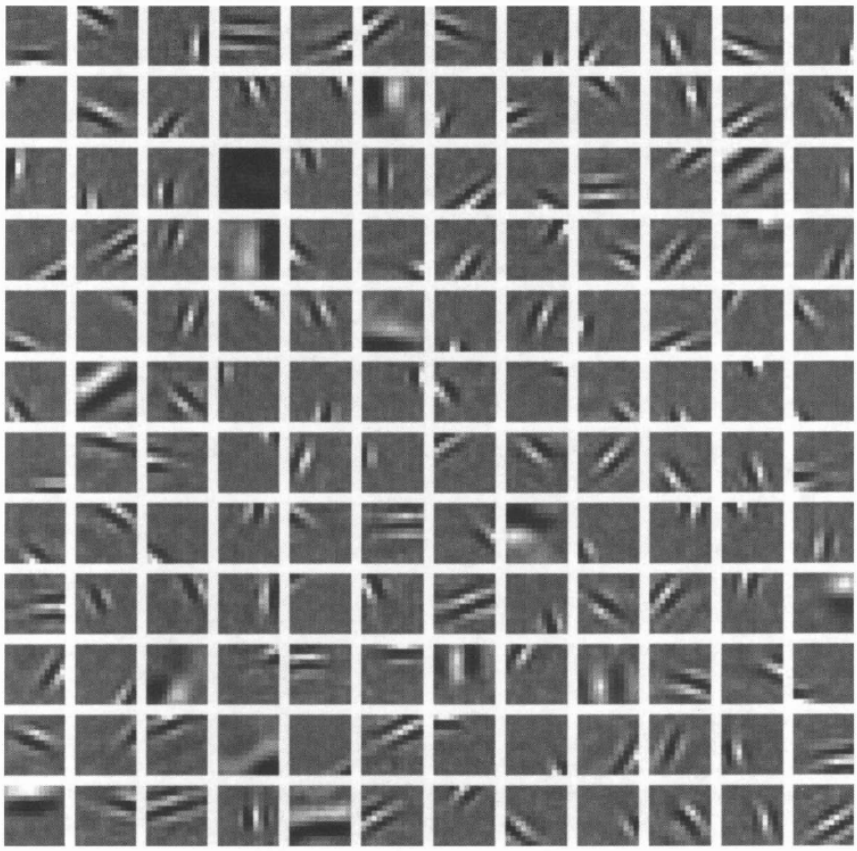


Figure 2.1: **Sparse Coding dictionary learned from natural image patches.** Figure taken from [53]

In [26, 6] the authors consider a bilinear version of sparse coding that incorporates style and content parameters. The dictionary features are separated in K groups of M features each. The style parameters x are common across all groups and linearly combine the features in each group to produce K dynamic ones: $U_k(x) = \sum_m x_m U_{km}$. These steerable features are then combined by using the content coefficients r to reconstruct the image. The bilinear model for image patches is given by:

$$I = \sum_k r_k \sum_m x_m U_{km} + n = \sum_k r_k U_k(x) + n \tag{2.3}$$

where n is again a Gaussian noise vector. The advantage of bilinear sparse coding is that it decomposes the input into content r and style x, effectively grouping together similar features. The bilinear model is actually a hypernetwork. The primary network corresponds to a linear combination of K features $U_k(x)$ that are dynamically generated via the same style vector x.

Typically bilinear sparse coding models are trained using pairs of video frames I_{t+1} and I_t, with r fixed and x inferred separately to account for the difference between frames:

$$\Delta I = I_{t+1} - I_t \simeq \sum_k r_k \sum_m \left(x_{t+1,m} - x_{t,m}\right) U_{km} = \sum_k r_k \sum_m \Delta x_{t,m} U_{km} \tag{2.4}$$

In [21] we identified a strong connection between bilinear models and the Lie group approach to vision [10, 55, 51] where two consecutive frames are modelled as $I_{t+1} = T(\Delta x) I_t$ where T is a transformation operator. The first-order Taylor series approximation of the Lie model [54, 51] is given by:

$$\Delta I = I_{t+1} - I_t = \sum_m \Delta x_{t,m} \nabla x_m I_t \tag{2.5}$$

Suppose $I_t \simeq \sum_k r_k B_k$ where the features $B_k \in \mathbb{R}^{d \times 1}$ (d is the dimension of I_t) form an underlying dictionary. Replacing ∇x_m with the transformation matrix $G_m \in \mathbb{R}^{d \times d}$, we obtain: $\Delta I \simeq \sum_m \Delta x_{t,m} G_m \sum_k r_k B_k = \sum_k r_k \sum_m \Delta x_{t,m} G_m B_k$. Comparing with Equation 2.4 above, we see that $U_{km} = G_m B_k$. Therefore the features within each group can be interpreted as transformed versions of each other.

2.2 Transformational Bilinear Sparse Coding

During early development, the brain learns a general-purpose internal representation of objects from unlabeled image sequences. This representation is compositional and leverages the decomposition of objects into parts, sub-parts, and features, along with their relative transformations. In contrast, modern object recognition systems based on deep learning require thousands of labeled examples and typically discard information about transformations (via pooling) in order to achieve invariance. Information about transformations is critical for tasks such as movement planning and spatial reasoning.

Most current unsupervised models produce representations that either lack interpretability or hierarchical depth. Variational autoencoders [38, 69] and generative adversarial networks (GANs) [23] typically produce non-interpretable features that do not match the object/parts hierarchy inherent in natural visual scenes. As discussed in Section 2.1, sparse coding and its variants can learn interpretable features from unlabeled images: these features resemble the localized oriented (Gabor) receptive fields found in the primary visual cortex. Ideally we would like to learn representations that are both hierarchical and interpretable.

By potentially stacking multiple sparse coding layers we could, in principle, learn interpretable feature hierarchies such as objects and their parts [77]. However, the number of features in each layer blows up quickly and the approach succumbs to the over-completeness requirement and to combinatorial explosion. A close inspection of these features (see Figure 2.1) shows that a lot of them are transformed versions of each other ; they can be obtained by translating, rotating or scaling another element in the dictionary. This suggests that sparse coding dictionaries essentially "tile" the space of all different versions of a feature [9]. Motivated by this observation our approach to mitigating the combinatorial explosion issue is to maintain a small dictionary of features that can be dynamically transformed to capture the structure of the input.

2.2.1 Related Work

A potential candidate sparse coding variant for building interpretable feature hierarchies from unlabeled data is its bilinear version discussed in 2.1. To train the model the authors in [26] extract sets of similar patches. The patches in each set are obtained by a window that is randomly translated around the same location in the source image. For each such set the content parameters r are assumed to be constant, while the style parameters vary according to the translation distance. "Clamping" the content in this manner makes the model not fully unsupervised. Note also that since the style parameters are the same across all groups, all dynamic filters share the same pose. This property might hold for patches that are translated versions of each other but does not necessarily hold for other settings where features can vary independently (i.e. video sequences). The diversity of features within each group will reflect the manual procedure via which each set of patches was extracted ; in this case each group mostly contains translated versions of the same feature.

The benefit of the decomposition of style and content in building hierarchies of features is also not obvious. Propagating both content and style parameters makes the model fall prey to the same issues with original sparse coding. A potential counter-measure to the combinatorial explosion is to propagate only the content vector r to the next layer and to only use x during reconstruction. In [78] the authors reduce the dimensionality of the inferred sparse code by pooling the sparse coefficients both spatially and across different features. They store the feature map locations that produced the maximum propagated value and use these "switches" in the backward reconstruction pass. While the model manages to learn at least two levels of interpretable features, the pooling operator discards information since it propagates only the maximum value. The switches, which are left behind in each layer and not discarded, contain information about the specific pose of each feature that might be relevant to the higher layers. The model in [75] attempts to mitigate this issue by introducing a differentiable pooling operator. The operator uses a Gaussian kernel to summarize the activations within the pooling area of a feature map. The parameters of

the kernel are propagated to the higher layer. This pooling operator is not applied across different feature maps, resulting in activations that are only pooled spatially. Finally both of these approaches only apply to static images, lacking any temporal element. The temporal continuity of natural videos offers rich information about transformations of various object parts, as well as object-part relationships.

In [9] the authors introduce the "Sparse Manifold Transform" that pools the sparse activations using a learnable pooling operator. The operator is trained using short sequences of frames and contracts the over-complete sparse code into a smaller dense one using a linearizing objective [24]. While the approach succeeds in learning more complex features it does not explicitly model entities and their transformations. Finally reversing the pooling operator requires solving another sparse inference problem.

2.2.2 Model

In this section I will introduce Transformational Bilinear Sparse Coding (TBSC). TBSC combines the bilinear model for sparse coding, with hypernetworks to learn interpretable, steerable features from unlabeled video data. TBSC builds on the bilinear model in various meaningful ways [21]: (a) Features have their own pose vectors x_k so that they can transform independently from frame to frame, (b) it goes beyond patch-based models and models large images, (c) the temporal continuity of videos is utilized to effectively group similar features together making the approach completely unsupervised and (d) pooling-via-hypernetwork is used to keep the size of the code tractable. The model consists of the following parts:

- K feature groups, each with a "base" feature U_k and M transformed features $\{U_{km}\}_{m=1}^{M}$.
- A set of sparse content coefficients r_k, one for each group and a set of sparse style vectors x_k of length M.
- K feature specific "mixing" matrices $E_k \in \mathbb{R}^{L \times M}$. Each such matrix converts the style vector x_k of a feature group (of size M) into a smaller pose embedding z_k of size L.

- L transformation matrices $G_l \in \mathbb{R}^{d \times d}$ where d is the number of pixels of a feature.

Of the above r and x are estimated during inference, while the rest constitute learnable parameters. I will first describe the model for image patch reconstruction. The extension to large images is straightforward and utilizes transposed convolutions [77, 76].

Suppose we have a current estimate of r and x. Each x_k is passed through the corresponding mixing matrix to obtain the pose embedding $z_k = E_k x_k$. The pose then linearly combines the transformation matrices to obtain a transformation operator $T(., z_k) = \sum_{l=1}^{L} z_{kl} G_l$. This operator transforms the base feature of the group, via z_k, to $U_k(z_k) = T(U_k, z_k)$. This is done by flattening the feature and multiplying it with the operator. This dynamic feature is then multiplied with the sparse coefficient r_k to obtain the total contribution of the group to the reconstruction $r_k T(U_k, z_k)$. The process via which each group contribution is estimated is illustrated in Figure 2.2.

Denote the M rows of a mixing matrix E_k by e_{km}. Each such row converts a style coefficient into a contribution to the pose embedding vector denoted by z_{km}. The embedding vector is given by the sum of all these contributions: $z_k = \sum_{m=1}^{M} z_{km} = \sum_{m=1}^{M} e_{km} x_{km}$. This contribution z_{km} can be propagated through the hypernetwork to generate the filter corresponding to that specific style parameter:

$$U_{km} = \sum_{l=1}^{L} z_{kml} G_l U_k \tag{2.6}$$

where U_{km} directly corresponds to a dictionary atom in 2.3. This property is a direct consequence of all the operations in the hypernetwork being linear and can be used to derive an expanded dictionary of transformed versions $\{U_{km}\}_{m=1}^{M}$ of the same base feature U_k.

The introduction of the pose embedding bottleneck z follows the intuition that the filters learned by sparse coding are mostly transformations of each other. It allows the size of the representation to stay manageable while still being expressive enough to learn dynamic features. The output of a TBSC layer consists of the concatenated content and pose vectors $\{(r_k, z_k)\}_{k=1}^{K}$. The resulting dimension can be significantly smaller than that of traditional

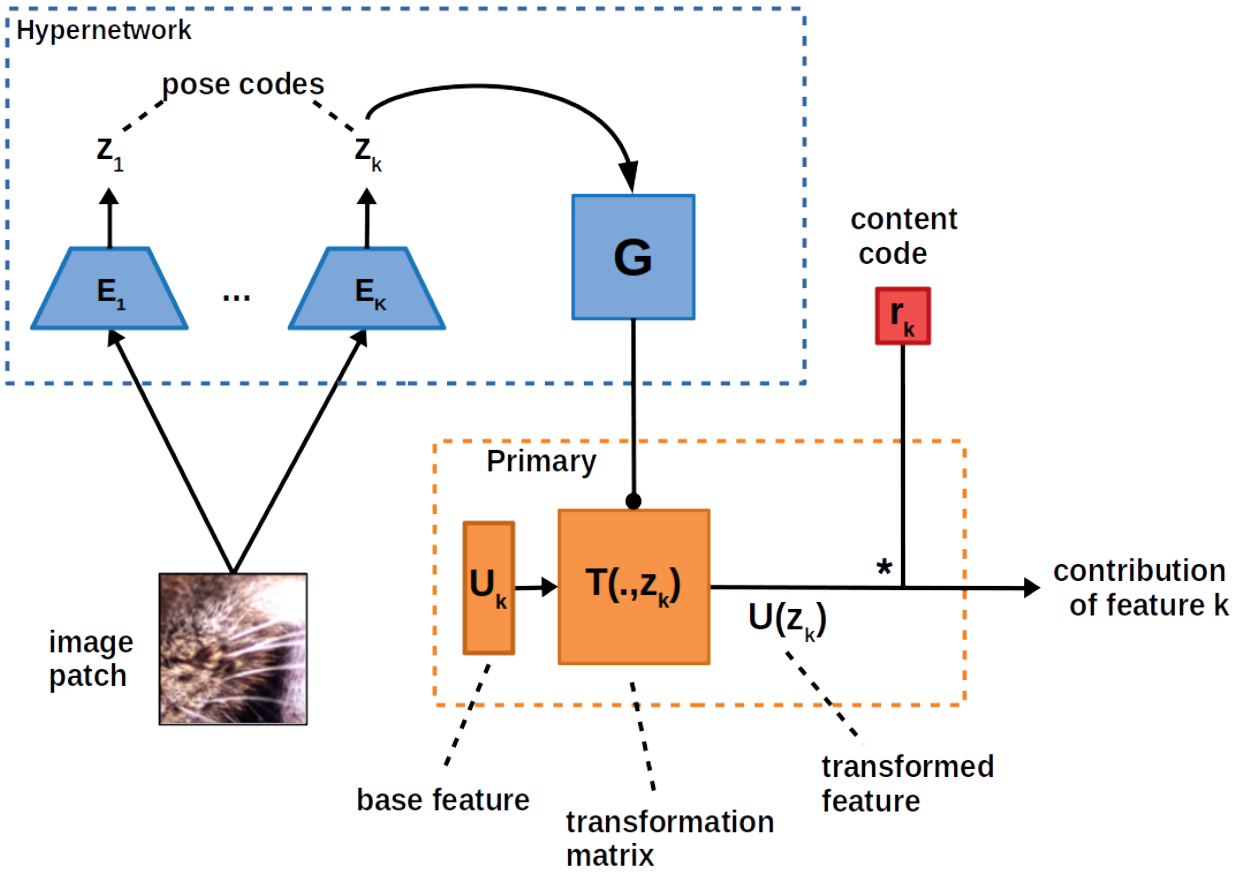


Figure 2.2: **Architecture diagram for Transformational Bilinear Sparse Coding (TBSC).**

bilinear sparse coding.

The TBSC reconstruction of the patch via the estimated r and x is given by:

$$\hat{I} = \sum_{k=1}^{K} r_k \sum_{m=1}^{M} \sum_{l=1}^{L} (e_{km} x_{km})_l G_l U_k \tag{2.7}$$

where $z_{km} = e_{km} x_{km}$.

2.2.3 Inference

To get around the need for manual "clamping" (as mentioned in Section 2.2.1) we utilized short sequences of T video frames. A reasonable assumption is that in such short sequences the content of a patch tends to stay invariant, while the style changes to account for the variation in the input. Therefore the content parameters r_k are constrained to be the same for all T frames in the sequence. The reconstruction-based loss function for T consecutive frames of a video is given by:

$$L(r, x_t) = \frac{1}{T} \sum_t \left\| I_t - \sum_{k=1}^{K} r_k \sum_{m=1}^{M} \sum_{l=1}^{L} (e_{km} x_{tkm})_l G_l U_k \right\|_2^2 \tag{2.8}$$

where r is constrained to be sparse and non-negative. Both the base features U_k as well as the derivative features $\{G_l U_k\}_{l,k}$ are projected to unit euclidean norm to properly constraint the problem. To capture the intuition that throughout the video sequences the content stays invariant, apart from using the same r vector, the style vector is projected to the probability simplex , enforcing that for each t and k, $\sum_{m=1}^{M} x_{tkm} = 1$ and that x is non-negative. This projection is similar to a sparsity and a non-negativity constraint. For the simplex projection we used the sorting-based algorithm in [13].

Jointly optimizing r and x can be costly and unstable. In practice we estimate x by setting $x_{tkm} =< U_{km}^T, I_t >$ followed by a projection to the simplex. The collection of features $\{U_{km}\}_{m=1}^{M}$ for each group k therefore functions as a single-layer encoder that infers the style vector based on the input. Alternatively these features together with the mixing matrix for the group can be regarded as a two-layer encoder that infers the dense embedding vector z_k (see Figure 2.2). This choice makes the style part of the model function in a manner that resembles that of winner take-all autoencoders [50]. To optimize r we use the fast iterative thresholding algorithm (FISTA) [3].

2.2.4 Results

For our experiments we used sequences of various animals moving in slow motion taken from YouTube videos. We derived $>$ 12K sequences of 10 consecutive frames each, with r assumed to be constant for each sequence during training. The sequences were converted to gray scale and scaled down to 96×176 pixels per frame. The frames were normalized using subtractive normalization[1]. To handle the size of the frame we use transposed convolutions with features of size 16×16 and a stride of 8. We used $K = 16$ groups with $M = 32$ style dimensions and $L = 8$ transformation dimensions, resulting in a 2×overcomplete dictionary of 512 features.

Our model learns localized oriented Gabor-like features similar to those seen in sparse coding. Figure 2.3 shows the learned expanded dictionary of features U_{km} ; each column shows $U_{k:}$ corresponding to different transformed versions of the same underlying feature. Note that not only translations but other transformations are learned as well, e.g., rotations and warping. The learned features allow for accurate reconstruction, as seen for example inputs in Figure 2.4. The dimension of the representation for each group is $L + 1 = 9$. The total dimension of the representation for each 16×16 pixel area is $K(L + 1) = 144$ which is significantly smaller than the dimension of the input $d = 16^2 = 256$.

[1] A Gaussian kernel is used to estimate the mean intensity around each pixel, which is then subtracted from the pixel value.

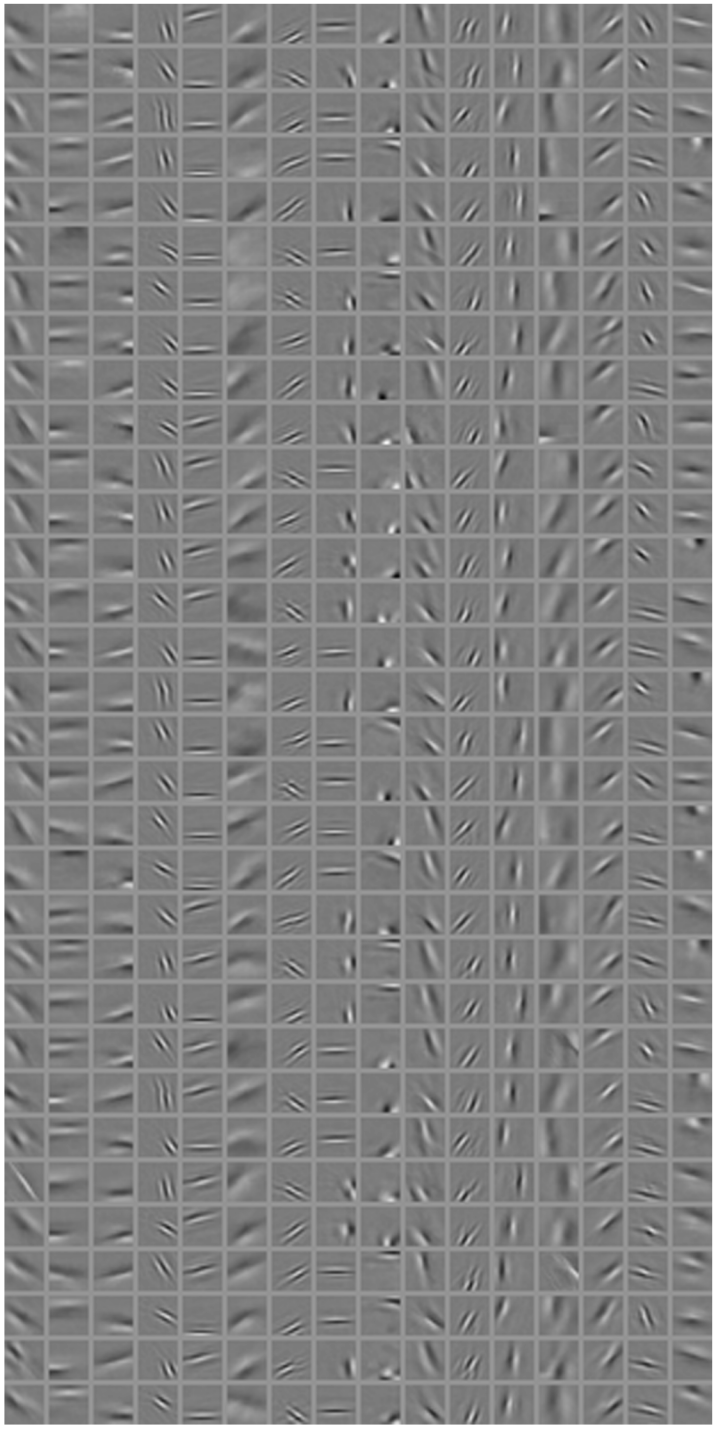


Figure 2.3: **Transformational Bilinear Sparse Coding for Natural Videos**. 16×16 pixel features U_{km}: each column shows transformed versions of a feature U_k for different m's.

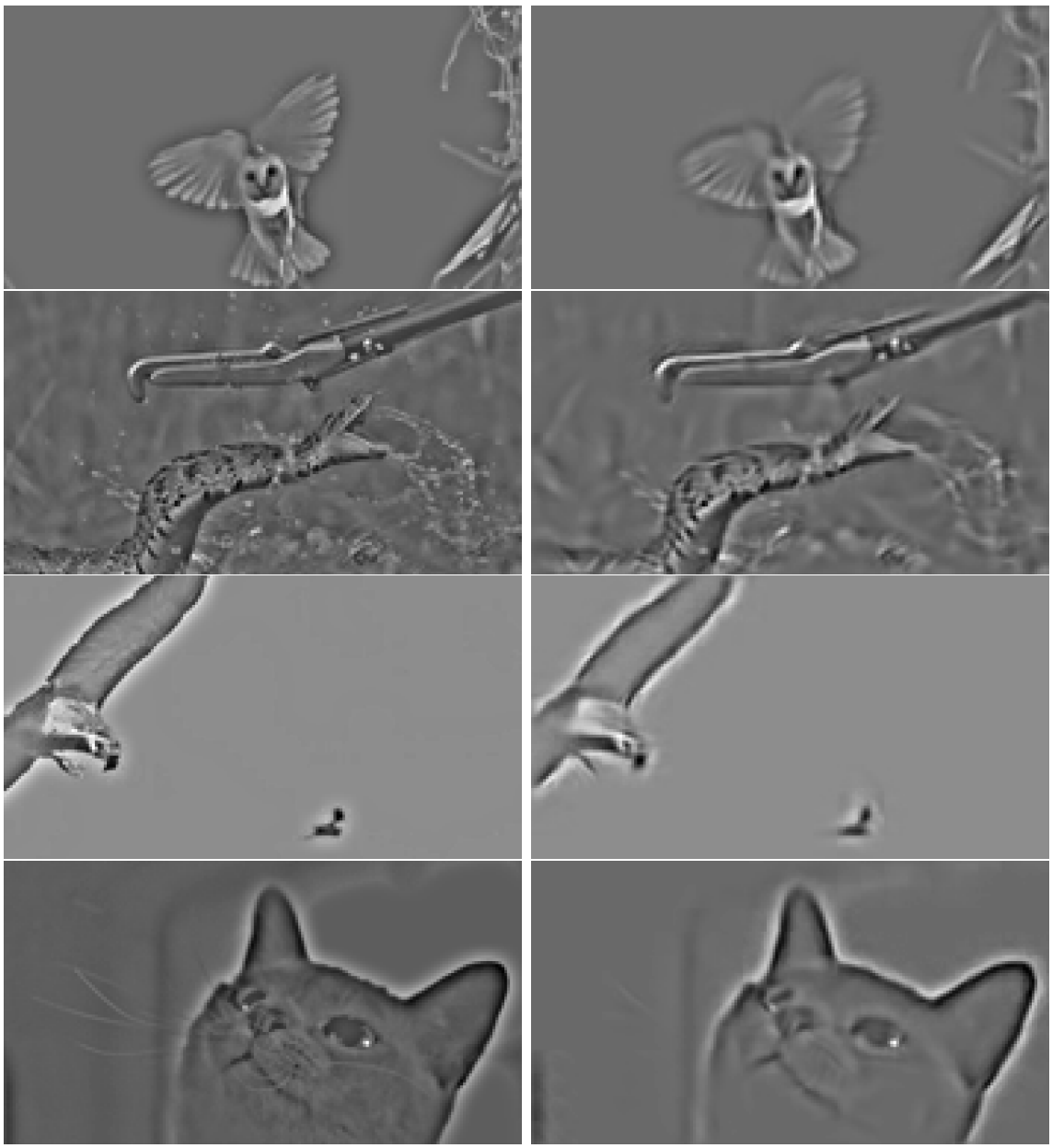


Figure 2.4: **Example TBSC Frame Reconstructions: (left column)** Original frames and **(right column)** their reconstruction using the dynamically generated features.

Figure 2.5 shows the effect of each of the L transformation dimensions in z on the base feature for each group, by visualizing $G_l U_k$.

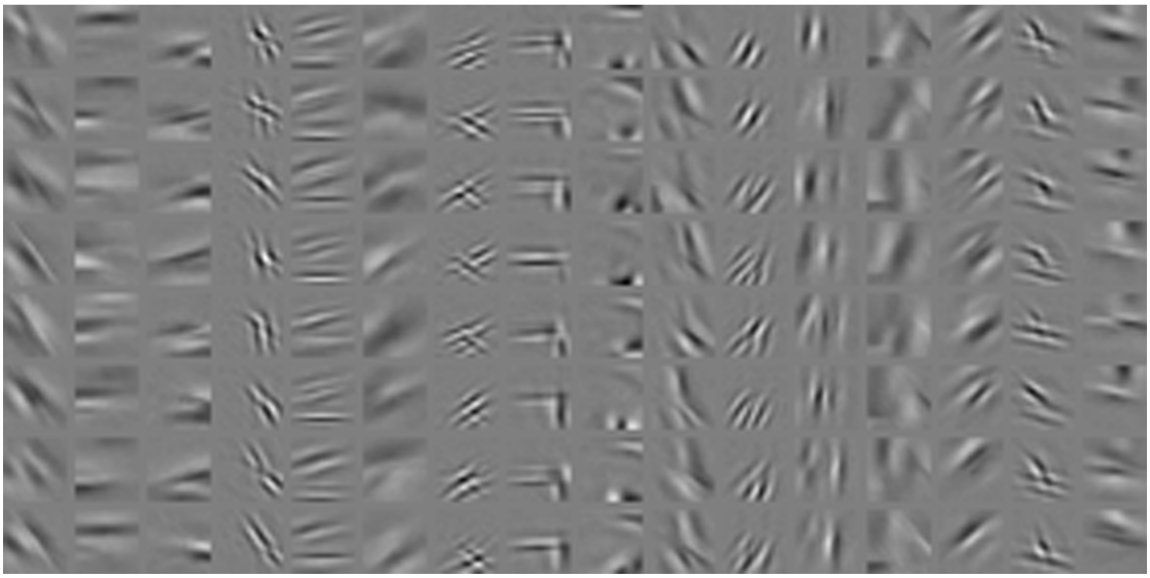


Figure 2.5: **Transformational Bilinear Sparse Coding:** Effect of transformations on base features. Learned $G_l U_k$ components are shown, with each column corresponding to a single k and each row to a single l.

While the expanded dictionary $\{U_{km}\}_{m=1}^{M}$ is used for inferring the pose embedding z_k, the resulting transformed features used in the reconstruction are not discrete in nature. Figures 2.6 to 2.9 show different instances of dynamically generated features for each group. As seen these generally represent distinct entities transformed in different ways.

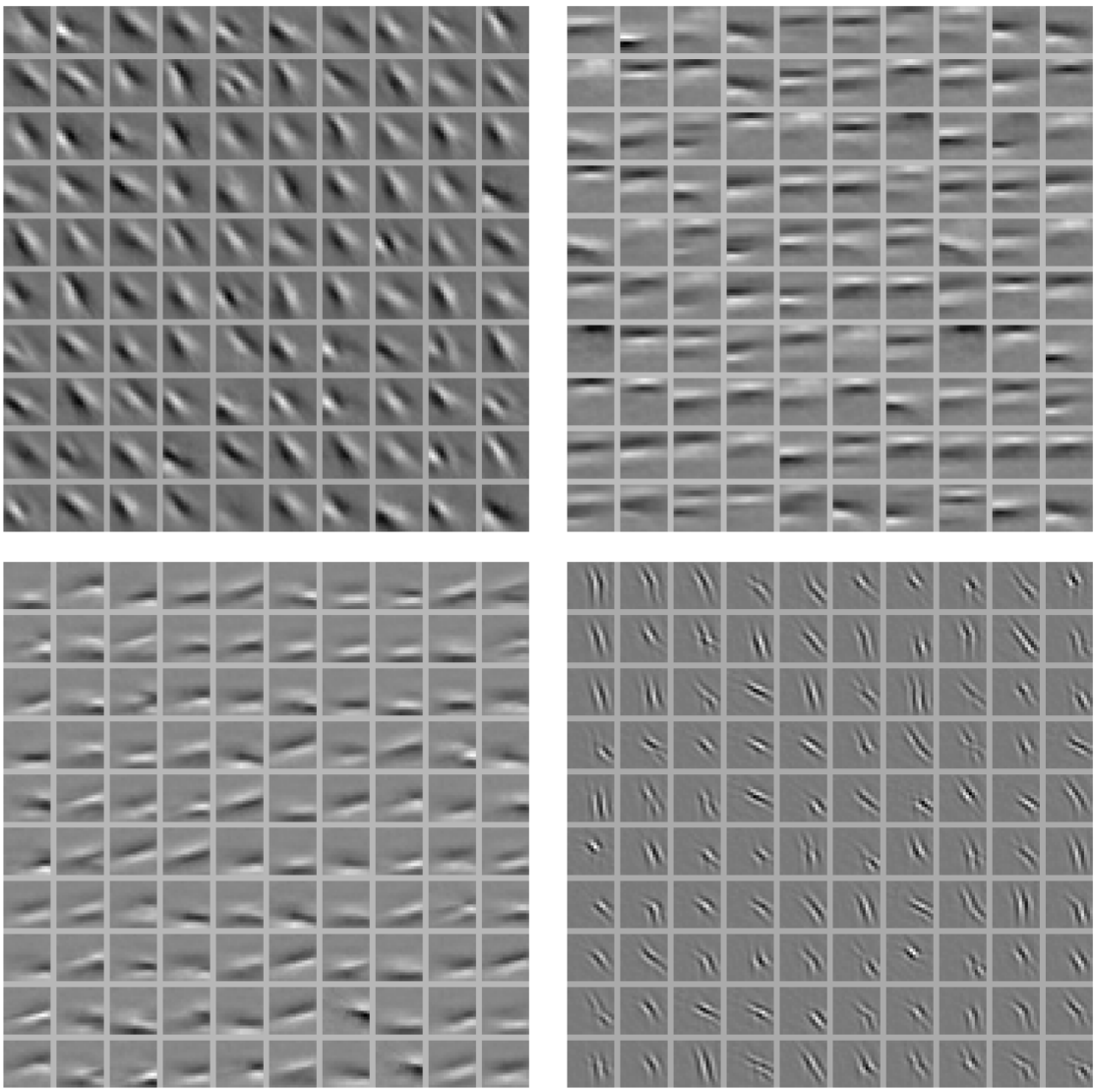


Figure 2.6: **Dynamically generated TBSC features for groups 1-4.**

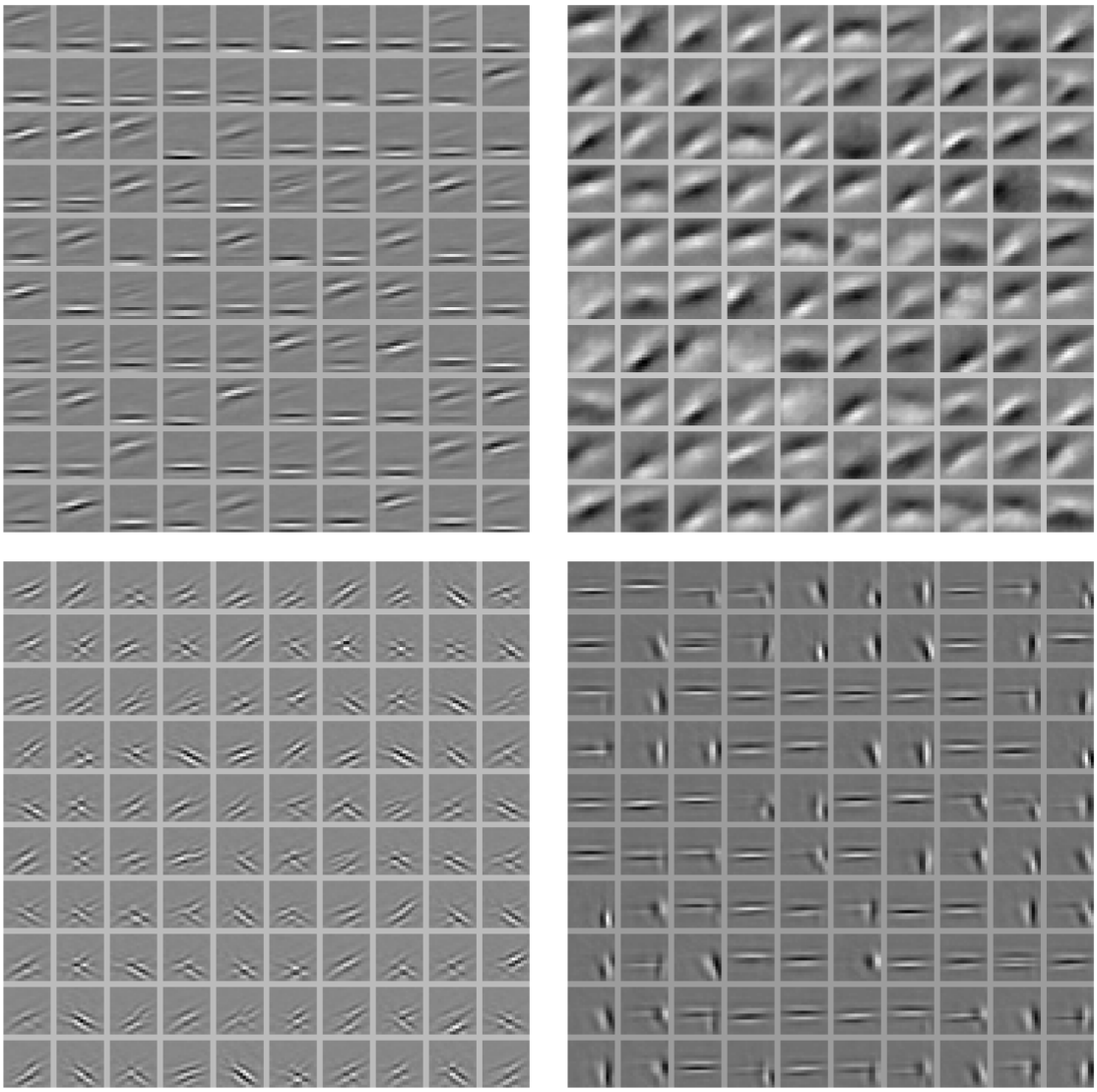


Figure 2.7: **Dynamically generated TBSC features for groups 5-8.**

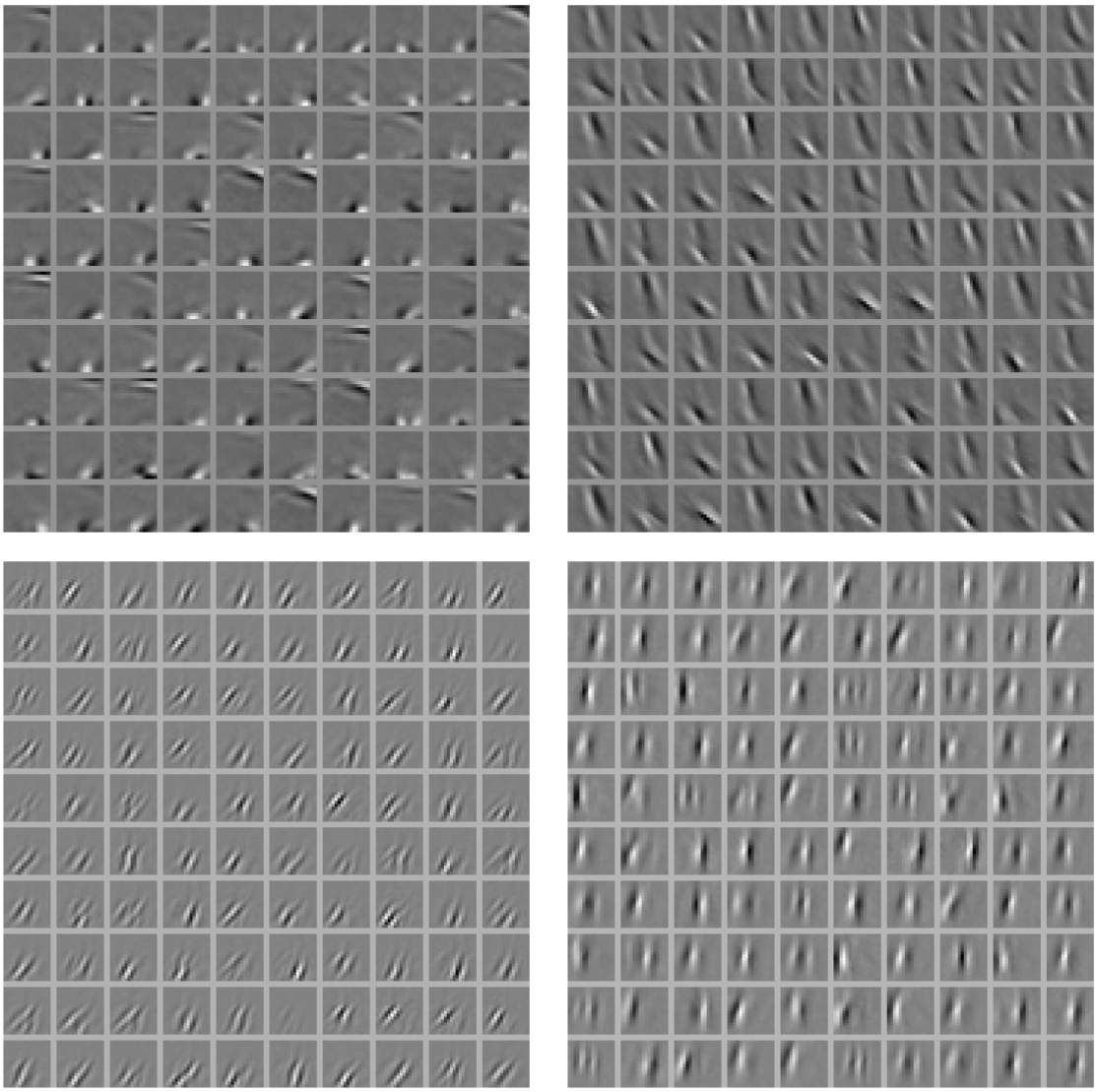


Figure 2.8: **Dynamically generated TBSC features for groups 9-12.**

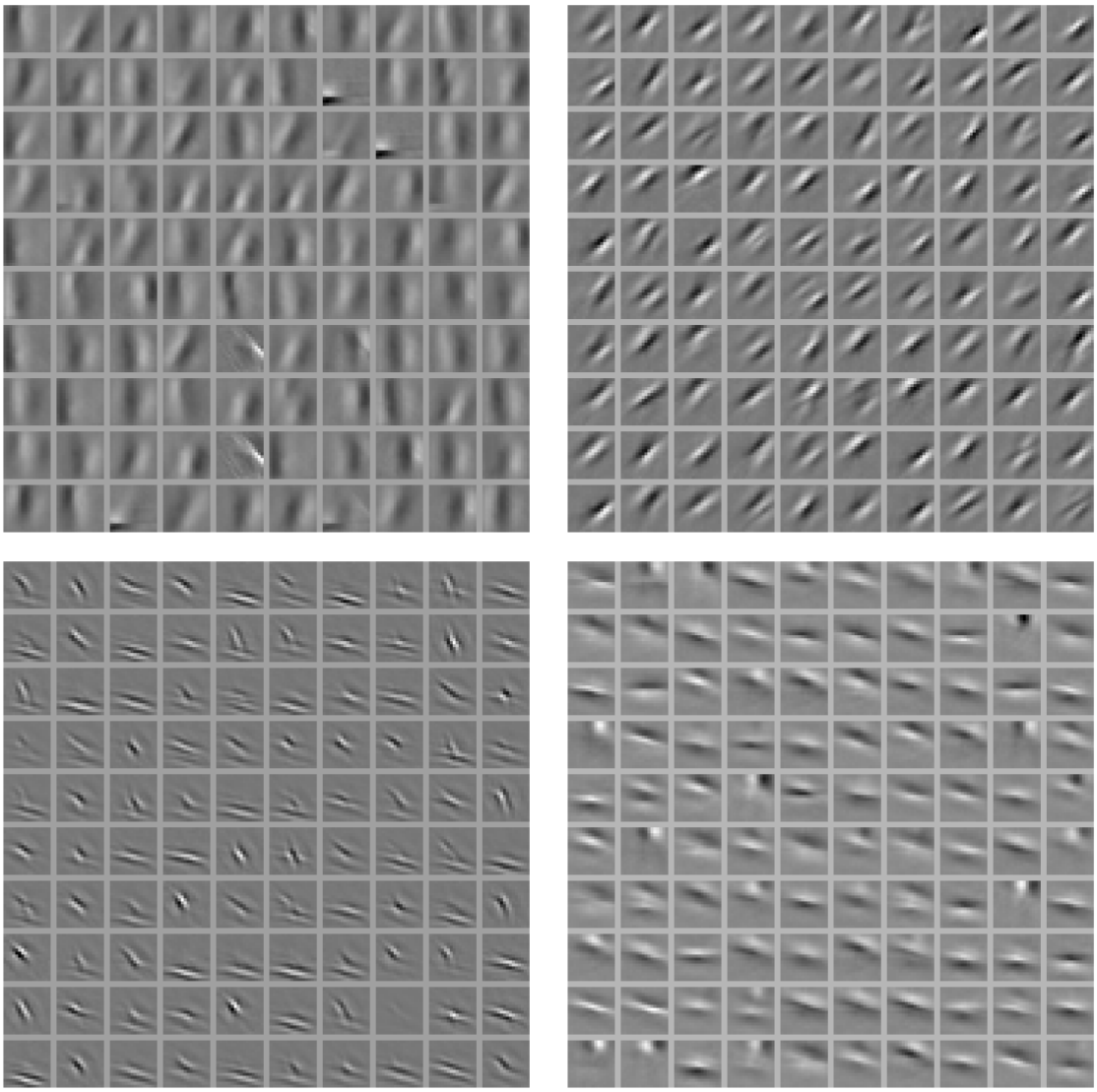


Figure 2.9: **Dynamically generated TBSC features for groups 13-16.**

To determine whether each feature group captures the progression of a single underlying feature across different time steps, we visualized the evolution of features across sequences of frames. Figures 2.10 to 2.17 show how features in each group progress throughout different video sequences. The learned features evolve across frames in a manner similar to spatiotemporal filters in the visual cortex, e.g., direction-selective Gabor filters moving in a particular direction.

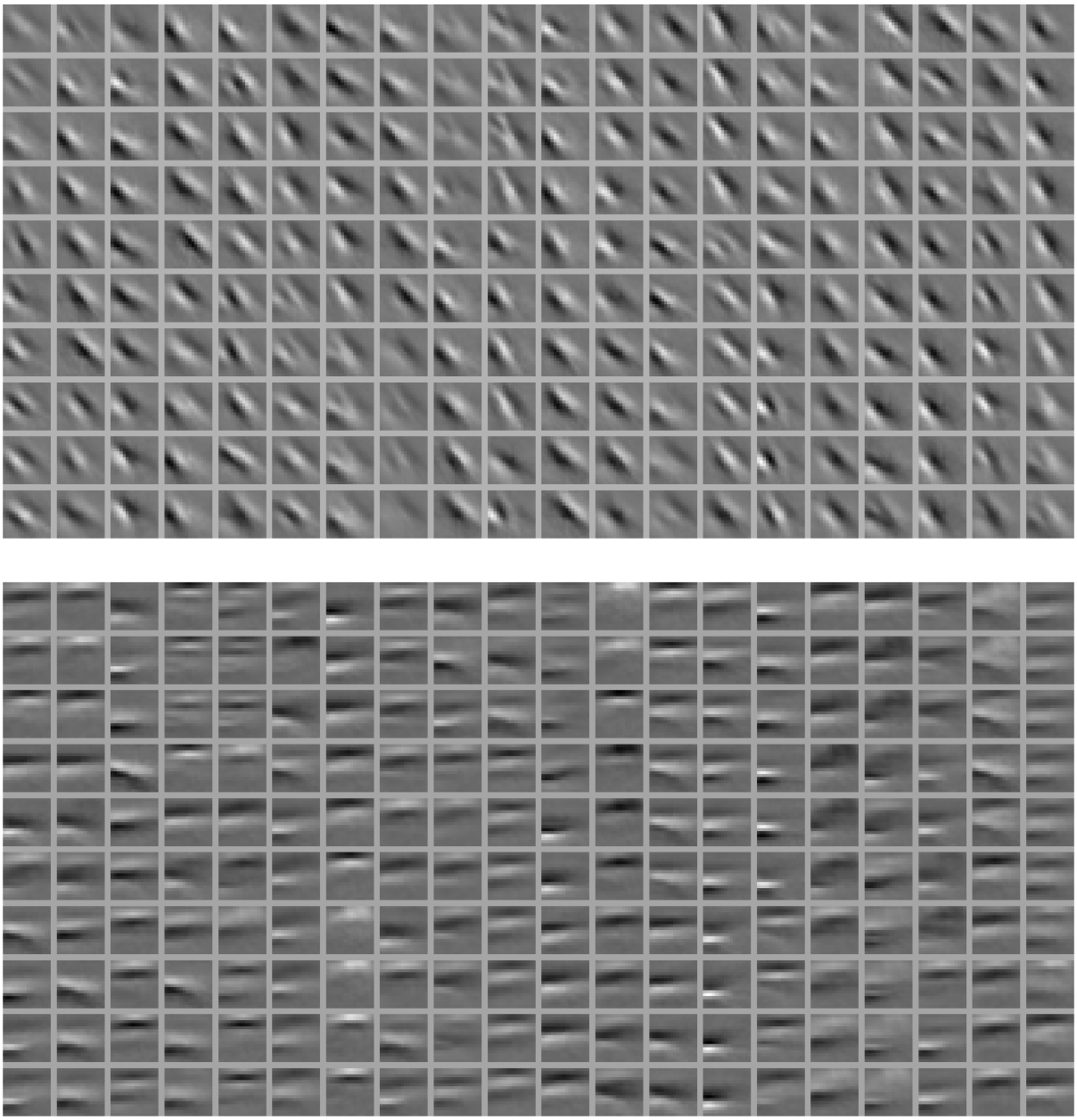


Figure 2.10: **Progressions of TBSC features through time for groups 1-2:** Each column shows a specific instance of a dynamic feature from a group, evolving throughout the 10 frames of the video sequence.

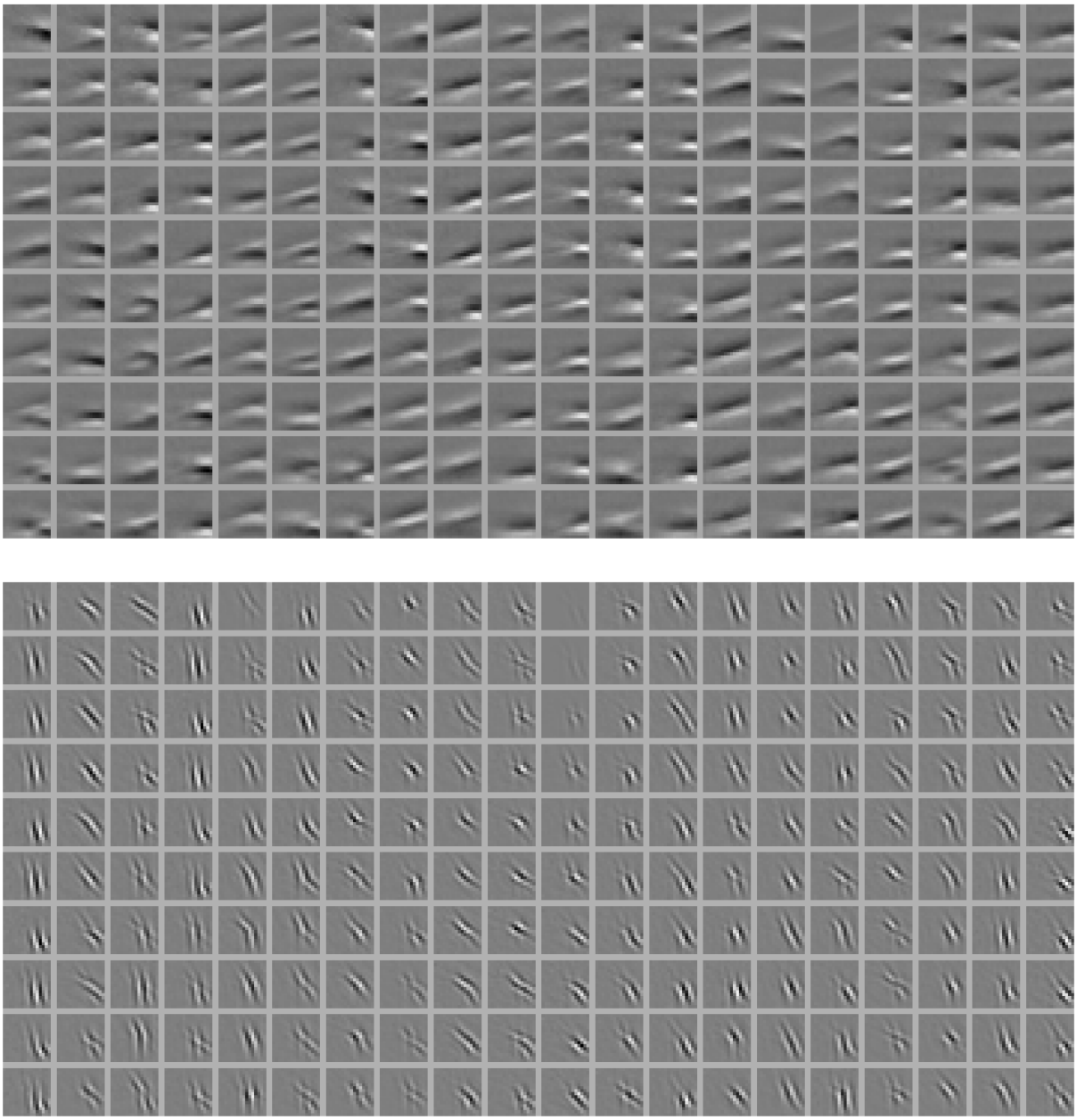


Figure 2.11: **Progressions of TBSC features through time for groups 3-4:** Each column shows a specific instance of a dynamic feature from a group, evolving throughout the 10 frames of the video sequence.

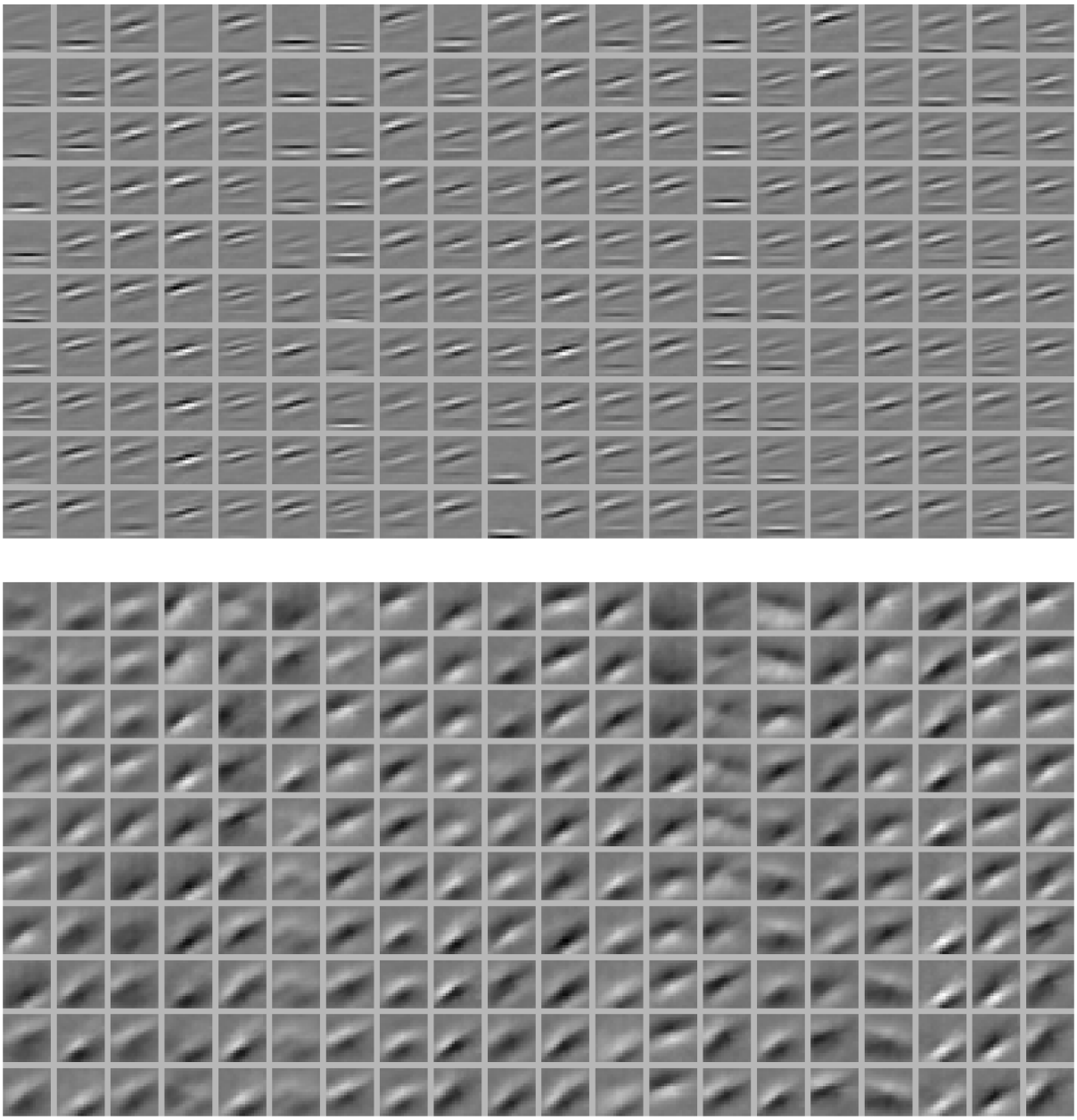


Figure 2.12: **Progressions of TBSC features through time for groups 5-6:** Each column shows a specific instance of a dynamic feature from a group, evolving throughout the 10 frames of the video sequence.

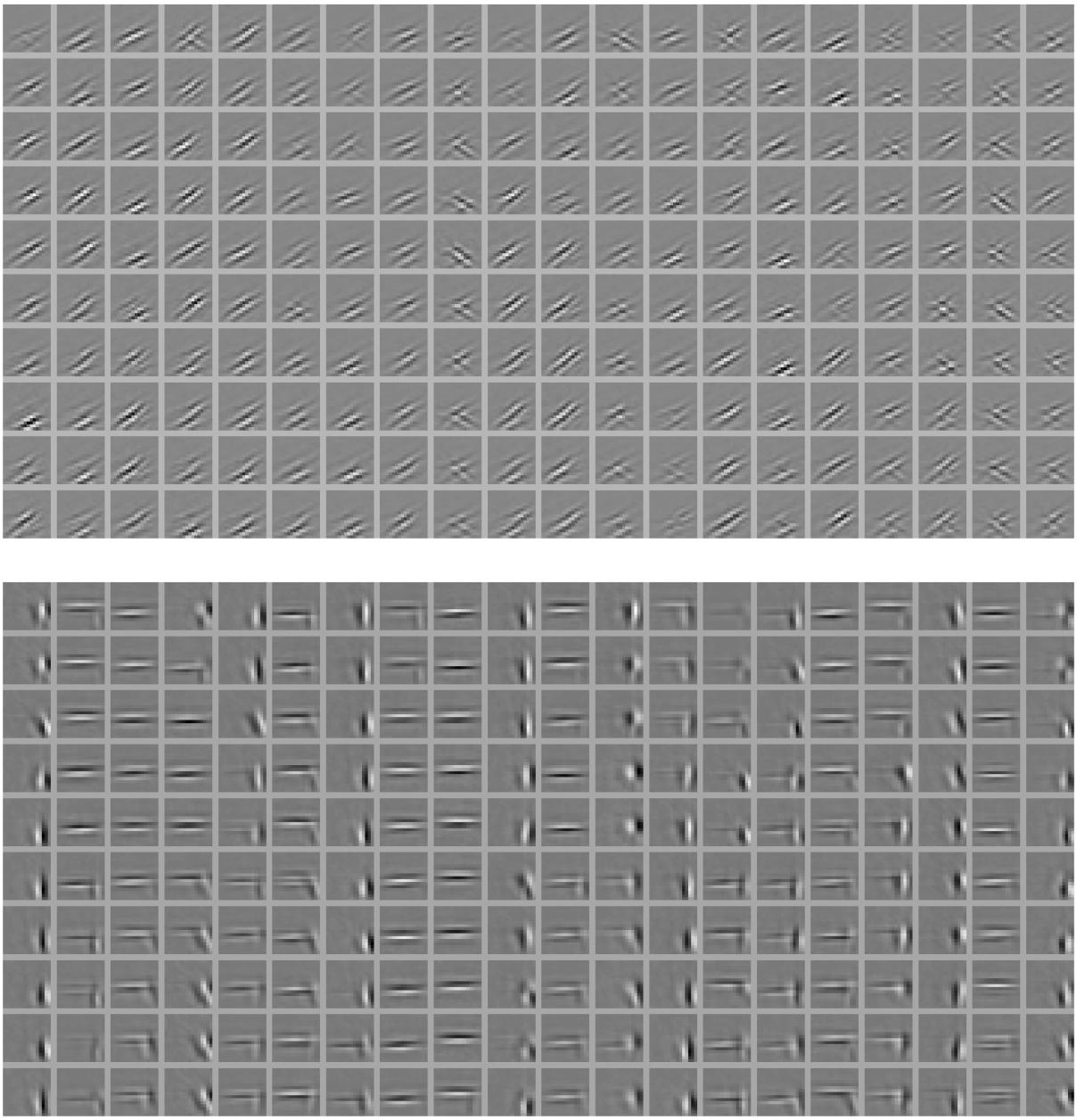


Figure 2.13: **Progressions of TBSC features through time for groups 7-8:** Each column shows a specific instance of a dynamic feature from a group, evolving throughout the 10 frames of the video sequence.

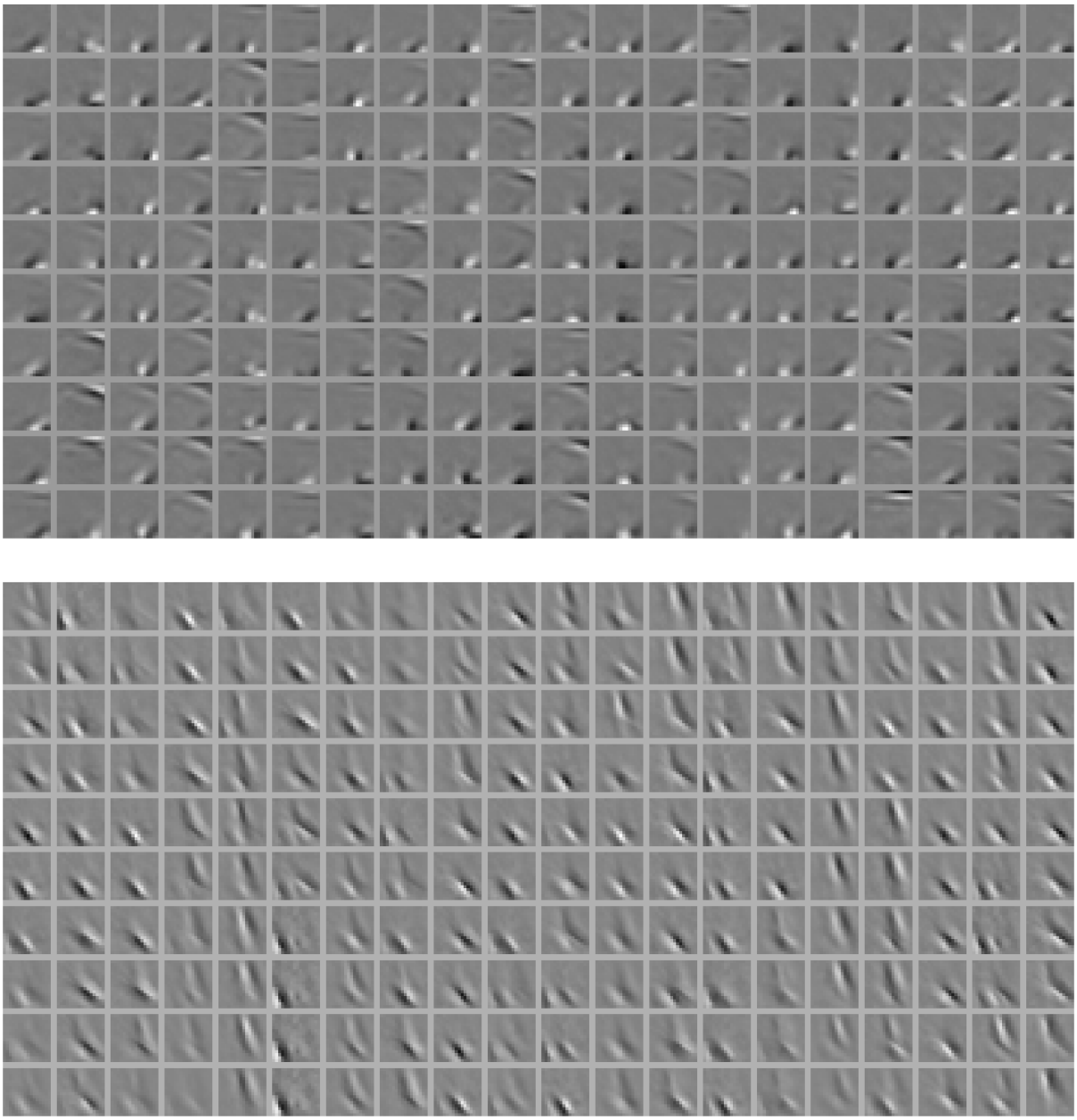


Figure 2.14: **Progressions of TBSC features through time for groups 9-10:** Each column shows a specific instance of a dynamic feature from a group, evolving throughout the 10 frames of the video sequence.

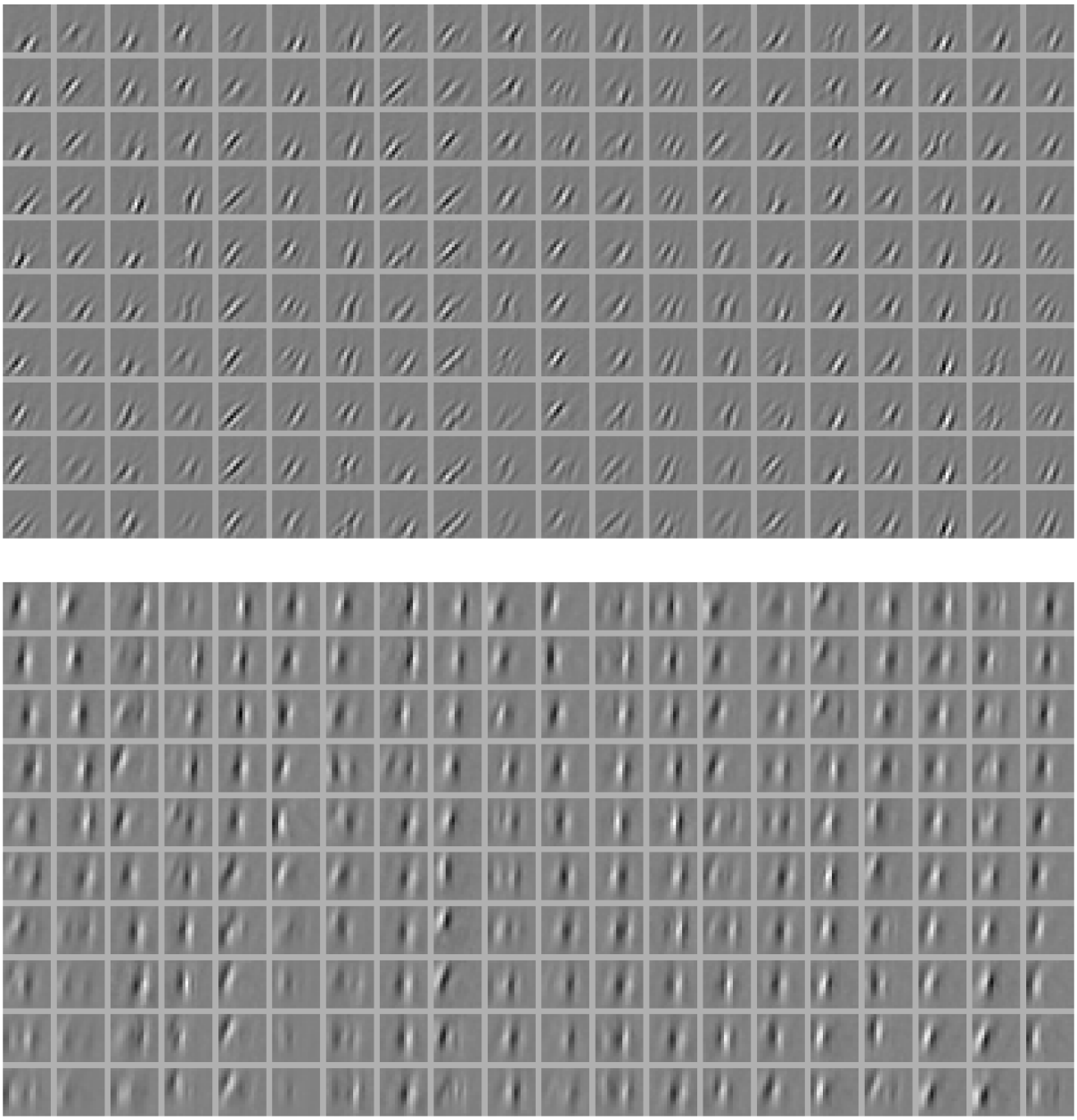


Figure 2.15: **Progressions of TBSC features through time for groups 11-12:** Each column shows a specific instance of a dynamic feature from a group, evolving throughout the 10 frames of the video sequence.

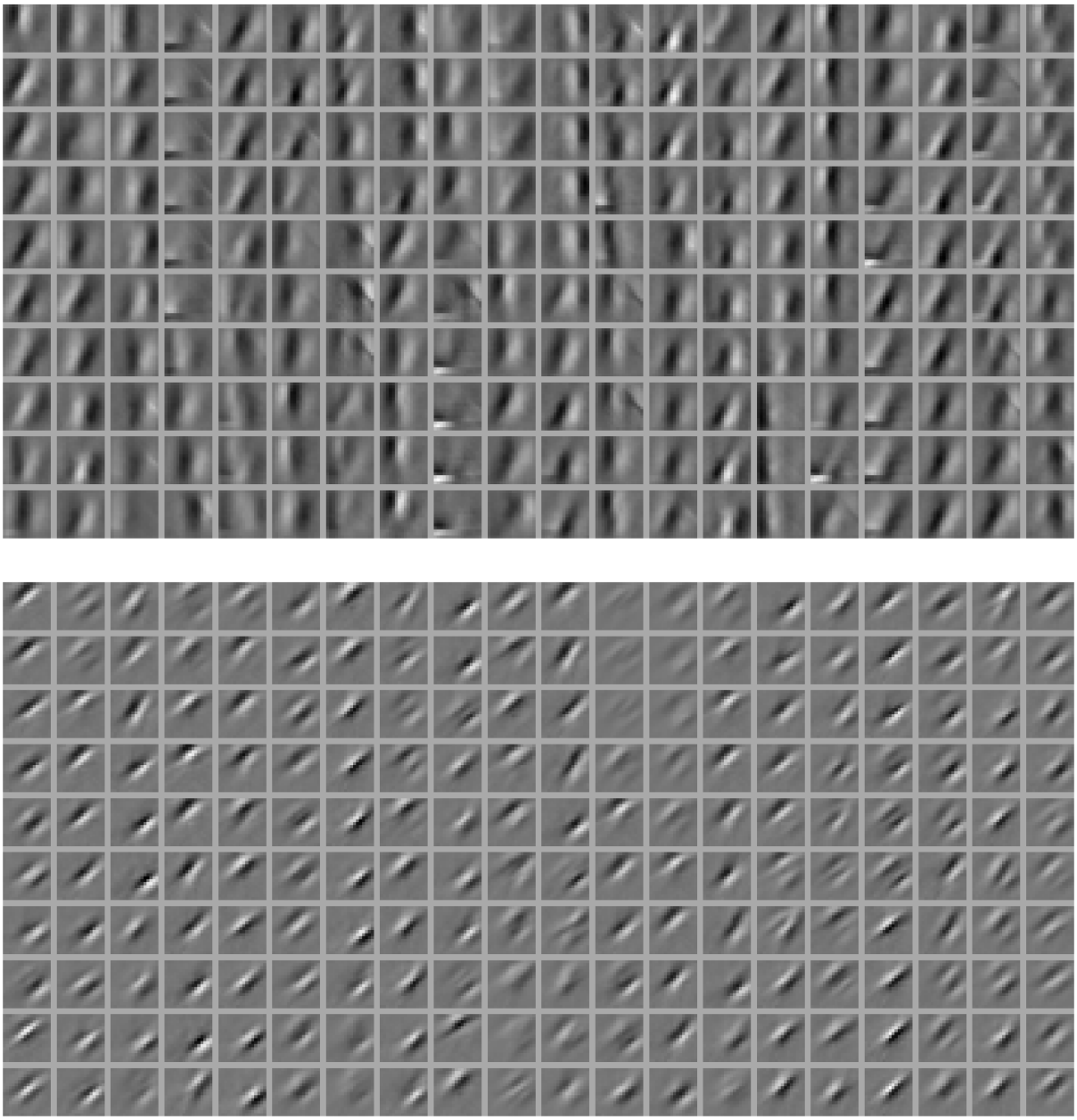


Figure 2.16: **Progressions of TBSC features through time for groups 13-14:** Each column shows a specific instance of a dynamic feature from a group, evolving throughout the 10 frames of the video sequence.

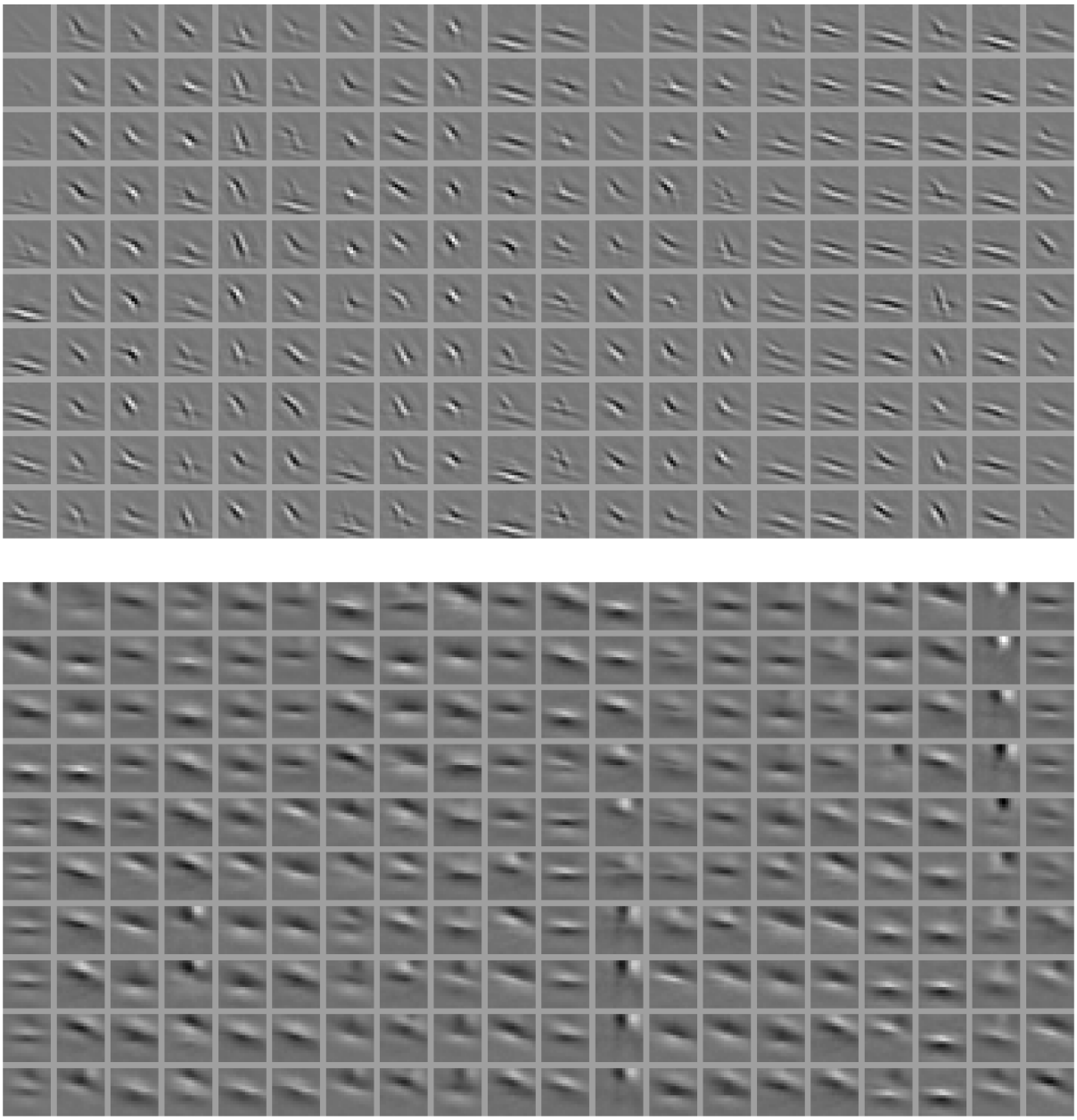


Figure 2.17: **Progressions of TBSC features through time for groups 15-16:** Each column shows a specific instance of a dynamic spatiotemporal feature from the same group, evolving throughout the 10 frames of the video sequence.

2.2.5 Conclusion

In this section I introduced Transformational Bilinear Sparse Coding, a hypernetwork-based bilinear model (TBSC) that learns features and their transformations from video, leveraging the smoothness of natural videos. The main limitation of this work is that I discussed only a single layer model. Stacking multiple TBSC layers, each receiving the previous layer's pooled code as input is a promising direction for future research.

2.3 Dynamic Predictive Coding

An important problem in theoretical neuroscience is providing a normative framework for understanding the orientation and direction selective receptive fields (RFs) of neurons in the primary visual cortex (V1). The sparse coding model and its bilinear version introduced in Section 2.1 apply only to static images. The transformational version of the bilinear model (TSCB) from Section 2.2 utilizes temporal information but makes the unrealistic assumption that the whole video sequence is available at once. The predictive coding model of Rao & Ballard [56] has emerged as an influential model of visual processing, but it too focuses on spatial receptive field properties. In this section, we propose Dynamic Predictive Coding (DPC) [36]. DPC combines predictive and sparse coding together with hypernetworks to explicitly learn the transition dynamics between the hidden states of a temporally changing visual world and uses it to generate temporal predictions.

2.3.1 Model

DPC uses a spatial generative model similar to sparse coding 2.1 and predictive coding [56]: $I_t = Ur_t + n$ where I_t is the input image at time t, U the features in the dictionary and r_t the sparse activations at time t. Note that these sparse activations change over time as opposed to the invariant activations of TBSC, since feature pose is not explicitly modeled.

DPC augments sparse coding with a temporal prediction model that consists of: (a) a set of basis transition matrices $\{V_k\}_{k=1}^{K}$ and (b) a recurrent modulation network $\mathcal{H}$ that adjusts

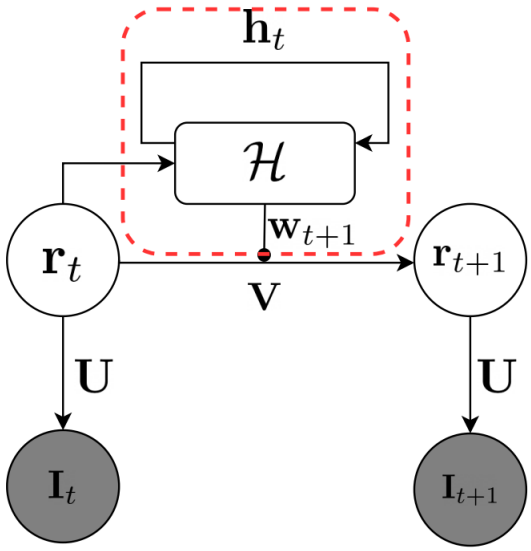


Figure 2.18: **Generative model for Dynamic Predictive Coding:** Red box: novel addition of a modulation network for generating dynamic transition matrices V for nonlinear prediction.

the linear mixing weights for these matrices (Fig. 2.18, red dashed box). At each time step t, $\mathcal{H}$ takes the current state r_t and the recurrent state h_t representing the history of past states to generate the next set of mixture weights and recurrent state: $w_{t+1}, h_{t+1} = \mathcal{H}(r_t, h_t)$. The time-varying transition matrix is then generated as: $V_{t+1} = \sum_{k=1}^{K} w_{t+1,k} V_k$. The sparse hidden state for time $t+1$ is predicted as: $\widehat{r}_{t+1} = \text{ReLU}\left(V_{t+1} r_t\right)$. Given the next input I_{t+1}, the prediction error is corrected using sparse coding to obtain a corrected estimate for r_{t+1} from the predicted $\widehat{r}_{t+1}$. The model was learned by jointly minimizing the mean-squared prediction error $\|I_t - U r_t\|_2^2$ and $\|r_t - \widehat{r}_t\|_2^2$ at all time steps with a sparsity constraint on r_t. Note that the DPC model can be regarded as a dynamic hypernetwork (Section 1.1.2). DPC also shares some similarities with the approaches in [35, 8].

2.3.2 Results

We trained the model on 5,700 natural video segments extracted from a video recording by a person walking on a forest trail. The frame size is 16×16 pixels and the segment length 20 frames ($\sim$0.7 seconds). The frames were spatially and temporally whitened to simulate retinal and LGN processing using kernels derived in [11]. We used $K = 5$ basis transition matrices (results did not change significantly with more matrices). The hidden state vector r was represented by 1000 model neurons. The test set consisted of 1,500 segments extracted from a different section of the recording.

	Training MSE	Test MSE
Static V	0.435	0.451
Dynamic V	0.355	**0.409**

Table 2.1: **Video Prediction MSE:** The dynamic approach outperforms the static one on test set MSE.

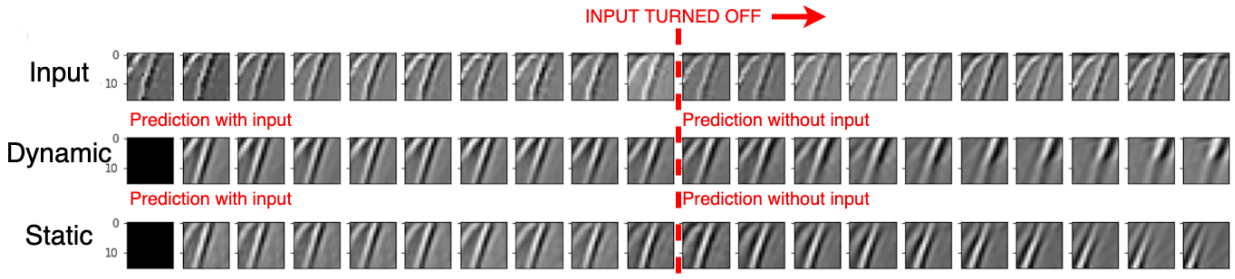


Figure 2.19: **Dynamic vs Static Predictive Coding Prediction Progressions.**

The dynamic predictive coding model outperformed the traditional model with a static V matrix (Table 2.1) as measured by the mean-squared image prediction loss $\|Ur_t - U\widehat{r}_t\|_2^2$ for both the training and the test datasets. This difference is illustrated in the example in Figure 2.19 which compares the temporal image predictions made by the dynamic model (middle

row) and the static model (bottom row). Both models generate predictions visually similar to the input but when the input is turned off (red dashed line), the static transition model generates the wrong direction of movement (leftwards) while the dynamic model correctly predicts motion towards the right, similar to the true input (top row).

These results demonstrate the advantage of using hypernetwork-modulated transition dynamics for temporal predictions. A future research direction would be to combine this model with TBSC by applying the transition dynamics to the pose embeddings z.

Chapter 3

ACTIVE PREDICTIVE CODING NETWORKS

Deep convolutional neural networks have enabled path-breaking advances in visual classification problems in recent years [43] but they suffer from a fundamental shortcoming: they do not preserve positional information about extracted features. Even though they may correctly classify an image, they are unable to explain the images they classify in the way humans do: in terms of objects, their locations in a scene, the parts of an object and the locations of these parts within the object, etc. This lack of interpretability of deep neural networks has prompted a search for alternate models that are inspired by how humans represent objects in terms of part-whole hierarchies and use compositionality of parts to explain new objects. For example, Hinton and colleagues [58, 41, 31] have explored a class of networks called Capsule networks which use a group of neurons ("capsule") to explicitly represent not only the presence of an object but also parameters such as position and orientation. More recently, Hinton [30] has proposed an "imaginary system" called GLOM to overcome some of the limitations of capsule networks. Independently, Hawkins and colleagues [47] have taken inspiration from neuroscience, specifically cortical columns and grid cells, to propose that the brain uses object-centered reference frames to represent objects, spatial environments and even abstract concepts.

What has been missing is a scalable framework that solves the following problem: how can neural networks learn intrinsic reference frames for objects and parse visual scenes into part-whole hierarchies by dynamically allocating nodes in a parse tree? In this chapter I introduce Active Predictive Coding Networks (APCNs) [22], a class of structured neural networks inspired by the neocortex that address this problem using hypernetworks [27] to learn and dynamically generate parse trees from images.

APCNs contribute to a number of lines of research that have not been connected before:

- **Predictive Coding:** APCNs build on predictive coding models of cortical function [56, 17, 36], which emphasize the role of hierarchical prediction and prediction errors in driving learning and inference.

- **Visual Attention Networks:** APCNs extend previous visual attention approaches such as the Recurrent Attention Model (RAM) [52, 2] and Attend-Infer-Repeat (AIR) [15, 40] by learning structured strategies for sampling the visual scene. We also define appropriate baselines for these types of models to demonstrate the utility of intelligent sampling.

- **Hierarchical Reinforcement Learning:** APCNs leverage ideas in hierarchical reinforcement learning by learning abstract macro-actions ("options" [66]) to hierarchically parse an image into parse trees via hypothesis testing.

By combining predictive coding, active sampling of the visual scene and hierarchical actions, our approach also suggests a neural solution to an important challenge posed by cognitive science and AI researchers [45]: how can neural networks in the brain and in AI learn hierarchical compositional representations that allow new objects, scenes and concepts to be quickly created, recognized and learned?

3.1 Active Predictive Coding Networks

Suppose there is an optical sensor with limited computational capacity connected to a device with large computational capacity via a communication channel with limited bandwidth. How can the sensor intelligently sample the scene to allow the computational device to parse and understand the scene? There is a direct correspondence between this arrangement and our visual system: the sensor is the retina, the device the visual cortex and the channel the optic nerve. As opposed to typical CNNs in AI, humans rely on intelligent sequential

sampling of visual scenes via eye movements ("saccades"). The Recurrent Attention Model (RAM) [52] and related approaches [15, 2] emulate this idea by utilizing a "glimpse sensor" that extracts high-resolution information about small parts of a larger input image; this information is conveyed to artificial neural networks for further processing. For example, in the RAM model, a "location network" decides on which location in the image to sample a glimpse from, and a recurrent neural network (RNN) integrates the sampled information for downstream tasks. Since the glimpse sensor used is not differentiable, the location network is trained via the reinforcement learning algorithm REINFORCE [65, 52].

Here we introduce active predictive coding networks (APCNs), which build on ideas explored by RAM and other models in the following ways:

- Information from glimpses are organized in a structured, hierarchical way using intrinsic reference frames computed by a hierarchical network.

- Inspired by the formalism of Partially Observable Markov Decision Processes (POMDPs) [37, 57], each level of the hierarchical network is composed of a state network and an action network. The state network at each level integrates the information from input samples and implements the state transition model for POMDPs at a particular level of abstraction. The action network at each level is task specific and responsible for planning actions at that particular level of abstraction.

- At every hierarchical level, the state network is trained via predictive coding, while the action network is trained via REINFORCE.

3.1.1 Basic Idea

At each level of a hierarchy, the APCN model uses two embedding vectors, one to represent the current "state" denoting an object/part, and the other to represent the current action denoting the position (or more generally, the transformation) of the object/part. Nonlinear functions (implemented as hypernetworks [27]) are used to map these vectors to lower level

state transition and action functions, which act as "programs" to parse various parts/sub-parts via sequences of sampled locations/transformations. This process can be repeated for an arbitrary number of levels.

Figure 3.1 (a) formalizes this idea and shows the canonical generative module for APCNs. The module consists of a higher level state vector $\mathbf{r}^{(i+1)}$, which uses a function (hypernetwork) H_s^i to generate a lower level state transition function f_s^i (implemented as an RNN), and a higher level action vector $\mathbf{a}^{(i+1)}$, which uses a function (hypernetwork) H_a^i to generate a lower level action (or policy) function f_a^i (also implemented as an RNN). In this thesis, I will focus on a two-level model (with a top level and bottom level) as shown in Figure 3.1 (b). The generation of states and actions for the two levels is shown in Figure 3.1 (c). The state and action RNNs at the lower level are generated independently by their parent RNNs (via the action/state embedding vectors) but exchange information horizontally within each level as shown in Figure 3.1 (c): the state network generates the next state prediction based on the current state and action while the action network generates the current action based on the current state and previous action.

In the context of parsing an image, the action vector at a given level chooses which sub-tree of the parse tree to explore next, while the state vector represents all the integrated scene information provided by the lower levels. The exploration of a sub-tree proceeds by dynamically generating state and action "sub-programs" for the level below via hypernetworks. In the present implementation, the lower level RNNs execute for a fixed number of time steps, generating parts and their locations, before returning control back to the higher level.[1] The higher level state then transitions to the next state (the next object/part) using that level's transition function F_s and the action specified by the higher level policy F_a, and the process continues.

[1]Future implementations will explore the use of termination functions [66, 15] to allow a variable number of time steps at each level and for each example.

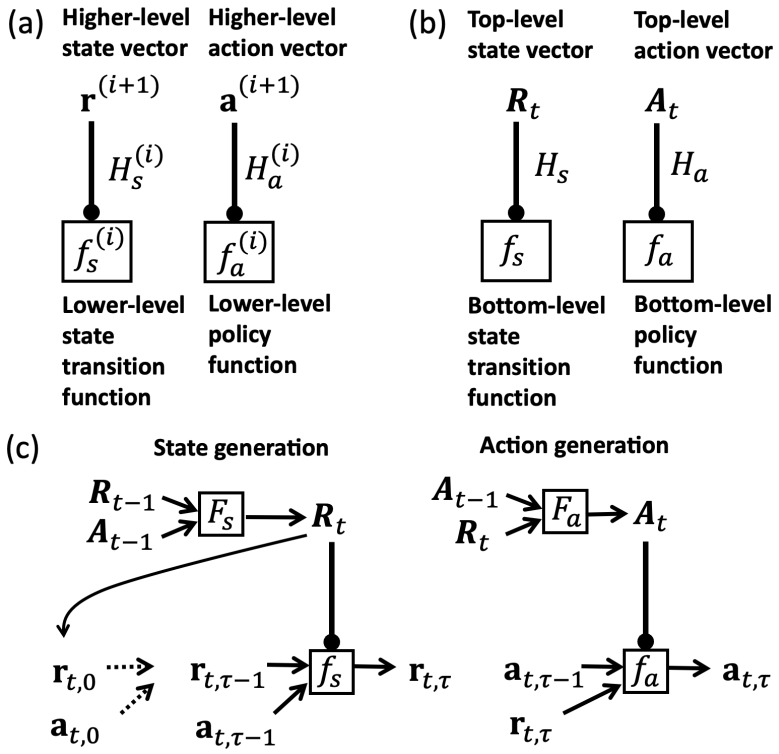


Figure 3.1: **Active Predictive Coding Networks (APCNs):** (a) Canonical generative module for hierarchical APCNs. The lower level functions are generated via hypernetworks based on the current higher level state and action embedding vectors. All functions (in boxes) are implemented as recurrent neural networks (RNNs). Arrows with circular terminations denote generation of function parameters (here, neural network weights and biases). (b) Two-level model used in this work. (c) Generation of states and actions in the two-level model based on past states and actions. RNNs implementing the functions in boxes additionally receive feedback from prediction errors and lower level states/actions as described in the text.

3.1.2 Parse Trees, Reference Frames and the Glimpse Sensor

Figure 3.2 depicts an example of a simple parse tree for a human body. We can think of the union of all these parse trees for all potential scenes as a graph representing a structured ontology of "parent-child" relations. This graph is hierarchical and consists of different layers, with connections between them representing part/sub-part relations. An APCN explores this ontology graph by testing different branches at each layer and extracts an appropriate parse tree for the scene.

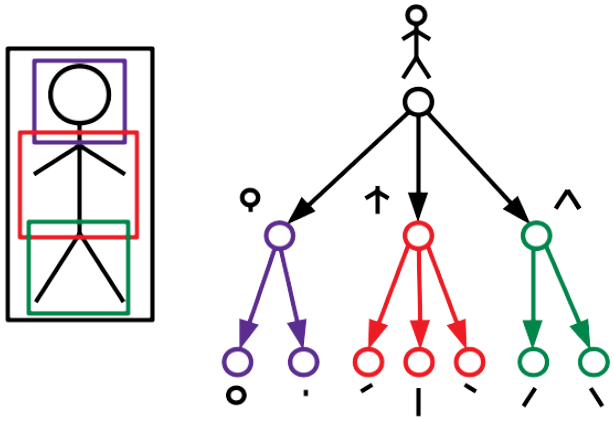
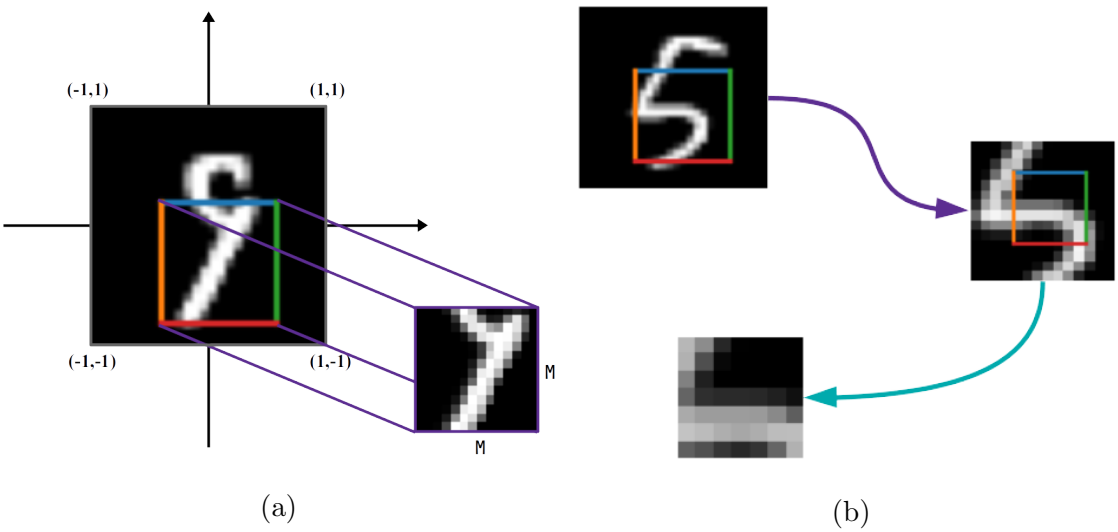


Figure 3.2: **Example of a Parse Tree for a Human Body:** Black box: Frame of reference for the entire object. Purple, red, green boxes: Frames of reference for the head, upper and lower body respectively. The parse tree on the right shows the part-whole hierarchy with respect to these reference frames. The leaves of the tree correspond to the lowest-level parts of the object. Locations for all parts are computed within the parent node's reference frame, e.g., the locations for the head, upper body and lower body are computed with respect to the body's reference frame (black box).



(a) (b)

Figure 3.3: **Reference Frames for Images:** (a) The top level of the model picks a location and focuses on a sub-region of a pre-specified size (here, $M \times M$) centered at that location. This sub-region contains a higher-level part of the object at a location relative to the original reference frame of the object (here, the digit "9"). This sub-region selected by the higher level in turn acts as the reference frame for the lower level. (b) Two-level model. The top level focuses on a sub-region $\frac{1}{4}$ the area of the initial input and fixes this region as the current reference frame. The next level focuses on locations within this local frame of reference and extracts sub-sub-regions, which contain sub-parts of the higher level part at particular locations, all calculated relative to the local frame of reference. This hierarchical factoring of parts, sub-parts and their transformations within local reference frames is critical for compositionality.

Parse trees for images imply spatial convergence as one goes up the representational hierarchy since typically an entity has a larger spatial extent than its constituent parts. APCNs implement this idea using recursive object-centered reference frames. The top level of an APCN architecture spans the entire image. At each step, the network chooses a sub-region of the image to focus on (Figure 3.3a). It then generates a lower-level parser

(comprised of state-action sub-networks) and assigns this image sub-region as the input. The bottom-most level has direct access to the image via small-sized glimpses. APCNs perform a type of depth-first exploration of the representational graph where each layer descends deeper into the graph with a new object-centered reference frame. These stacks of reference frames can be composed to derive the absolute location of any sampled glimpse within the image. Figure 3.3b shows an example of recursive reference frame traversal down a two-level hierarchy.

Interactions with an image I (of size $N \times N$ pixels) are carried out through a glimpse sensor G. This sensor takes in a location l and a fixed scale fraction m, and extracts a square glimpse/patch $g = G(I, l, m)$ centered around l and of size $(mN) \times (mN)$. Since l is continuous, the sensor is implemented using a sub-differentiable bilinear interpolation module as introduced in [34]. The image dimensions are normalized so that $l \in [-1, 1]$ and m is hard-coded for each layer. Other transformations such as rotation and shear are ignored in the current version of this model but present an obvious direction for future research.

Note that APCNs share some similarities with the Transformational Bilinear Sparse Coding (TBSC) model introduced in Section 2.2. Both approaches model the image using a basis set of image patches and their transformations. APCNs currently only use translations but model the image hierarchically, whereas TBSC learns the transformations directly from video sequences.

3.1.3 Inference in the Active Predictive Coding Network

Without loss of generality, we consider a two-level version of the APCN architecture in this chapter, with the understanding that it can be easily extended to more levels. The two levels operate at different time scales. For a given input, the top level runs for T_2 steps which we will refer to as "macro-steps." For each macro-step, the bottom level runs for T_1 "micro-steps," yielding a total of $T = T_2 T_1$ steps for the entire network to process each input. Let F_s, F_a be the top level state and action networks respectively. Let R_t, A_t be the recurrent activity vectors of these networks (i.e., the top level state and action vectors) at macro-step

t. Let $f(;\theta)$ denote a network parameterized by θ. In the case of a fully connected network with L layers, $\theta = \{W_l, b_l\}_{l=1}^{L}$ contains the weight matrices and biases for all the layers. The state and action RNNs of the bottom level are denoted by $f_s(;\theta_s)$ and $f_a(;\theta_a)$ while their activity vectors are denoted by $r_{t,\tau}$ and $a_{t,\tau}$ respectively, where t ranges over macro-steps and τ over micro-steps. Figure 3.1C depicts these RNNs and activity vectors.

Higher-Level Network Operation

At each macro-step t, the top level action RNN updates its activity vector to A_t which generates two values: (a) a location L_t and (b) a macro-action (or option) z_t (Figure 3.4a). The location L_t is used to restrict the bottom level to a sub-region $I_t^{(1)} = G(I, L_t, M)$ corresponding to a new frame of reference of scale M, centered around L_t (Figure 3.3a). The option z_t is used as an embedding vector input to a non-linear function, implemented by a hypernetwork H_a, to dynamically generate the parameters $\theta_a(t) = H_a(z_t)$ of the lower-level action RNN. For exploration during reinforcement learning, we treat the output of the location network as a mean value $\bar{L}_t$ and add Gaussian noise with fixed variance to sample an actual location: $L_t = \bar{L}_t + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

The state vector R_t and location L_t are fed as inputs to the state hypernetwork H_s to generate the parameters $\theta_s(t) = H_s(R_t, L_t)$ specifying a dynamically generated bottom-level state RNN for the current frame of reference. Figure 3.4a illustrates this top-down generation process.

Lower-Level Network Operation

At the beginning of each micro-step, the higher-level state R_t is used to initialize the bottom-level state vector via a small feedforward network Init_s to produce $r_{t,0} = \text{Init}_s(R_t)$ (Figure 3.4b). Each micro-step proceeds in a manner similar to a macro-step. The bottom-level action RNN updates its activity vector $a_{t,\tau}$ based on the current state and past action, and a location $l_{t,\tau}$ is chosen as a function of $a_{t,\tau}$ (Figure 3.4a, lower right). This results in a glimpse image $g_{t,\tau} = G(I_t^{(1)}, l_{t,\tau}, m)$ of scale m centered around $l_{t,\tau}$ within the image sub-region $I_t^{(1)}$

specified by the higher level (Figure 3.4c). The frames of reference and the corresponding image sub-regions across the two levels are depicted in Figure 3.3b.

To predict the next glimpse image at the location specified by the action network, the lower-level state vector $r_{t,\tau}$, along with locations L_t and $l_{t,\tau}$, are fed to a generic decoder network D to generate the predicted glimpse $\hat{g}_{t,\tau}$ (Figure 3.4c). This predicted glimpse is compared to the actual glimpse image to generate a prediction error $\epsilon_{t,\tau} = g_{t,\tau} - \hat{g}_{t,\tau}$. Following the predictive coding model [56], the prediction error is used to update the state vector via the state network: $r_{t,\tau+1} = f_s(r_{t,\tau}, \epsilon_{t,\tau}, l_t; \theta^{(s)}(t))$ (Figure 3.4a, lower left). For the bottom-level locations, we follow the same Gaussian noise-based exploration strategy as the top-level.

At the end of each macro-step (after T_1 bottom-level micro-steps have finished executing), the top level state RNN activity vector is updated using the final bottom-level state vector and the top-level location: $R_{t+1} = F_s(R_t, \rho_s(r_{t,T_1}), L_t)$ where $\rho_s()$ is a single-layer state "feedback" network. Figure 3.4d (left side) depicts this process. The top-level action RNN activity vector A_t is then updated using R_{t+1} and $\rho_a(a_{t,T_1})$ (Figure 3.4d (right side)), and the process continues. The above steps correspond to a sub-program in the state/action hierarchy terminating and returning its result to be integrated by its parent. Note that this architecture can be readily extended to more levels by having F_s, F_a be dynamically generated by another parent level, and so on.

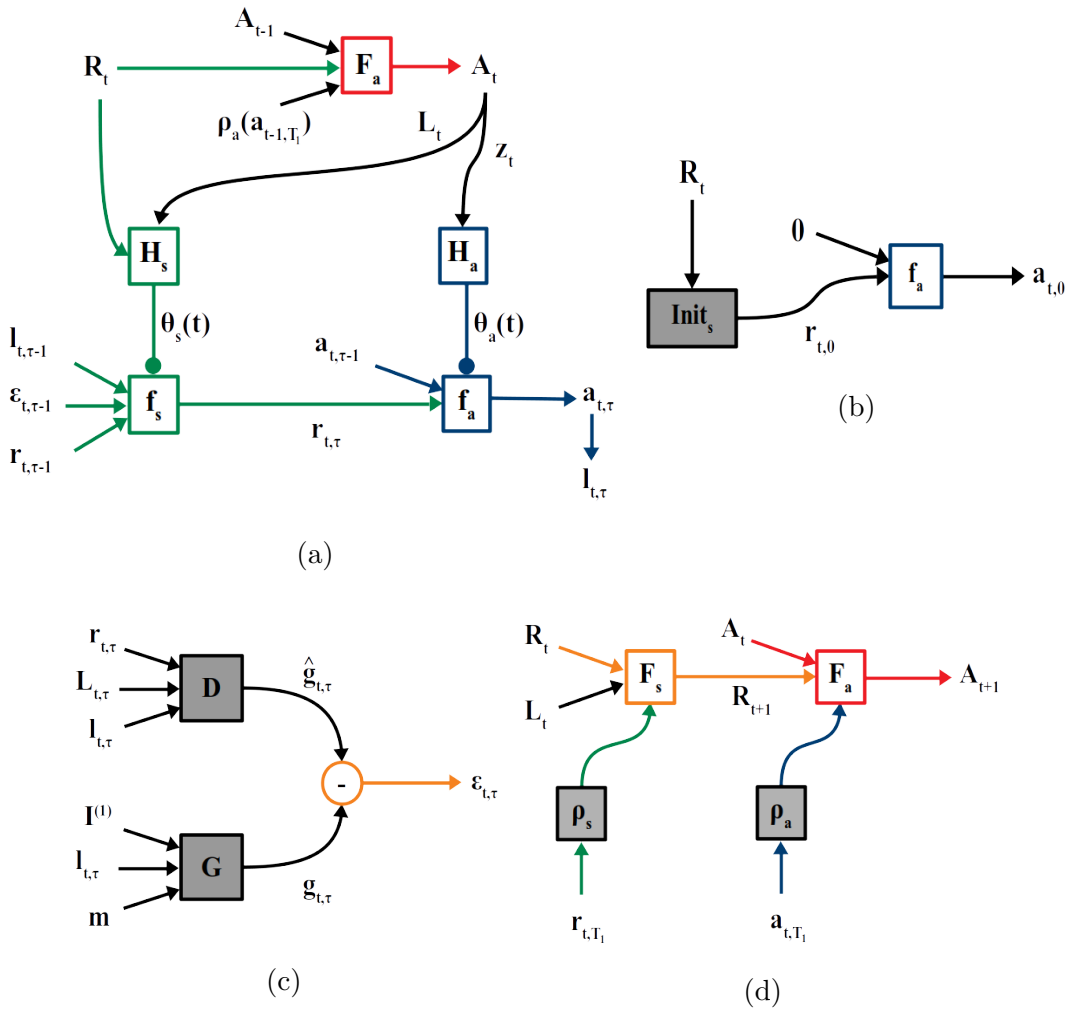


Figure 3.4: **Inference in Active Predictive Coding Networks:** (a) Dynamic generation of bottom-level state RNN f_s and action RNN f_a ("sub-programs") from top-level state vector R_t and action vector A_t. This diagram elaborates the one in Figure 3.1 (c) for the case of parsing images, showing how actions produce locations and options, and how prediction errors and feedback from the lower level are used to update state vectors at the two levels. See text for details. (b) Initialization of bottom-level state by the higher-level state R_t via a network Init_s. (c) Computation of prediction error between predicted glimpse $\hat{g}$ generated by decoder D for the current time step (t, τ) and actual glimpse image g from the glimpse sensor after moving to the location $l_{t,\tau}$ produced by the action network. (d) Update of top-level state R_t and action A_t based on feedback (via networks ρ_s and ρ_a) upon bottom-level sub-program termination (after T_1 micro-steps).

3.1.4 Training the Active Predictive Coding Network

The state and action networks are trained separately via different loss functions. The state networks are trained to minimize prediction errors via backpropagation while the action networks are trained to minimize total expected task loss via REINFORCE together with backpropagation. During training, whenever the state vectors at any given level are passed as input to that level's action network (see Figure 3.4a), the gradients for backpropagation are cut off. The goal of the state prediction network is to predict the next state and is task-agnostic. The goal of the action network is to choose effective actions given past states and actions, so that the task loss is minimized.

State Networks

The prediction error $\epsilon_{t,\tau}$ is given by:

$$\epsilon_{t,\tau} = g_{t,\tau} - \hat{g}_{t,\tau} = G(I_t^{(1)}, l_{t,\tau}, m) - D(r_{t,\tau}, L_t, l_{t,\tau}) \tag{3.1}$$

The prediction error loss function is given by:

$$L_{\text{pred}} = \sum_{t=1}^{T_2} \sum_{\tau=1}^{T_1} \|\epsilon_{t,\tau}\|_2^2 \tag{3.2}$$

At the end of a macro-step t, the higher level also reconstructs the current reference image $I_{\text{ref}}^{(1)}$, downsampled to the size of a lower-level glimpse, using a decoder D_{ref} with inputs R_{t+1} and L_t, yielding the loss function $L_{\text{ref}} = \sum_{t=1}^{T_2} \|I_{\text{ref}}^{(1)} - D_{\text{ref}}(R_{t+1}, L_t)\|_2^2$. The total loss function for training the state networks at the two levels via backpropagation is given by:

$$L_{\text{state}} = L_{\text{pred}} + L_{\text{ref}} \tag{3.3}$$

Action Networks

To apply APCNs to a given task (such as image reconstruction or classification), either the state or action RNN vectors can be provided as input to another neural network trained for the task. Here we use the action vectors. Let $A_{\text{out}}(t, \tau) = [A_t \; a_{t,\tau}]^T$ be the concatenation of

top- and bottom-level action vectors for time step (t, τ). Let L_{task} be the task loss. Using just the final A_{out} (as in RAM [52]) for training actions has the shortcoming that the resulting reward function is sparse (the model is evaluated only after the final step). We use a dense, structured reward function (in our case, a dense loss function) as follows. For each micro-step, we compute the marginal change in loss after the action for that step (i.e., fixating on a new location) has been executed:

$$R_{t,\tau} = L_{\text{task}}(A_{\text{out}}(t, \tau - 1)) - L_{\text{task}}(A_{\text{out}}(t, \tau)) \tag{3.4}$$

For example, if the task is reconstruction of an image, the reward is positive if the new action (new fixation location) reduced the reconstruction error.

For each macro-step, we compute the marginal change in loss due to the whole macro-step:

$$R_t = L_{\text{task}}(A_{\text{out}}(t - 1, T_1)) - L_{\text{task}}(A_{\text{out}}(t, T_1)) \tag{3.5}$$

The top layer is trained using the cumulative reward from all future macro-steps $\Phi_t = \sum_{i=t}^{T_2} R_i$, whereas the bottom layer is trained using the future rewards within each macro-step $\Phi_{t,\tau} = \sum_{j=\tau}^{T_1} R_{t,j}$. This corresponds to the intuition that micro-actions taken inside different frames of reference should not affect each other in terms of reward.

We use an adjusted version of the baseline-based variance reduction technique introduced in [65] and used in [52]. We learn two separate baselines: $b_{t,\tau} = \mathbb{E}[\Phi_{t,\tau}]$ and $b_t = \mathbb{E}[\Phi_t]$ and use the baseline-removed cumulative rewards $\Phi_{t,\tau} - b_{t,\tau}$ and $\Phi_t - b_t$ for training.

The REINFORCE loss is given by:

$$L_{\text{REINFORCE}} = J(\theta_{\text{L}}, \theta_{\text{l}}) = -\sum_{t=1}^{T_2} \underbrace{\left(\log P(L_t | A_t; \theta_L)(\Phi_t - b_t) + \sum_{\tau=1}^{T_1} \log P(l_{t,\tau} | a_{t,\tau}; \theta_l)(\Phi_{t,\tau} - b_{t,\tau}) \right)}_{\text{Action log-probabilities}} \tag{3.6}$$

As mentioned earlier, to allow exploration during training with REINFORCE, the locations at each macro- or micro- step were the location network's output plus Gaussian noise. Therefore the logarithmic probability terms above reduce to the squared Euclidean distances between the mean and the sampled locations.

We combine the REINFORCE loss with a dense version of the task loss to get the combined loss function for the action networks:

$$L_{\text{action}} = \underbrace{L_{\text{REINFORCE}}}_{\text{Location networks}} + \underbrace{\sum_{t}\sum_{\tau} L_{\text{task}}(A_{\text{out}}(t,\tau))}_{\text{Action sub-system minus location networks}} \tag{3.7}$$

For example, if the task is reconstruction, the second term in the combined loss allows minimization of the reconstruction error at every time step. Overall, the combined loss function increases the performance of the intermediate action vectors from step to step in the context of the task, producing more interpretable results.

Ideally the model would avoid generating locations exceeding the boundaries of the image. Several implementations of RAM [52] use clipping or the hyperbolic tangent activation function. In practice, I found out that constraining the locations via an appropriate penalty was more effective. I calculate a threshold c so that if a glimpse is centered c units away from the image boundary ($l \in [-1.0 + c, 1.0 - c]$), then the glimpse resides entirely within that boundary. I applied a thresholded version of ℓ_2 normalization:

$$L_{\text{reg}}(l) = (\text{LRelu}(l - c))^2 + (\text{LRelu}(-l - c))^2 - 2(\alpha c)^2$$

where LRelu is the leaky rectified linear unit with $\alpha = 0.2$. The structure of this penalty can be seen in Figure 3.5.

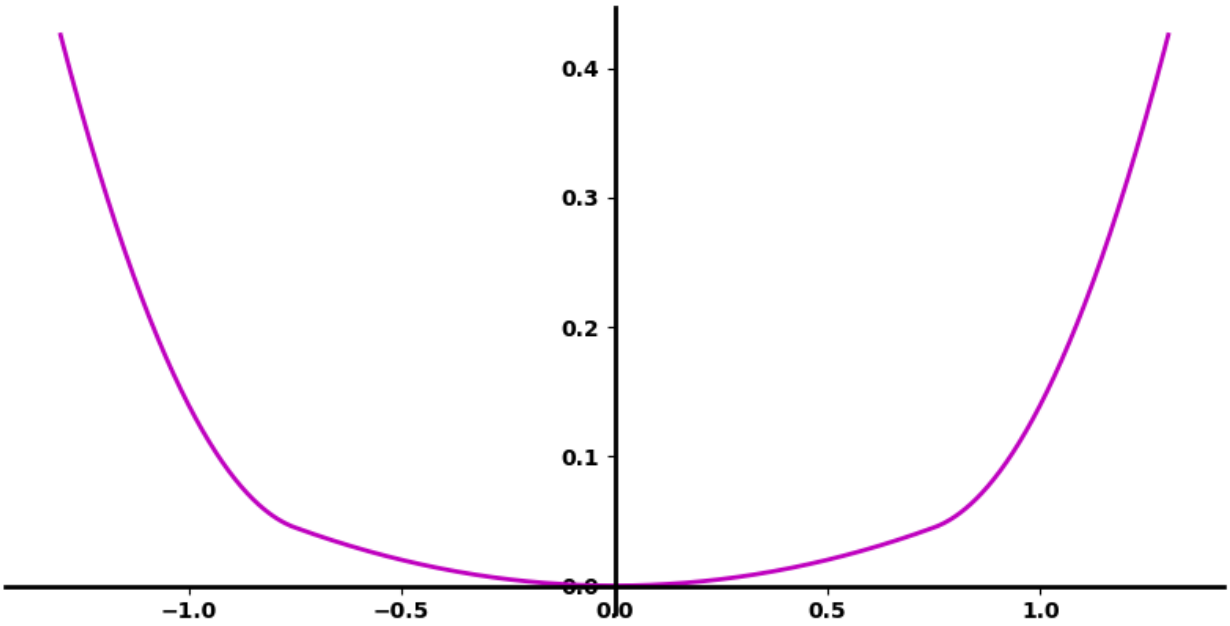


Figure 3.5: **APCN location penalty function for** $c = 0.75$**.**

3.2 Results

Our first set of experiments compares the performance of ACPNs to baseline methods. The second set of experiments demonstrates the ability of APCNs to learn part-whole hierarchies. The third set of experiments evaluates the compositionality of learned APCN representations in a transfer learning task.

3.2.1 Comparison with Baselines

Baseline 1: Random Policy

To evaluate whether the learned policies in APCNs are "intelligent," it is crucial to compare them with appropriate baselines. A simple baseline policy is randomly sampling different glimpse locations. We refer to this as the Randomized Baseline model with T glimpses or RB(T). We sample T i.i.d. locations $\{l_t\}_{t=1}^{T}$ from a box of height and width such that the whole glimpse g_t resides within the boundaries of the image. Each glimpse and its location are concatenated and passed through a feature extractor F to obtain a feature vector $f_t = F([g_t, l_t])$. The T feature vectors are averaged to obtain the latent vector $\bar{f} = \frac{1}{T}\sum_{t=1}^{T} f_T$. This vector is given as input to a feedforward network that is trained for the task.

The authors in [52] considered a baseline that consists of the RAM model applied to a single glimpse randomly sampled from the whole image. This baseline achieved 57.15% accuracy on the MNIST classification task. In our case, the RB(3) baseline achieved 93.1% classification accuracy.

Given this strong classification performance of RB(3), we conclude that simple datasets such as MNIST may be unsuitable for evaluating the performance of intelligent sampling (attention) models in classification tasks since a few random glimpses are sufficient to achieve reasonably high accuracy without any intelligent strategy. Our results also suggest that the "intelligent sampling" in RAM-like model for MNIST may be spurious, having no major impact on classification performance.

These results also suggest that rather than classification, the task of reconstructing an

object, such as an MNIST digit, might be a more appropriate task for learning and enumerating parts of an object. We therefore use image reconstruction as the task for evaluating APCNs.

Baseline 2: Single level APCN

Our second baseline is a single level version of our APCN model, which is similar to the original RAM model except: (a) instead of a single RNN, there is a separation into an action RNN responsible for the task and a state RNN that integrates glimpse information; (b) the state network is trained via predictive coding applied to predicting the next glimpses as described in Section 3.1.4; (c) we use dense rewards to improve training and interpretability. Note that all of the above are novel additions to the original RAM model, enabling the new model to: (a) re-use the state network for multiple tasks, and (b) make the contribution of each glimpse interpretable. We will refer to this model as APCN-1 and to the two layer APCN as APCN-2. Results comparing APCN-2 to the two baselines on the reconstruction task are described in the next section.

3.2.2 Learning Part-Whole Hierarchies via Active Predictive Coding

We applied APCNs to the task of part prediction and reconstruction of objects in the following datasets: (a) MNIST: Original MNIST dataset of 10 classes of handwritten digits [46]. (b) Fashion-MNIST: Instead of 10 digits as in MNIST, the dataset consists of 10 classes of clothing items [73]. (c) Omniglot: 1623 hand-written characters from 50 alphabets, with 20 samples per character [44]. (d) affNIST: MNIST digits embedded in a 40×40-pixel frame and transformed via random affine transformations. For all APCN models, a single dense layer, together with an initial random glimpse are used to initialize the state and action vectors of the top level.

	MNIST	Fashion MNIST	Omniglot-Test	Omniglot-Transfer
RB	0.0120	0.0145	0.0307	0.0301
APCN-1	0.0114	**0.0138**	0.0324	0.0323
APCN-2	**0.0085**	**0.0138**	**0.0227**	**0.0226**

Table 3.1: **Reconstruction Mean-Squared-Error (per pixel).**

Task Performance

We first applied APCNs to the task of reconstructing MNIST and Fashion-MNIST datasets. For APCN-2, we used 3 macro- and 3 micro-steps. A comparison of APCN-2 performance with the baselines based on test set MSE is shown in Table 3.1. Note that APCN-2 is more constrained than APCN-1 since the locations within a macro-step have to reside within the frame of reference of $I^{(1)}$. APCN-2 receives additional information in the form of T_2 peripheral glimpses obtained by downsampling $I_t^{(1)}$ as discussed in Section 3.1.4. These peripheral glimpses are used during training but not during inference. The fact that both APCN models perform better than the random baseline shows that intelligent sampling strategies have an effect on reconstruction task performance.

We also applied APCN-2 to reconstructing Omniglot characters using 4 macro-steps instead of 3. Table 3.1 shows that APCN-2 outperforms the baselines on this task.

Parse Strategies and Learned Part-Whole Hierarchies

An example of a learned parsing strategy is shown in Figure 3.6. For each input, APCN-2 learns structured parsing strategies: the top-level learns to cover the input image sufficiently while the bottom level learns to parse sub-parts inside those sub-areas of the object.

A learned part-whole hierarchy for an MNIST input, in the form of a parse tree of parts and sub-parts with locations, is shown in Figure 3.7. The learned strategies sample a wide variety of parts and sub-parts of the object (strokes and mini-strokes).

An important question is whether APCN-2 learns different parsing strategies and different part-whole hierarchies for different classes of objects. Figure 3.8 shows that this is indeed the case, for example, if we consider two different Fashion-MNIST clothing items, e.g., t-shirts versus sneakers. The figure shows that the average sampled locations of learned parts are different for these two different classes. Additional examples of learned higher-level part locations for each class in the Fashion-MNIST dataset are shown in Figure 3.9.

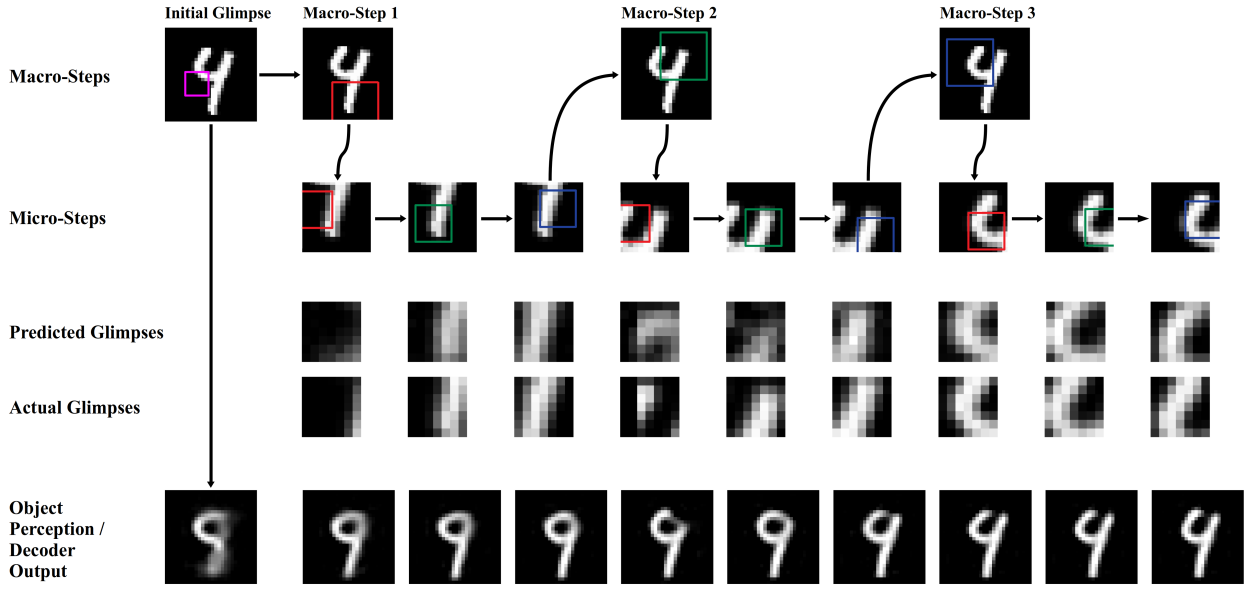
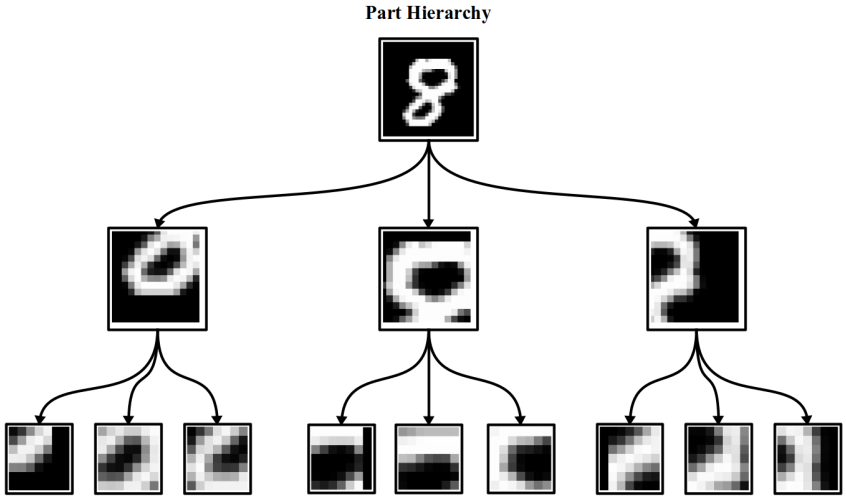
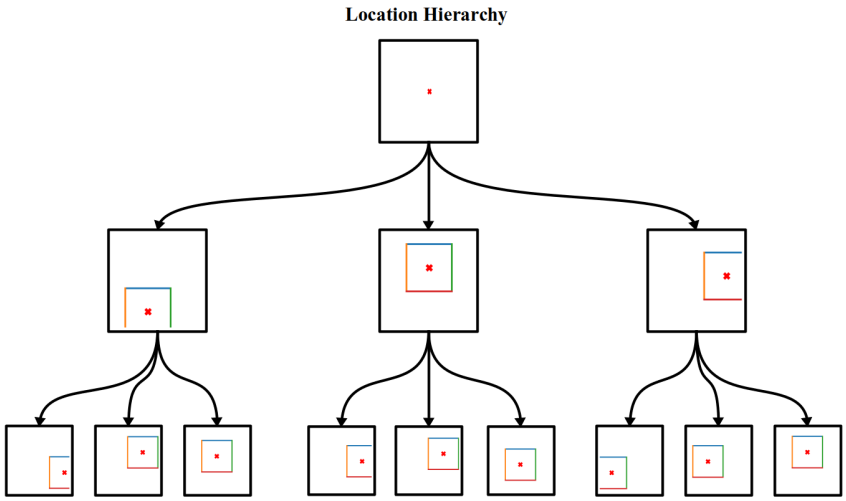


Figure 3.6: **Example of Learned Two-Level Parsing Strategy:** 1st row: Initialization glimpse (purple box) and sampled top-level reference frames (red, green, blue boxes), 2nd row: Sampled bottom level parts within each top-level frame, 3rd & 4th rows: Predicted versus actual parts/glimpses, and 5th row: "Perception" of the model (object reconstructed from current network state) over time.

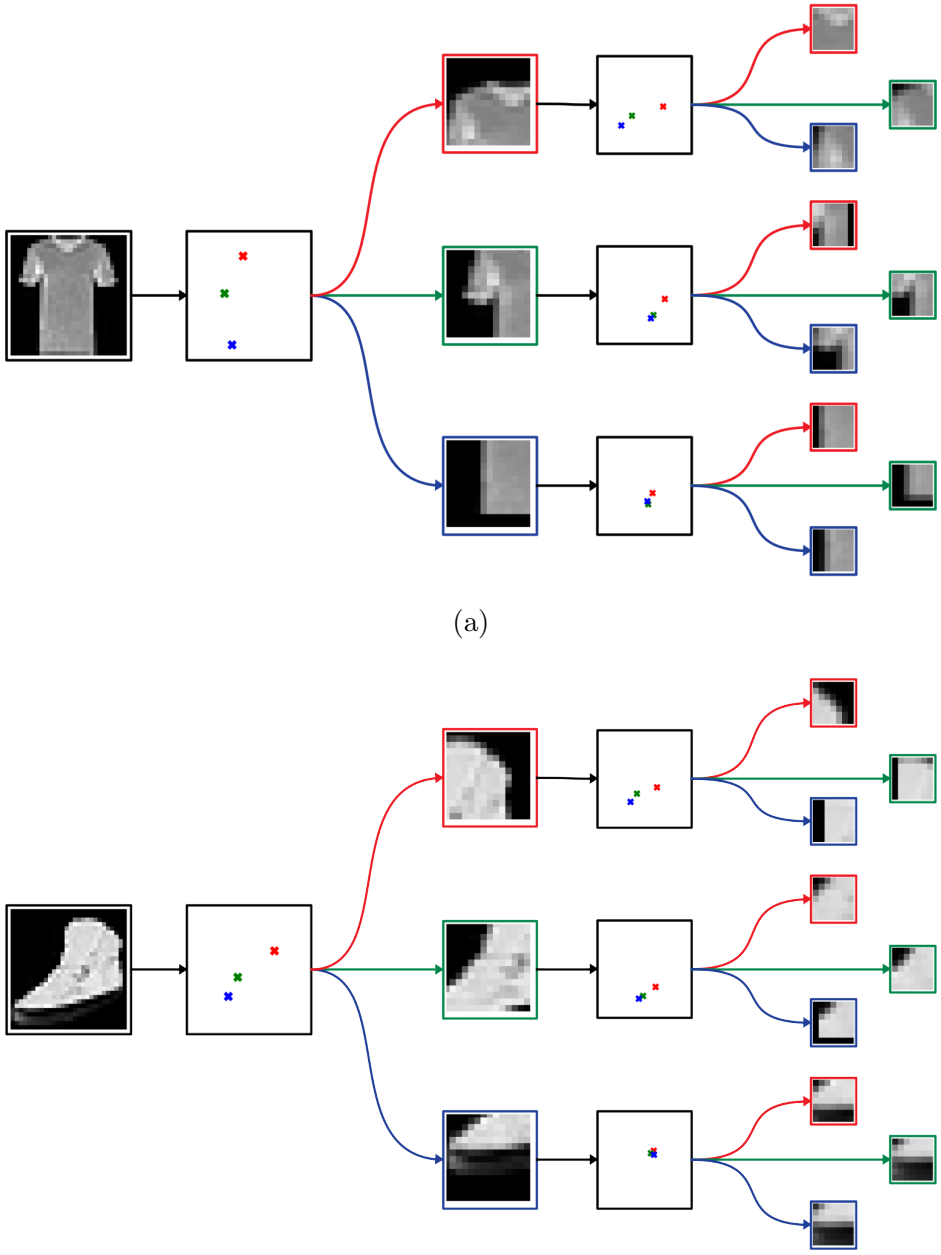


(a)



(b)

Figure 3.7: **Example Parse Tree with Inferred Locations of Parts:** Hierarchy of (a) sampled parts and (b) sampled locations, inducing a hierarchy of reference frames.



(a)

(b)

Figure 3.8: **Class-Based Hierarchical Representation of Object Parts and Locations:** Average sampled locations per class, together with sampled parts for one specific example for (a) the t-shirt and (b) the sneaker classes. The order of sampled locations within each frame of reference is 1st: red, 2nd: green and 3rd: blue.

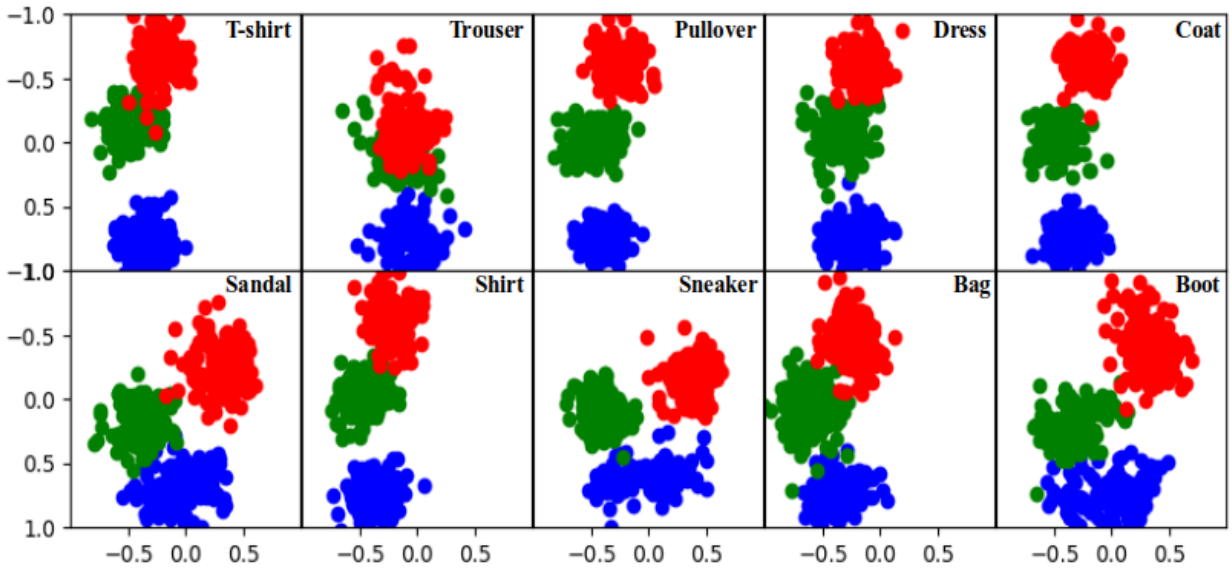


Figure 3.9: **Top-level Part Locations for Fashion-MNIST Examples by Class:** Red: first, Green: second and Blue: third location. Note the differences in top-level action strategies between vertically symmetric items (shirts, trousers, bags) and footwear (sandals, sneakers, boots).

	affNIST
APCN-2-Traditional	0.0110
APCN-2-Hypernet	**0.0105**

Table 3.2: **Hypernetworks vs Traditional Alternative:** Generating sub-programs for affNIST.

An alternative to using hypernetworks to generate the state and action sub-programs is using a traditional feedforward network and concatenating its output with the input to the bottom level RNN. Table 3.2 shows a comparison between these two approaches on the affNIST dataset. This result indicates that hypernetworks provide a clean abstraction for sub-program generation.

Prediction of Parts and Pattern Completion

To investigate the predictive and generative ability of the model, we had the model "hallucinate" different parts of an object by setting the prediction error input to the lower level network to zero for all or some macro-steps. This disconnects the model from the input, forcing it to predict the next sequence of parts and "complete" the object. Figure 3.10 shows that the model has learned to generate plausible predictions of parts given the initial glimpse (and any additional glimpses).

3.2.3 Transfer Learning

We tested transfer learning for reconstruction of unseen character classes for the Omniglot dataset. We trained an APCN model to reconstruct examples from a subset of classes from the Omniglot alphabets. For each alphabet, we used 85% of the classes for training. The rest of the classes were used to test transfer. Specifically, the trained model had to use its learned representations and programs (as generated by the hypernets) to compose and reconstruct

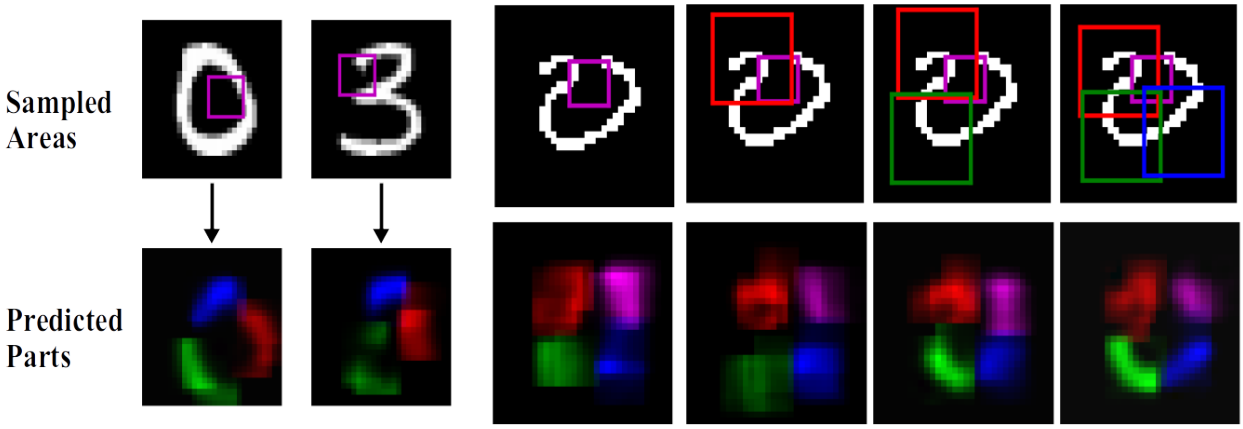


Figure 3.10: **Prediction of Parts and Pattern Completion by APCNs:** (Left two columns) Given only an initialization glimpse (purple box) for an input image (here, a 0 and a 3 from MNIST), the trained APCN predicts its best guess of the parts of the object and their locations (colored segments in row below). (Right columns) In other cases (here, an Omniglot character), the initial glimpse allows only a coarse prediction of parts. This prediction can be refined (bottom row) with additional glimpses (red, green, blue boxes).

new character classes for each alphabet. Table 3.1 shows the performance of the model on the transfer task. Figure 3.11 shows example hierarchical parsing strategies for characters from previously unseen classes, along with the reconstructions of these novel characters by the model.

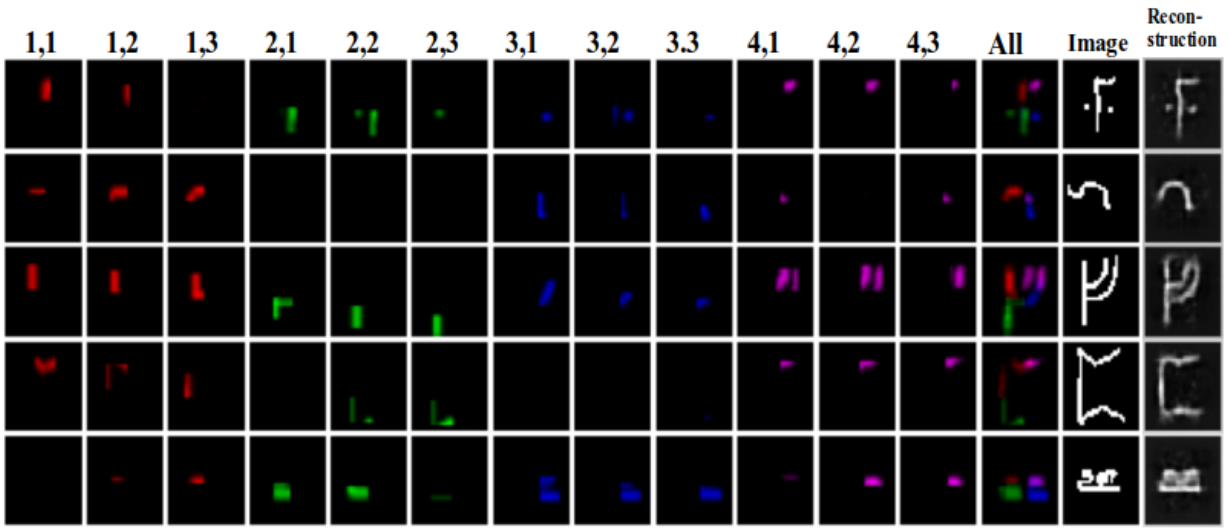


Figure 3.11: **Transfer Learning on Unseen Classes in Omniglot:** APCNs can transfer their learned knowledge to new, previously unseen classes of Omniglot objects. Each column corresponds to parts from bottom-level glimpses at time (t, τ). The "All" column shows all the parts together. The last two columns show the input from the novel class and its reconstruction by the APCN.

3.3 Some Limitations of the Model

For simple datasets such as MNIST, APCN-2 tends to converge to a general strategy that works well for all digits, resulting in little inter-class location diversity, while for other datasets such as Omniglot, a general strategy that "tests" a diverse set of image sub-areas might still be appropriate. For Fashion-MNIST, different strategies are learned for vertically symmetric clothing items versus non-symmetric ones such as footwear (Figure 3.9). These results can be attributed to our use of a general decoder to reconstruct the entire image based on current state within an attention-based encoder. An interesting future direction is to use a decoder where each macro-/micro- step renders a part of the object within a larger canvas used for computing the reconstruction error, as in [15].

Another limitation is the use of fixed time horizons for each option: our model parses each sub-area for a fixed number of steps whereas different sub-areas of an image might require fewer or more steps to process. Techniques such as the one used in [15] could be employed to address this limitation.

APCNs have yet to be applied to more challenging datasets (e.g., ImageNet) and other tasks (e.g., regression of object properties). Deeper versions of our model (with more than two levels) may be necessary for more complex image datasets.

Finally we used REINFORCE, which is inefficient compared to state-of-the-art reinforcement learning algorithms. The results could potentially be improved using more sophisticated policy gradient methods or designing novel methods tailored to the structure of the model.

3.4 Conclusion

APCNs are to my knowledge, the first hierarchical neural network capable of end-to-end learning and parsing of part-whole hierarchies from images. The proposed framework is highly flexible and offers a number of potential applications and future research directions. For example, actions in APCNs could include not just position but arbitrary transformations of parts, allowing the network to learn hierarchical equivariant representations, a long-sought

goal in machine vision and AI [30, 31]. More broadly, this framework offers a new approach to hierarchical reinforcement learning and planning in continuous state and action spaces. Finally, given the close connection between APCNs and predictive coding models of brain function, the proposed framework paves the way for a new interpretation of the hierarchical architecture of the cortex and a new role for cortical feedback connections in modulating the dynamics of lower-level networks [36] similar to the role played by hypernets in APCNs.

Chapter 4

HYPER-UNIVERSAL POLICY FUNCTION APPROXIMATORS

The American psychologist James Gibson first proposed the idea of *object affordances* [19], namely, that an object is perceived not only by the object's visual features, but also by the potential motor actions it affords. The idea that a single image of an object can suggest action plans to achieve particular goals has important implications for computer vision-based robotic agents: it can allow zero-shot generalization of learned skills to new objects and environments. Can such a capability be realized by computer vision systems?

In reinforcement learning, action plans are formalized in terms of a policy function $\pi(s) \rightarrow a$ that maps the current state s of the agent to the desired action a. Policy-based techniques, widely used in reinforcement learning [65], aim to find strategies that maximize the expected reward received from the environment. In most realistic scenarios, agents are required to be adept in many different tasks and good policies often exploit shared structure defined by task similarity and the dynamics of the environment. For example, in the case of navigation, policies that guide an agent to two different goals in close proximity should agree for most states s. Similarly, policies for different but closely-related environments should also be related. To extrapolate policies in novel environments, humans often rely on visual cues. In navigation tasks, humans can utilize a visual map to navigate in a previously unknown space.

To formalize this correspondence between visual (and potentially other) auxiliary information and action policies, I propose the concept of a universal policy function (UPF) $\pi_E(s, g)$, where g is the goal and E a description of the environment, obtained, for example, from a visual map. UPFs are related to general value functions [67, 59], but besides gener-

alizing value functions to novel goals, they additionally tackle the challenge of transferring policies to novel environments, a new dimension which introduces significant complexity.

Although UPFs can be approximated by any generic neural network, training such models based on a limited set of goal and environment samples can be challenging. An agent with limited computational capacity and communication bandwidth faces another challenge; storing or transmitting a monolithic model with lots of parameters can be unwieldy or even impossible. Such constraints are frequently encountered in designing edge devices.

In this chapter I will introduce Hyper-Universal Policy Approximators (HUPAs) [20] which, to my knowledge, is the first framework for mapping visual information, e.g., a single map image, to universal policy functions for agents with computational and communication constraints. HUPAs leverage the modularity properties of hypernetworks [18] to generate small environment-conditional, policy functions from an image (or other auxiliary information). These policy functions can then be easily transmitted to the agent and fine-tuned. I will compare HUPAs to the traditional embedding alternative and demonstrate a significant performance gap between the two approaches when the generated policy network is size-constrained. While the focus is on a simple map-based navigation task, these results are the first to highlight the advantage of hypernetworks in representing UPFs.

4.1 Universal Policy Approximation

In the context of a Markov Decision Process [64, 4] (MDP), a UPF is denoted by $\pi_E(s, g) \rightarrow a$, where g is the goal state, s the current state, a the chosen action, and E a context vector that serves as a description of the environment. This description contains any auxiliary information that can be useful in conditioning the resulting policy. It can be given explicitly (e.g., image of a map for map-based navigation, an embedding-based natural language description, etc.) or implicitly (inferred from local observations such as images from a robot's on-board camera). The goal is to efficiently generate policies π_E conditioned on a specific novel environment description E.

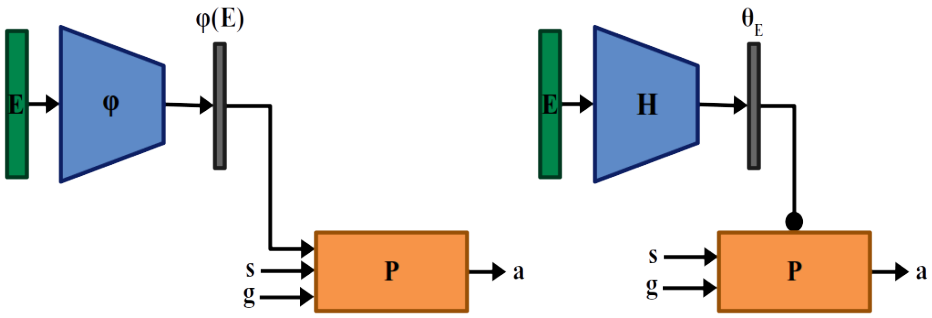
We make the following assumptions for our setting: **(a)** the agent/edge device is limited

in computational capacity and operates within a specific context/environment E, **(b)** there is device with large computational capacity hosted on the cloud that has access to context specific information that is useful to the agent and **(c)** the two are connected via a (potentially unreliable) communication channel with limited bandwidth.

A suggested solution would be to have the cloud device transmit the context information to the agent. The agent hosts a generic model which then takes the contextual information as input to condition it to the specific context E. This solution however is not feasible since the agent has limited computational capacity. Another potential solution would be to have the agent simply transmit its input s, g to the cloud device and receive an answer $\pi_E(s, g)$, removing the need for computation on the agent side. This solution might not be feasible since the communication channel between the two devices can be of limited bandwidth and not constantly available. We are particularly interested in zero-shot generation of suitable policies based on just the contextual information E.

A way to approximate π_E, respecting the constraints, is to use a large network ϕ hosted on the cloud to obtain a context embedding $\phi(E)$. This embedding is then transmitted to the edge device. The edge device hosts a size-limited network P that takes as input $x = [s, g, \phi(E)]$ and predicts the appropriate action for each x. We refer to this technique as the "embedding" approach. Note that instead of transmitting the whole embedding vector it suffices to transmit its projection to the first layer of P which is usually smaller in size. A diagram for the embedding approach is shown in Figure 4.1a.

An alternative proposition is that of Hyper-Universal Policy Approximators (HUPAs) [20], based on hypernetworks [27]. This static hypernetwork H (see Section 1.1.1), which is hosted on the cloud, takes as input the context vector and outputs a set of parameters $\theta_E = H(E)$. These parameters are then transmitted to the edge device and are used to parameterize a small primary network P. The HUPA model is shown in Figure 4.1b. Recent results on the modularity of hypernetworks [18] show that they are provably more efficient approximators of functions of this form compared to the embedding approach (see Section 1.1.3).



(a) Embedding Approach (b) HUPA

Figure 4.1: **Embedding versus Hypernet-based Approaches for Predicting Action Policies from a Single Environmental Input.**

4.1.1 Related Work

Hypernetworks have not been widely explored in the context of reinforcement learning. Perhaps the closest approach to HUPAs is the one introduced in [72]. The authors use a hypernetwork to generate networks that capture the transition dynamics of various control problems. These dynamics are used for model-based reinforcement learning through planning. The hypernetwork generates the transition dynamics based on observation/action pairs in the immediate past. The authors show that the approach can be used to handle changing conditions in the environment. In [16] the authors use hypernetworks in a model that takes a desired reward r as input and returns a policy function that approximately achieves that reward. In [33] the authors consider hypernetworks as a solution to continual model-based reinforcement learning. None of the above approaches consider the computational constraints of the agent or the decomposition of computation into remotely-hosted context and policy functions operating at the edge. Finally in [62] the authors use hypernetworks for personalized federated learning. Their setting resembles ours in the sense that the agents are separated from the hypernetwork. There is no consideration for computational constraints.

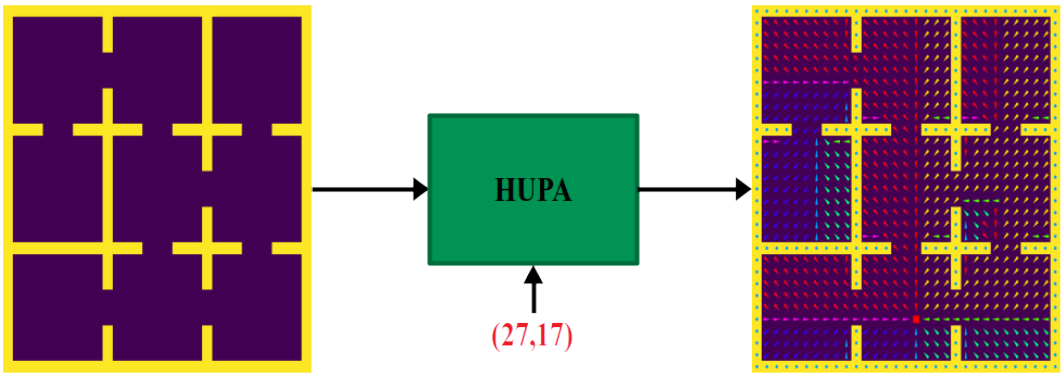
4.2 Evaluation

4.2.1 Dataset and Metrics

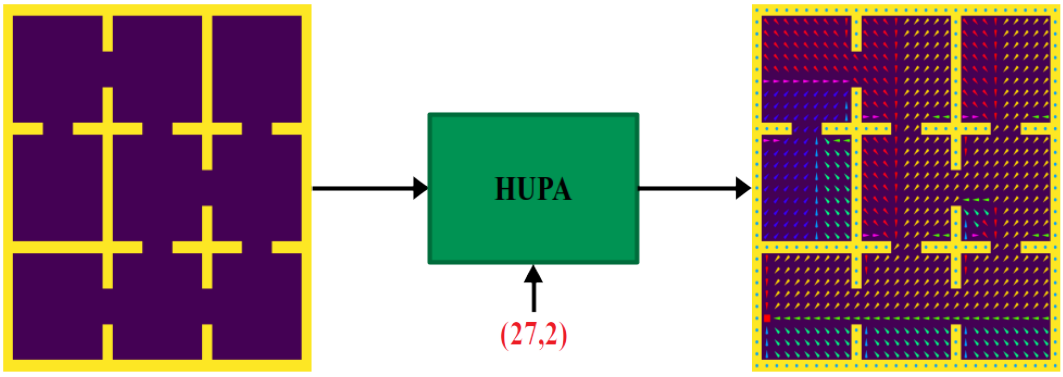
We evaluate HUPAs on a novel, artificially generated dataset based on a two-dimensional navigation task. The agent is given a map E, a starting coordinate s and must navigate to a goal g. The maps consist of discretized tiles which are either open or occupied by a wall. The action space of the agent consists of 8 actions/angles $a \in \{\frac{k\pi}{4} : k \in \{1, ..., 8\}\}$ that transfer the agent to one of the neighboring tiles (up, down, left, right and the diagonals) as long as they are open. Our goal is to learn a HUPA that, given previously unseen maps and goals, can generate policies that transfer the agent successfully to the target location. Since the action space is discrete, the generated primary network P takes the form of a classifier.

Map Generation

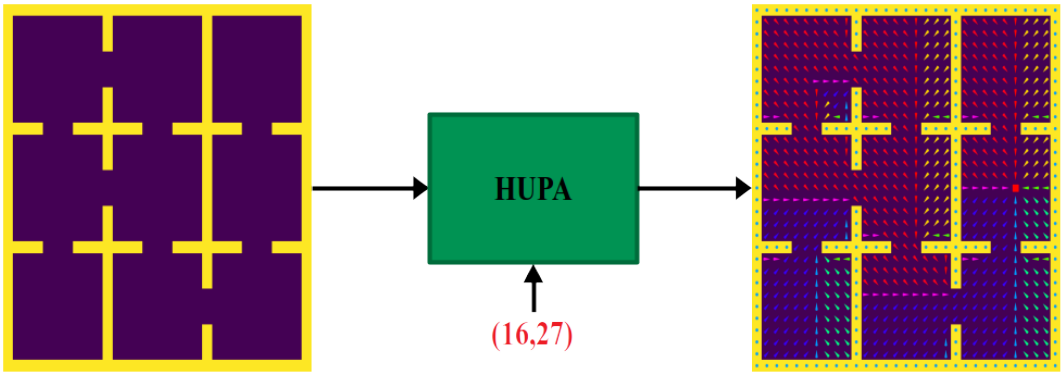
For our experiments we use a nine room generalization of the four-room environment commonly used in RL navigation scenarios. In this regime, each map E is derived from a base map that contains rooms arranged in a 3×3 grid separated by walls. Each room consists of a 9×9 grid of open tiles, and neighbouring rooms are connected by doors in the center of the separating wall. There are twelve doors in total. To generate a new map from the base map, three doors are chosen to be blocked off under the constraint that the space remains connected. This results in 164 potential maps. From each map all possible start and goal coordinates are sampled to create the dataset. By representing states s, g by their two-dimensional coordinates (x, y) the maps can be conveniently represented as images, as we do in the following experiments. HUPAs function as zero-shot policy generators for previously unseen environments based on these single images (see Figure 4.2).



(a) Known E - Known g



(b) Known E - Unknown g



(c) Unknown E - Unknown g

Figure 4.2: **Zero-Shot Learning of Policies from Single Images:** After training on known E and g samples, a HUPA **(a)** predicts policies for novel goals **(b)** and novel environments **(c)** from a single input map image. The goal is marked by a red square on the right-side policy maps; action vectors are color-coded by angle.

Reachability Ratio

While canonical accuracy serves as a good measure of how close our generated policies are to the ground truth policy, it does not encapsulate the usability of a policy. The successful transition of the agent to the goal state relies on a sequence of several correctly predicted actions, a requirement that is not captured by plain accuracy. As an example a generated policy can leave a whole room disconnected from the goal, whereas most of the state-action pairs inside the room are predicted correctly (for examples of such generated policies, see Figure 4.3d).

A natural way to properly assess the quality of a generated policy is to evaluate the fraction of states from which it successfully guides the agent to the goal. We call this the Reachability Ratio (RR) metric. To that end we construct an incidence graph where each state s is represented by a node n_s. For each such state we add an edge $n_s \to n_{\pi_E(s,g)}$ that corresponds to the immediate transition that results from following the policy at s. Let C_g be the connected component of this graph that contains n_g. Then C_g contains all the nodes from which the policy successfully reaches the goal. Hence the Reachability Ratio of a set T of test goals is the average fraction of nodes in the target set "accessible" to each state in the map:

$$\mathrm{RR}(T) = \frac{1}{|S|} \sum_{g \in S} \frac{|T \cap C_g|}{|T|} \tag{4.1}$$

where C_g is computed via the reachability graph of π_E.

4.2.2 Experiments

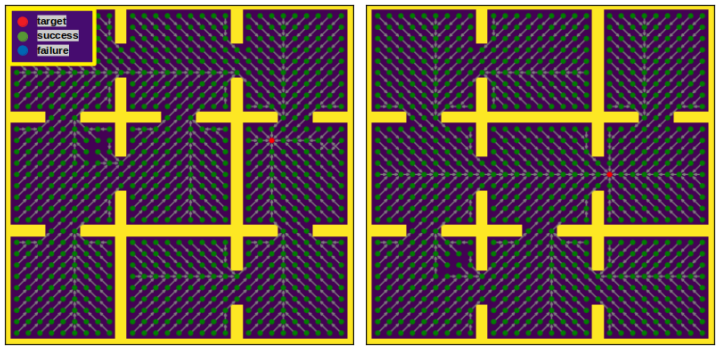
For the embedding function ϕ, we used a convolutional neural network with four residual blocks, followed by a fully connected bottleneck layer. An additional layer outputs the embedding vector $\phi(E)$. The hypernetwork had almost identical structure, except for an additional layer that generates the parameters θ_E. For a fair comparison we adjust the size of the embedding vector accordingly, while keeping the size of the primary network fixed, to approximately equate the total number of parameters.

The primary network consists of three hidden layers with the same number of neurons, followed by an output layer that predicts the movement direction. We compare the two approaches on primary networks with $16, 32, 64, 128$ and 256 neurons. The resulting models have $315K, 544K, 1.4M, 4.7M$ and $17.6M$ parameters respectively. We used 50 maps and 40% of the states at random as our training set. We used early stopping on a validation set of 5 novel maps and 10% novel goals, with patience equal to 10. For evaluation we used 20 test maps and the remaining goals.

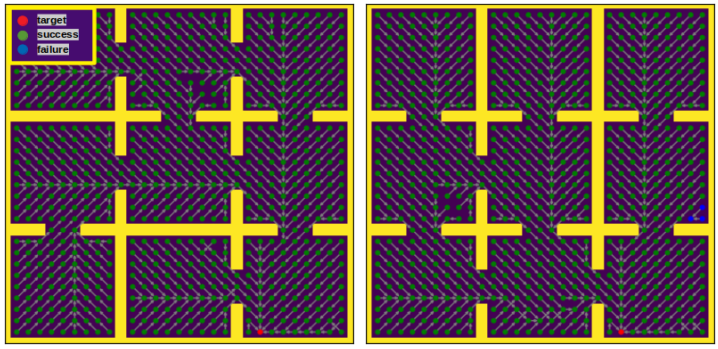
Figure 4.3 shows examples of generated policies represented by their reachability graphs. Figures 4.3a and 4.3b show high-quality policies for known maps and known/unknown goals. Figure 4.3c shows successful zero-shot policy generation for unknown maps and goals. Finally Figure 4.3d shows examples of policies that fail to correctly understand the structure of the map, leaving whole rooms disconnected from the goal. However, these rooms are disconnected due to only a few erroneous decisions. Fine-tuning the generated policies using reinforcement learning based on a few episodes could potentially correct these errors.

We quantitatively compared HUPAs to the embedding alternative both in terms of accuracy and reachability. As seen in Figure 4.4, HUPAs significantly outperform the baseline in both metrics. The effect is more pronounced for more constrained primary networks, with the embedding model approaching the performance of HUPAs as the parameter count becomes significantly higher.

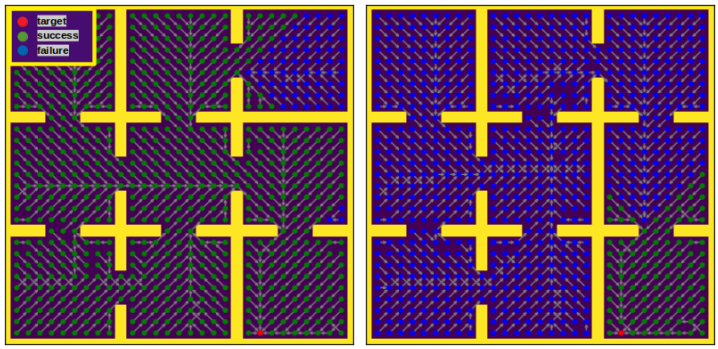
To evaluate the robustness of the HUPA approach, we varied the number of training maps and goal states. We chose the 128 neuron model for our primary architecture. As seen in Figure 4.5, the hypernetwork generalizes well even when the map/goal training set is sparse. HUPAs consistently outperform the embedding model in all sparsity settings.



(a) Known E - Known g (b) Known E - Unknown g



(c) High quality policies: Unknown E - Unknown g



(d) Low quality policies: Unknown E - Unknown g

Figure 4.3: **Reachability Graph Examples:** Examples of reachability graphs in policies predicted by the HUPA model for unseen maps and goals. Walls are denoted by yellow, open tiles by purple, while the red node marks the goal. Green nodes are states from which the goal is reachable under the predicted policy. Blue nodes do not reach the goal. The failure example in (d) (right) illustrates the importance of using reachability compared to accuracy.

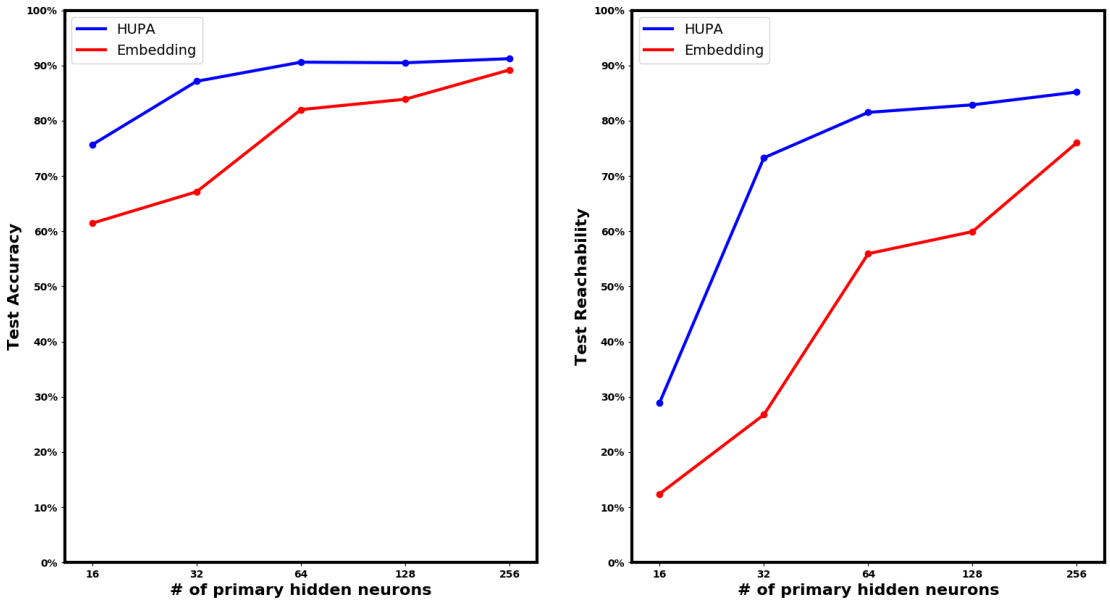


Figure 4.4: **Generalization to Novel Maps and Goals:** Performance of HUPAs on novel maps and goal compared to the embedding approach: policy accuracy **(left)**; goal reachability **(right)**.

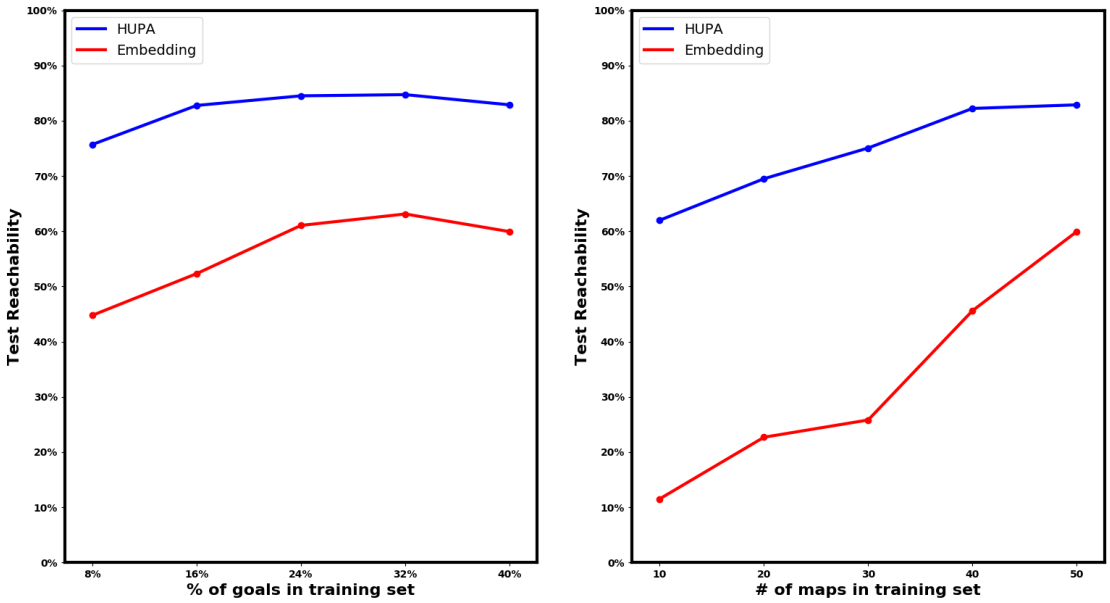


Figure 4.5: **Robustness to Map and Goal Training Data Sparsity:** Goal reachability as a function of percentage of total possible goals **(left)** and maps **(right)** used for training.

4.3 Future Research Directions

In this chapter I introduced HUPAs, a hypernetwork-based model for conditioning policy functions to contextual information. I compared HUPAs to the traditional, embedding approach on a simple map-based navigation task. The results shows that HUPAs achieve $> 80\%$ reachability for primary networks with ≥ 64 neurons per primary hidden layer, significantly outperforming the baseline. HUPAs are also more robust to sparse map/goal training data. These results indicate that HUPAs are a much more suitable approach for conditioning policy functions on context, especially in the presence of computational/communication constraints.

HUPAs provide a promising framework for computing hierarchical functions in general. In this chapter I treated UPFs as functions that contain two nested levels of abstraction ; there are various maps E and for each one of them various inputs s and g. An alternative would be to treat the same function as containing three nested levels by conditioning first on the map E, then on the goal g and finally on the current state s. Such functions of more than two layers of abstraction can be represented by nested hypernetworks [49]. Suppose we want to approximate a function $f_{E_2,E_1}(x)$ where for each E_2 there are many E_1 settings and for each one of those many potential inputs x. In the extended HUPA architecture for approximating such a function, a hyper-hypernetwork $H^{(2)}$ would generate a hypernetwork $H^{(1)}$ based on E_2, which would then generate the primary network f based on the context E_1 (see Figure 4.6). Example applications include modeling policy functions such as the ones in the original HUPA setting, sub-classing classifiers based on the hierarchy of labels and hierarchical generative models.

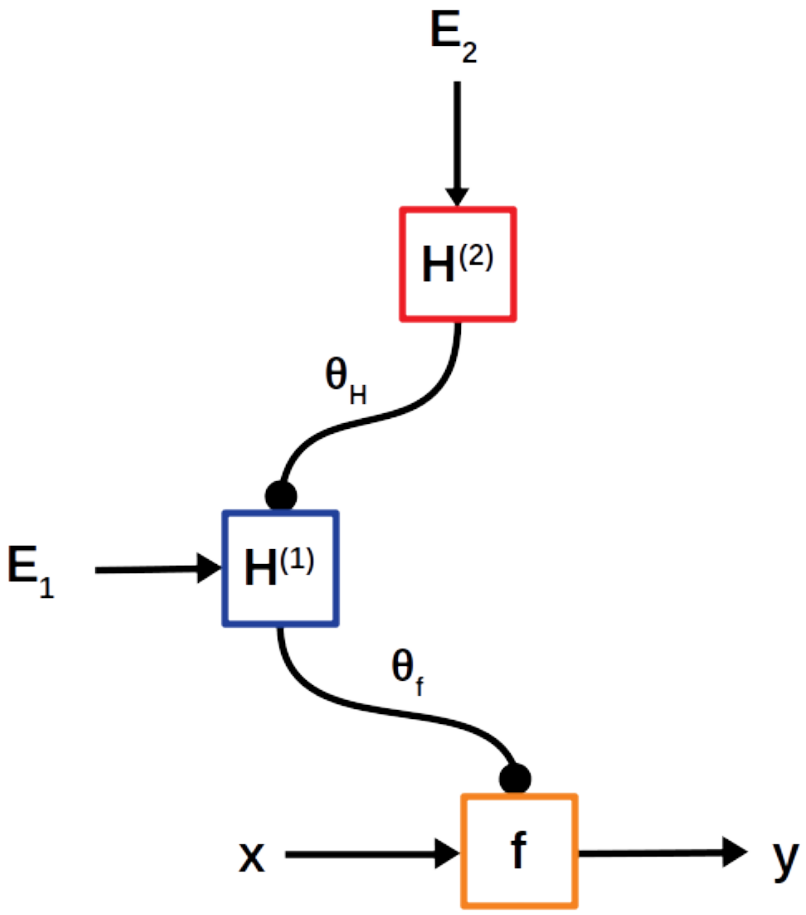


Figure 4.6: **General HUPA paradigm for successive conditioning:** Each conditioning operation is performed via a hypernetwork.

Chapter 5

CONCLUSIONS

In this thesis I investigated neural networks that generate other neural networks (hypernetworks). These models dynamically modify their own connections to adapt to inputs with different characteristics as opposed to traditional models that use a fixed set of parameters.

In Chapter 2 I presented Transformational Bilinear Sparse Coding (TBSC), a novel model based on bilinear sparse coding that can learn features and their transformations conjointly from unlabeled videos. We also introduced Dynamic Predictive Coding, a model that utilizes multiplicative interactions to allow for more expressive transition dynamics between sparse codes in video. TBSC uses a hypernetwork to allow feature groups to have independent pose parameters while still sharing a common transformation model. A TBSC layer represents its input as a set of feature groups, each with a sparse activation and a dense pose vector. These groups resemble the capsules in [58, 31].

In Chapter 3 I introduced Active Predictive Coding Networks (APCNs) a hypernetwork-based recurrent visual attention model. APCNs process images by sequentially obtaining small image samples (glimpses), rather than receiving the whole image in parallel. APCNs also split the task in a hierarchical manner by using neural sub-programs generated via hypernetworks. I applied APCNs to reconstructing MNIST and affNIST digits, Fashion-MNIST items and Omniglot characters. The results show that APCNs can learn intelligent sampling strategies and reconstruct the input effectively. Their predictive coding aspect allows them to predict unseen parts of an object.

In Chapter 4 I introduced Hyper-Universal Policy Approximators (HUPAs). HUPAs utilize hypernetworks to generate policies that are conditioned on contextual input. I focused on remote agents with limited computational capacity and access to a device on the cloud that

hosts the contextual input. My results show that the hypernetwork architecture efficiently generates navigation policies based on maps of different spaces, outperforming the embedding alternative.

My findings indicate that the dynamic connection capacity offered by hypernetworks can significantly enhance the capabilities of existing models (Chapter 2: Transformational Bilinear Sparse Coding), allow for new types of hierarchical models (Chapter 3: Active Predictive Coding Networks) and enable novel paradigms of computation for devices with restricted computational capacity (Chapter 4 Hyper-Universal Policy Approximators).

One future research direction that I find very interesting is to stack multiple TBSC layers into a hierarchy. The content and pose activations can be concatenated into different channels and propagated to the level above (see Figure 2.2) since the representation is no longer over-complete. Exploring the connections between the iterative inference used for the sparse content code and the Expectation-Maximization procedure used in [31] is also interesting.

Another promising future research direction would be to experiment with deeper APCNs and larger images with more complex structure or to train APCNs to answer "visual questions" about a scene [68]. Another idea is to apply more sophisticated reinforcement learning techniques to learning glimpse strategies instead of REINFORCE.

Finally a future research direction that I find most intriguing is to extend HUPAs to nested functions of more levels, where each level has its own contextual input and generates a hypernetwork for the layer below (see Figure 4.6). This successive conditioning of the approximated function on contextual information at different levels of abstraction strongly resembles object-oriented programming and could have interesting applications for Internet of Things (IoT) devices.

BIBLIOGRAPHY

[1] Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4338–4346, Red Hook, NY, USA, 2016. Curran Associates Inc.

[2] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *CoRR*, abs/1412.7755, 2014.

[3] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[4] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[5] Christopher M. Bishop. Mixture density networks. Technical report, 1994.

[6] Jack Culpepper David K Warland Bruno A. Olshausen, Charles Cadieu. Bilinear models of natural images, 2007.

[7] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[8] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[9] Yubei Chen, Dylan Paiton, and Bruno Olshausen. The sparse manifold transform. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[10] Peter C. Dodwell. The Lie transformation group model of visual perception. *Perception & Psychophysics*, 34(1):1–16, 1983.

[11] Dawei W. Dong and Joseph J. Atick. Temporal decorrelation: a theory of lagged and nonlagged responses in the lateral geniculate nucleus. *Network: Computation in Neural Systems*, 6(2):159–178, 1995.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[13] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the L1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 272–279, New York, NY, USA, 2008. Association for Computing Machinery.

[14] Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative models as distributions of functions. *CoRR*, abs/2102.04776, 2021.

[15] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E Hinton. Attend, Infer, Repeat: Fast scene understanding with generative models. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[16] Francesco Faccio, Vincent Herrmann, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber. Goal-conditioned generators of deep policies. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*, 2022.

[17] Karl Friston and Stefan Kiebel. Predictive coding under the free-energy principle. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 364:1211–21, 06 2009.

[18] T. Galanti and L. Wolf. On the modularity of hypernetworks. In *Advances in Neural Information Processing Systems*, volume 33, pages 10409–10419. Curran Associates, Inc., 2020.

[19] J. J. Gibson. *The senses considered as perceptual systems.* Houghton Mifflin, Boston, 1966.

[20] Dimitrios C. Gklezakos, Rishi Jha, and Rajesh P. N. Rao. Hyper-universal policy approximation: Learning to generate actions from a single image using hypernets. Neurovision Workshop CVPR 2022.

[21] Dimitrios C. Gklezakos and Rajesh P. N. Rao. Learning a convolutional bilinear sparse code for natural videos. In *NeurIPS 2019 Workshop Neuro AI: Real Neurons & Hidden Units: Future directions at the intersection of neuroscience and artificial intelligence*, 2019.

[22] Dimitrios C. Gklezakos and Rajesh P. N. Rao. Active Predictive Coding Networks: A neural solution to the problem of learning reference frames and part-whole hierarchies. *bioRxiv*, 2022.

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[24] Ross Goroshin, Michael F Mathieu, and Yann LeCun. Learning to linearize under uncertainty. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[25] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[26] David Grimes and Rajesh P. N. Rao. A bilinear model for sparse coding. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2003.

[27] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 630–645, Cham, 2016. Springer International Publishing.

[29] Felix Heide, Wolfgang Heidrich, and Gordon Wetzstein. Fast and flexible convolutional sparse coding. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5135–5143, 2015.

[30] Geoffrey E. Hinton. How to represent part-whole hierarchies in a neural network. *CoRR*, abs/2102.12627, 2021.

[31] Geoffrey E. Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[33] Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 799–805, 2021.

[34] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[35] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[36] Linxing Preston Jiang, Dimitrios C. Gklezakos, and Rajesh P. N. Rao. Dynamic predictive coding with hypernetworks. *bioRxiv*, 2021.

[37] Leslie P Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. Technical report, USA, 1996.

[38] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[39] Sylwester Klocek, Lukasz Maziarka, Maciej Wolczyk, Jacek Tabor, Jakub Nowak, and Marek Smieja. Hypernetwork functional image representation. In Igor V. Tetko, Vera Kurková, Pavel Karpov, and Fabian J. Theis, editors, *Artificial Neural Networks and Machine Learning - ICANN 2019 - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings - Workshop and Special Sessions*, volume 11731 of *Lecture Notes in Computer Science*, pages 496–510. Springer, 2019.

[40] Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[41] Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[42] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[44] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, December 2015.

[45] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

[46] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[47] Marcus Lewis, Scott Purdy, Subutai Ahmad, and Jeff Hawkins. Locations in the neocortex: A theory of sensorimotor object recognition using cortical grid cells. *Frontiers in Neural Circuits*, 13, 2019.

[48] Gidi Littwin and Lior Wolf. Deep meta functionals for shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[49] Shahar Lutati and Lior Wolf. Hyperhypernetworks for the design of antenna arrays. In *ICML*, 2021.

[50] Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[51] Xu Miao and Rajesh P. N. Rao. Learning the Lie Groups of Visual Invariance. *Neural Comput.*, 19(10):2665–2693, October 2007.

[52] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[53] Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997.

[54] Rajesh Rao and D Ballard. Development of localized oriented receptive fields by learning a translation-invariant code for natural images. *Network (Bristol, England)*, 9:219–34, 06 1998.

[55] Rajesh Rao and Dan Ruderman. Learning Lie groups for invariant visual perception. *Advances in Neural Information Processing Systems*, 11, 01 1999.

[56] Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, Jan 1999.

[57] Rajesh PN Rao. Decision making under uncertainty: a neural model based on partially observable markov decision processes. *Frontiers in computational neuroscience*, 4:146, 2010.

[58] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3859–3869, Red Hook, NY, USA, 2017. Curran Associates Inc.

[59] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR.

[60] J. Schmidhuber. A 'self-referential' weight matrix. In Stan Gielen and Bert Kappen, editors, *ICANN '93*, pages 446–450, London, 1993. Springer London.

[61] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[62] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. Personalized federated learning using hypernetworks. In *ICML*, 2021.

[63] Przemysław Spurek, Sebastian Winczowski, Jacek Tabor, Maciej Zamorski, Maciej Zieba, and Tomasz Trzcinski. Hypernetwork approach to generating point clouds. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9099–9108. PMLR, 13–18 Jul 2020.

[64] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[65] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

[66] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[67] Richard S. Sutton et al. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS*, pages 761–768. IFAAMAS, 2011.

[68] Benjamin W Tatler, Nicholas J Wade, Hoi Kwan, John M Findlay, and Boris M Velichkovsky. Yarbus, eye movements, and vision. *i-Perception*, 1(1):7–27, 2010. PMID: 23396904.

[69] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19667–19679. Curran Associates, Inc., 2020.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[71] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

[72] Zhou Xian, Shamit Lal, Hsiao-Yu Tung, Emmanouil Antonios Platanios, and Katerina Fragkiadaki. Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks. In *International Conference on Learning Representations*, 2021.

[73] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

[74] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

[75] Matthew D. Zeiler and Rob Fergus. Differentiable pooling for hierarchical feature learning. *CoRR*, abs/1207.0151, 2012.

[76] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[77] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *In CVPR*, 2010.

[78] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision*, pages 2018–2025, 2011.