UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title
Emergence of structured representations in neural network learning

Permalink
https://escholarship.org/uc/item/8wd6422b

Author
Cheung, Brian

Publication Date
2019

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Emergence of structured representations in neural network learning

by

Brian Cheung

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Vision Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Bruno Olshausen, Chair
Professor Jack Gallant
Professor Alexei Efros
Professor Jitendra Malik

Spring 2019

Emergence of structured representations in neural network learning

Copyright 2019
by
Brian Cheung

Abstract

Emergence of structured representations in neural network learning

by

Brian Cheung

Doctor of Philosophy in Vision Science

University of California, Berkeley

Professor Bruno Olshausen, Chair

Learning is one of the hallmarks of human intelligence. It marks a level of flexibility and adaptation to new information that no artificial model has achieved at this point. This remarkable ability to learn makes it possible to accomplish a multitude of cognitive tasks without requiring a multitude of information from any single task. In this thesis, we describe emergent phenomena which occur during learning for artificial neural network models. First, we observe how learning well-defined tasks can lead to the emergence of structured representations. This emergent structure appears at multiple levels within these models. From semantic factors of variation appearing in the hidden units of an autoencoder to physical structure appearing at the sensory input of an attention model, learning appears to influence all parts of a model. With this in mind, we develop a new method to guide this learning process for acquiring multiple tasks within a single model. Such methods will endow neural networks with greater flexibility to adapt to new environments without sacrificing the emergent structures which have been acquired previously from learning.

To all those who have given me the freedom to explore and play.

Contents

List of Figures

List of Tables

Acknowledgments

There are many things to be thankful for in my time as a graduate student. The chance to learn both academically and personally in an open and welcoming atmosphere has been a privilege. For this I would like to thank my advisor, Bruno Olshausen, for embracing any idea I presented with patience and encouragement. I would also like to thank my parents, Anita and Tak Cheung for supporting my choice to pursue academia which has kept me optimistic and motivated. Finally, thanks to all the friends who have made this a journey from which I will continue eagerly on to no end. From all those around me, I learned that my path forward has no boundaries.

Chapter 1

Learning in neural networks

Learning is a dynamic process. To distinguish it from other dynamics, learning implies a progressive improvement towards a goal. Learning algorithms involve integrating information from some external state and adapting internal variables to that information. In machine learning, this external state is referred to as data and the internal variables are referred to as a model. In some cases, this data will also be dependent on the model itself to form a feedback loop (e.g. active vision). In this context, the external state is commonly referred to as the environment. As shown in Figure 1.1, the model changes the environment through actions. Consequently, the boundary of where the data ends and where the model begins is be blurred in this feedback dynamic. To resolve this ambiguity, the internal variables are manually specified to be distinct from the external variables beforehand.

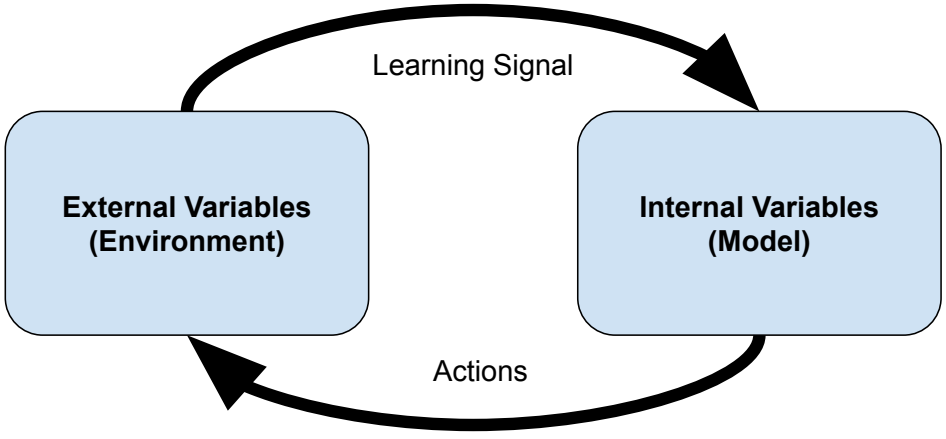

Figure 1.1: An abstract view of learning as an interaction between two distinct parts: an external environment and internal variables.

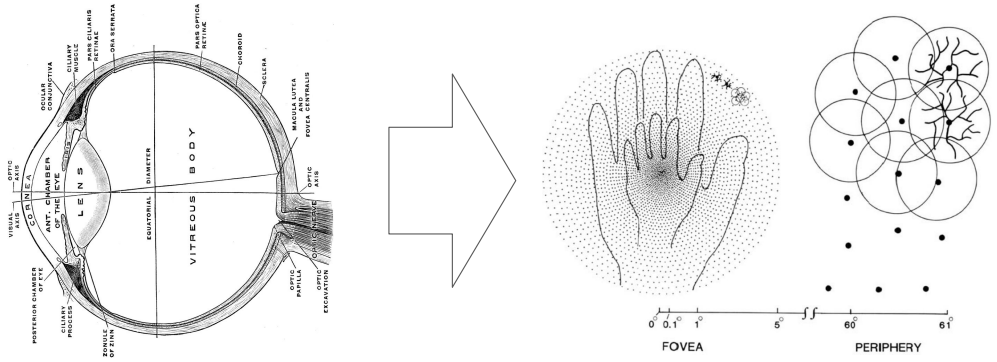

Figure 1.2: Anatomy of the visual sensory system for a primate. Left: Anatomy of the eye, Right: scale-invariant sampling lattice of retinal ganglion cells of the eye (left figure from [16], right figure from [65])

1.1 Learning Signal

While the external state can have many variables, the subset of variables which are relevant for the adaption of internal variables represents the learning signal. The learning signal can be anything that the adaptation depends upon to make adjustments to internal variables of a model. A significant portion of research in learning processes involves adaptation to achieve an externally defined objective or goal. This objective explicitly describes the dependency between the external and internal variables. Once an objective is specified, a wealth of knowledge from other fields of research such as optimization and control theory can be brought to bear to create learning algorithms which accomplish well-defined objectives.

Sensory Input

A sensory system lies at the boundary and serves as the interface between the external and internal state of a model. This defines the dependence structure of the model to the external state or environment. For any given model, this can also be referred to as an input. Figure 1.2 is a diagram of the sensory input of the primate visual system.

A commonly ignored part of the sensory input for a system which is learning with a predefined objective is the sensory input of the error or reward signal. In supervised learning, the sensory input is a label. In unsupervised learning, the sensory input is a self-supervised error like reconstruction. For gradient descent learning, the sensory input is the gradient of the objective. Regardless of the context, an objective-based learning system relies on both a sensory and an error input to adapt the internal variables of a model.

Representation Learning

In representation learning, the goal is to learn a transformation of the sensory input. This transformation maps the input into a representation space where an objective can be accom-

plished more easily than in the original input space. The definition of 'more easily' varies and often refers to the complexity required to manipulate the representation to accomplish high-level tasks. This field of learning encapsulates many ideas from cognitive science, neuroscience and machine learning. Models which focus on learning representations include neural network models like sparse coding, autoencoders and energy-based models.

1.2 Complementary Learning Systems

Adaptation can occur at a multiple levels and at different rates within a learning system. Some parts of the system may adapt quickly to the input while other parts will integrate information more slowly. For biological learning systems such as mammals, it is hypothesized that there are two complementary levels of learning. The hippocampus is believed to be a system which supports fast adaptation or learning while the neocortex more gradually integrates information [35, 44].

These levels of adaptation depend on the learning signal provided to the system. The learning signal for artificial systems typically comes from a set of data which is composed of individual elements (i.e. examples). As a set, there is no order to the elements and the order in which the elements are presented to the learning system is permutation invariant. Given this setup, there are only two levels of learning: the dataset level and the example level.

Internal states which adapt at the dataset level are considered parameters. Parameters represent information aggregated across all elements of the dataset. In contrast, variables which adapt at the example level are considered states of the learning system. For instance, in the K-means clustering algorithm, the states of the system are the cluster assignments for each element in the dataset and the parameters of the system are the cluster centers. The learning system which iteratively updates these two internal variables is known as expectation maximization.

In neural networks, individual element in the dataset are transformed into activations which are a representation of that element. These activations reflect the state of the model and represents the model's adaptation at the element level. In turn, these activations and the learning signal are used to generate updates to the weights and biases of the neural network which represents the model parameters. This update will change the activations of the model create a cycle. This update cycle is known as backpropagation or stochastic gradient descent.

1.3 Continual Learning

Datastreams

Previously, we described a distinction between the dataset and the examples which it is composed of. This distinction defines a boundary between information present at the distribution level and information at the sample level. But this boundary may be artificial. In

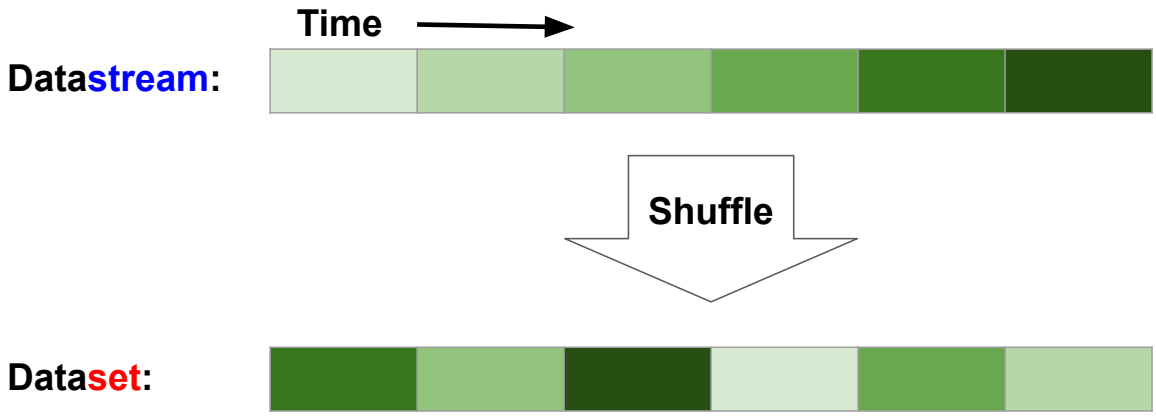

Figure 1.3: Through shuffling, a datastream of ordered elements is converted to a dataset where elements have lost the temporal ordering.

real world environments, data often has multiple scales of variation which make it difficult to form any consistent definition of what a sample is. For instance, images collected in a park in New York throughout the year can have drastic visual differences at both the scale of a single day and the scale of multiple months. Within one day, the visual scene transitions from day to night. Within one year, the visual scene transforms from summer to winter. When the original ordering of data is preserved, we call this a *datastream*.

Datastream to dataset

But the dominant assumption for a majority of learning systems is that samples in a dataset are independent and identically distributed. In Figure 1.3, a datastream is converted to a dataset by simply shuffling the elements in the stream at a predefined scale. In expectation, this form of sampling will reflect the distribution which the dataset was collected from. However, this is a strong assumption about the environment which is often taken for granted. Shuffling can result in a loss of information that may have been present in the original environment. Once data is shuffled, it can only live at the timescale of the shuffling. Any information at larger timescales is lost. This becomes a dilemma in current learning systems. Methods such as stochastic gradient descent rely on this assumption of statistical stationarity to properly minimize an objective function. In other words, the objective by design is invariant to the order of the elements.

Fields such as continual learning and online learning have begun to relax this assumption. Rather than assuming the objective and training distribution is fixed for all timepoints, continual learning objectives tend to be composed of multiple tasks presented to the learning system in a sequential order. Within each task, the learning signal is stationary, but not

Figure 1.4: Relaxing the datastream structure to a datastream of tasks.

necessarily stationary between tasks. A diagram of this setup is shown in Figure 1.4.

Chapter 2

Learning structure in hidden units

Deep learning has enjoyed a great deal of success because of its ability to learn useful features for tasks such as classification. But there has been less exploration in learning the factors of variation apart from the classification signal. By augmenting autoencoders with simple regularization terms during training, we demonstrate that standard deep architectures can discover and explicitly represent factors of variation beyond those relevant for categorization. We introduce a cross-covariance penalty (XCov) as a method to disentangle factors like handwriting style for digits and subject identity in faces. We demonstrate this on the MNIST handwritten digit database, the Toronto Faces Database (TFD) and the Multi-PIE dataset by generating manipulated instances of the data. Furthermore, we demonstrate these deep networks can extrapolate 'hidden' variation in the supervised signal.

One of the goals of representation learning is to find an efficient representation of input data that simplifies tasks such as object classification [34] or image restoration [8]. Supervised algorithms approach this problem by learning features which transform the data into a space where different classes are linearly separable. However this often comes at the cost of discarding other variations such as style or pose that may be important for more general tasks. On the other hand, unsupervised learning algorithms such as autoencoders seek efficient representations of the data such that the input can be fully reconstructed, implying that the latent representation preserves all factors of variation in the data. However, without some explicit means for factoring apart the different sources of variation the factors relevant for a specific task such as categorization will be entangled with other factors across the latent variables. Our goal in this work is to combine these two approaches to disentangle class-relevant signals from other factors of variation in the latent variables in a standard deep autoencoder.

Previous approaches to separating factors of variation in data, such as 'content' vs. 'style' [63] or 'form' vs. 'motion' [18, 48, 24, 45, 4, 5], have relied upon a bilinear model architecture in which the units representing different factors are combined multiplicatively. Such an approach was recently utilized to separate facial expression vs. identity using higher-order restricted Boltzmann machines [52]. One downside of bilinear approaches in general is that they require learning an approximate weight tensor corresponding to all three-way

multiplicative combinations of units. Despite the impressive results achieved with this approach, the question nevertheless remains as to whether there is a more straightforward way to separate factors of variation using standard nonlinearities in feedforward neural networks. Earlier work by [57] demonstrated class-irrelevant aspects in MNIST (style) can be learned by including additional unsupervised units alongside supervised ones in an autoencoder. However, their model does not disentangle class-irrelevant factors from class-relevant ones. More recently, [30] utilized a variational autoencoder in a semi-supervised learning paradigm which is capable of separating content and style in data. It is this work which is the inspiration for the simple training scheme presented here.

Autoencoder models have been shown to be useful for a variety of machine learning tasks [53, 66, 37]. The basic autoencoder architecture can be separated into an encoding stage and a decoding stage. During training, the two stages are jointly optimized to reconstruct the input data from the output of the decoder. In this work, we propose using both the encoding and decoding stages of the autoencoder to learn high-level representations of the factors of variation contained in the data. The high-level representation (or encoder output) is divided into two sets of variables. The first set (observed variables) is used in a discriminative task and during reconstruction. The second set (latent variables) is used only for reconstruction. To promote disentangling of representations in an autoencoder, we add two additional costs to the network. The first is a discriminative cost on the observed variables. The second is a novel cross-covariance penalty (XCov) between the observed and latent variables across a batch of data. This penalty prevents latent variables from encoding input variations due to class label. [54] proposed a similar penalty over terms in the product between the Jacobians of observed and latent variables with respect to the input. In our penalty, the variables which represent class assignment are separated from those which are encoding other factors of variations in the data.

We analyze characteristics of this learned representation on three image datasets. In the absence of standard benchmark task for evaluating disentangling performance, our evaluation here is based on examining qualitatively what factors of variation are discovered for different datasets. In the case of MNIST, the learned factors correspond to style such as slant and size. In the case TFD the factors correspond to identity, and in the case of Multi-PIE identity specific attributes such as clothing, skin tone, and hair style.

2.1 Semi-supervised Autoencoder

Given an input $\mathbf{x} \in \mathbb{R}^D$ and its corresponding class label $\mathbf{y} \in \mathbb{R}^L$ for a dataset $\mathcal{D}$, we consider the class label to be a high-level representation of its corresponding input. However, this representation is usually not invertible because it discards much of the variation contained in the input distribution. In order to properly reconstruct $\mathbf{x}$, autoencoders must learn a latent representation which preserves all input variations in the dataset.

Using class labels, we incorporate supervised learning to a subset of these latent variables transforming them into observed variables, $\hat{\mathbf{y}}$ as shown in Figure 2.1. In this framework, the

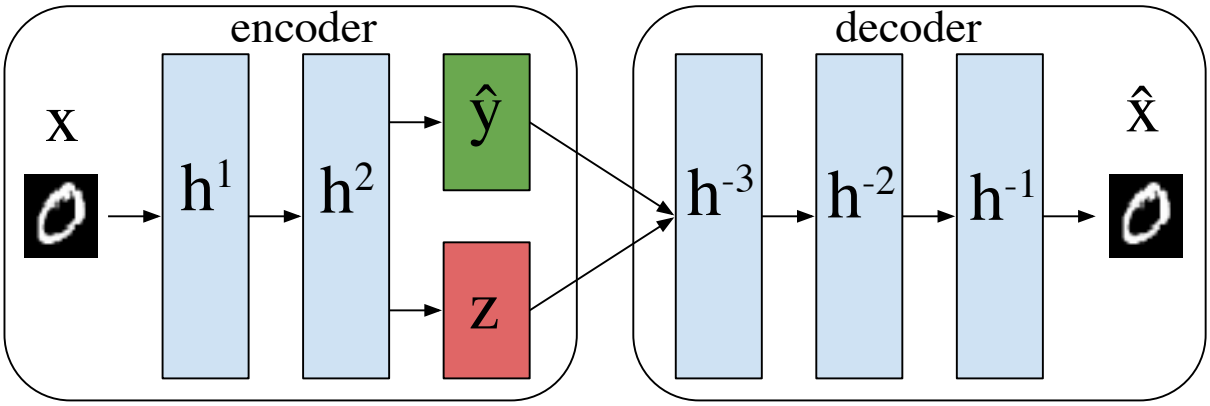

Figure 2.1: The encoder and decoder are combined and jointly trained to reconstruct the inputs and predict the observed variables $\hat{\mathbf{y}}$.

remaining the latent variable $\mathbf{z}$ must account for the remaining variation of dataset $\mathcal{D}$. We hypothesize this latent variation is a high-level representation of the input complementary to the observed variation. For instance, the class label '5' provided by $\mathbf{y}$ would not be sufficient for the decoder to properly reconstruct the image of a particular '5'. In this scenario, $\mathbf{z}$ would encode properties of the digit such as style, slant, width, etc. to provide the decoder sufficient information to reconstruct the original input image. Mathematically, the encoder F and decoder G are defined respectively as:

$$\{\hat{\mathbf{y}}, \mathbf{z}\} = F(\mathbf{x}; \theta) \tag{2.1}$$

$$\hat{\mathbf{x}} = G(\mathbf{y}, \mathbf{z}; \phi) \tag{2.2}$$

where θ and ϕ are the parameters of the encoder and decoder respectively.

Learning

The objective function to train the network is defined as the sume of three seperate cost terms.

$$\hat{\theta}, \hat{\phi} = \arg\min_{\theta,\phi} \sum_{\{\mathbf{x},\mathbf{y}\}\in\mathcal{D}} ||\mathbf{x} - \hat{\mathbf{x}}||^2 + \beta \sum_i y_i log(\hat{y}_i) + \gamma C. \tag{2.3}$$

The first term is a typical reconstruction cost (squared error) for an autoencoder. The second term is a standard supervised cost (cross-entropy). While there are many potential choices for the reconstruction cost depending on the distribution of data vector x, for our experiments we use squared-error for all datasets. For the observed variables, the form of the cost function depends on the type of variables (categorical, binary, continuous). For

our experiments, we had categorical observed variables so we parametrized them as one-hot vectors and compute $\hat{\mathbf{y}} = softmax(\mathbf{W}_{\hat{\mathbf{y}}}\mathbf{h^2} + \mathbf{b}_{\hat{\mathbf{y}}})$.

The third term C is the unsupervised cross-covariance (XCov) cost which disentangles the observed and latent variables of the encoder.

$$C(\hat{\mathbf{y}}^{1\ldots N}, \mathbf{z}^{1\ldots N}) = \frac{1}{2}\sum_{ij}[\frac{1}{N}\sum_{n}(\hat{y}_i^n - \bar{\hat{y}}_i)(z_j^n - \bar{\hat{z}}_j)]^2. \tag{2.4}$$

The XCov penalty to disentangle $\hat{\mathbf{y}}$ and $\mathbf{z}$ is simply a sum-squared cross-covariance penalty between the activations across samples in a batch of size N where $\bar{\hat{y}}_i$ and $\bar{\hat{z}}_j$ denote means over examples. n is an index over examples and i, j index feature dimensions. Unlike the reconstruction and supervised terms in the objective, XCov is a cost computed over a batch of datapoints. It is possible to approximate this quantity with a moving average during training but we have found that this cost has been robust to small batch sizes and have not found any issues when training with mini-batches as small as $N = 50$. Its derivative is provided in the supplementary material.

This objective function naturally fits a semi-supervised learning framework. For unlabeled data, the multiplier β for the supervised cost is simply set to zero. In general, the choice of β and γ will depend on the intended task. Larger β will lead to better to classification performance while larger γ to better separation between latent and observed factors.

2.2 Experimental Results

We evaluate autoencoders trained to minimize 2.3 on three datasets of increasing complexity. The network is trained using ADADELTA [71] with gradients from standard backpropagation. Models were implemented in a modified version of Pylearn2 [13] using deconvolution and likelihood estimation code from [14].

Datasets

MNIST Handwritten Digits Database

The MNIST handwritten digits database [38] consists of 60,000 training and 10,000 test images of handwritten digits 0-9 of size 28x28. Following previous work [12], we split the training set into 50,000 samples for training and 10,000 samples as a validation set for model selection.

Toronto Faces Database

The Toronto Faces Database [61] consists of 102,236 grayscale face images of size 48x48. Of these, 4,178 are labeled with 1 of 7 different expressions (anger, disgust, fear, happy, sad, surprise, and neutral). Examples are shown in Figure 2.2. The dataset also contains 3,784 identity labels which were not used in this paper. The dataset has 5 folds of training,

validation and test examples. The three partitions are disjoint and contain no overlap in identities.

Figure 2.2: Left: Example TFD images from the test set showing 7 expressions with random identity. Right: Example Multi-PIE images from the test set showing 3 of the 19 camera poses with variable lighting and identity.

Multi-PIE Dataset

The Multi-PIE datasets [19] consists of 754,200 high-resolution color images of 337 subjects. Each subject was recorded under 15 camera poses: 13 spaced at 15 degree intervals at head height, and 2 positioned above the subject. For each of these cameras, subjects were imaged under 19 illumination conditions and a variety of facial expressions. We discarded images from the two overhead cameras due to inconsistencies found in their image. Camera pose and illumination data was retained as supervised labels.

Only a small subset of the images possess facial keypoint information for each camera pose. To perform a weak registration to appoximately localize the face region, we compute the maximum bounding box created by all available facial keypoint coordinates for a given camera pose. This bounding box is applied to all images for that camera pose. We then resized the cropped images to 48x48 pixels and convert to grayscale. We divide the dataset into 528,060 training, 65,000 validation and 60,580 test examples. Splits were determined by subject id. Therefore, the test set contains no overlap in identities with the training or validation sets. Example images from our test set are shown in Figure 2.2.

The Multi-PIE dataset contains significantly more complex factors of variation than MNIST or TFD. Unlike TFD, images in Multi-PIE includes much more of the subject's body. The weak registration also causes significant variation in the subject's head position and scale.

Model Performace

Classification

As a sanity check, we first show that the our additional regularization term in the cost negligibly impacts performance for different architectures including convolution, maxout and dropout. Tables 2.2 and 2.3 show classification results for MNIST and TFD are comparable

Table 2.1: Network Architectures (Softmax (SM), Rectified Linear (ReLU))

MNIST	**TFD**	**ConvDeconvMultiPIE**
500 ReLU	2000 ReLU	20x20x32 ConvReLU
500 ReLU	2000 ReLU	2000 ReLU
10 SM, 2 Linear	7 SM, 793 Linear	2000 ReLU
500 ReLU	2000 ReLU	13 SM, 19 SM, 793 Linear
500 ReLU	2000 ReLU	2000 ReLU
784 Linear	2304 Linear	2000 ReLU
		2000 ReLU
		2000 ReLU
		48x48x1 Deconv

Table 2.2: MNIST Classification Performance

Model	**Accuracy**	**Model Selection Criterion**
MNIST	98.35	Reconstuction: $\beta = 10, \gamma = 10$
ConvMNIST	98.71	Reconstuction: $\beta = 10, \gamma = 10$
MaxoutMNIST + dropout	99.01	Accuracy: $\beta = 100, \gamma = 10$
Maxout + dropout [12]	99.06	Accuracy

to previously published results. Details on network architecture for these models can be found in the supplementary material.

Learned Factors of Variation

We begin our analysis using the MNIST dataset. We intentionally choose $\mathbf{z} \in \mathbb{R}^2$ for the architecture described in Table 2.1 for ease in visualization of the latent variables. As shown in Figure 2.3a, $\mathbf{z}$ takes on a suprisingly simple isotropic Normal distribution with mean 0 and standard deviation .35.

Visualizing Latent Variables

To visualize the transformations that the latent variables are learning, the decoder can be used to create images for different values of $\mathbf{z}$. We vary a single element z_i linearly over a set interval with $z_{\setminus i}$ fixed to 0 and $\mathbf{y}$ fixed to one-hot vectors corresponding to each class label as shown in Figure 2.3b and c. Moving across each column for a given row, the digit style is maintained as the class labels varies. This suggests the network has learned a class invariant latent representation. At the center of $\mathbf{z}$-space, (0,0), we find the canonical MNIST

Table 2.3: TFD Classification Performance

Model	Accuracy	Model Selection Criterion
TFD	69.4	Reconstuction: $\beta = 10, \gamma = 1e3$ (Fold 0)
ConvTFD	84.0	Accuracy: $\beta = 10, \gamma = 1e3$ (Fold 0)
disBM [52]	85.4	Accuracy
CCNET+CDA+SVM [54]	85.0	Accuracy (Fold 0)

digit style. Moving away from the center, the digits become more stylized but also less probable. We find this result is reliably reproduced without the XCov regularization when the dimensionality of $\mathbf{z}$ is relatively small suggesting that the network naturally prefers such a latent representation for factors of variation absent in the supervised signal. With this knowledge, we describe a method to generate samples from such an autoencoder with competative generative performance in the supplementary material.

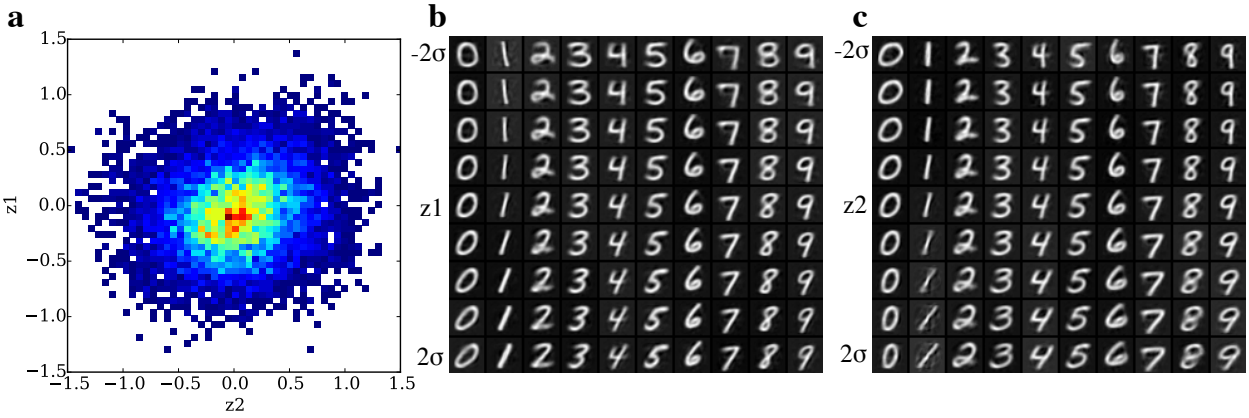

Figure 2.3: **a**: Histogram of test set z variables. **b**: Generated MNIST digits formed by setting z_2 to zero and varying z_1. **c**: Generated MNIST digits formed by setting z_1 to zero and varying z_2. σ was calculated from the variation on the test set.

Moving From Latent Space to Image Space

Following the layer of observed and latent variables $\{\mathbf{y}, \mathbf{z}\}$, there are two additional layers of activations $\mathbf{h^{-3}}, \mathbf{h^{-2}}$ before the output of the model into image space. To visualize the function of these layers, we compute the Jacobian of the output image, $\hat{\mathbf{x}}$, with respect to the activation of hidden units, $\mathbf{h^k}$, in a particular layer. This analysis provides insight into the transformation each unit is applying to the input $\mathbf{x}$ to generate $\hat{\mathbf{x}}$. More specifically, it is a measure of how a small perturbation of a particular unit in the network affects the output

$\hat{\mathbf{x}}$:

$$\Delta \hat{x}_i^k = \frac{\partial \hat{x}_i}{\partial h_j^k} \Delta h_j. \tag{2.5}$$

Here, i is the index of a pixel in the output of the network, j is the index of a hidden unit, and k is the layer number. We remove hidden units with zero activation from the Jacobian since their derivatives are not meaningful. A summary of the results are plotted in Figure 2.4.

The Jacobian with respect to the $\mathbf{z}$ units shown in Figure 2.4b locally mirror the transformations seen in Figure 2.3b and c further confirming the hypothesis that this latent space smoothly controls digit style. The slanted style generated as z_2 approaches 2σ in Figure 2.3c is created by applying a gabor-like filter to vertically oriented parts of the digit as shown in the second column of Figure 2.4b.

Rather than viewing each unit in the next layer individually, we analyze the singular value spectrum of the Jacobian. For $\mathbf{h^{-3}}$, the spectrum is peaked and thus there are a small number of directions with large effect on the image output, so we plot singular vectors with largest singular value. For all digits besides '1', the first component seems to create a template digit and the other componets make small style adjustments. For $\mathbf{h^{-2}}$, the spectrum is more degenerate, so we choose a random set of columns from the Jacobian to plot which will better represent the layer's function. We notice that for each layer moving from the encoder to the output, their contributions become more spatially localized and less semantically meaningful.

Generating Expression Transformations

We demonstrate similar manipulations on the TFD dataset which contains substantially more complex images than MNIST and has far fewer labeled examples. After training, we find the latent representation $\mathbf{z}$ encodes the subject's identity, a major factor of variation in the dataset which is not represented by the expression labels. The autoencoder is able to change the expression while preserving identity of faces never before seen by the model. We first initialize $\{\hat{\mathbf{y}}, \mathbf{z}\}$ with an example from the test set. We then replace $\hat{\mathbf{y}}$ with a new expression label $\hat{\mathbf{y}}'$ feeding $\{\hat{\mathbf{y}}', \mathbf{z}\}$ to the decoder. Figure 2.5 shows the results of this process. Expressions can be changed while leaving other facial features largely intact. Similar to the MNIST dataset, we find the XCov penalty is not necessary when the dimensionality of $\mathbf{z}$ is low ($<$10). But convergence during training becomes far more difficult with such a bottleneck. We achieve much better reconstruction error with the XCov penalty and a high-dimensional $\mathbf{z}$. The XCov penalty simply prevents expression label variation from 'leaking' into the latent representation. Figure 2.5 shows the decoder is unaffected by changes in $\mathbf{y}$ without the XCov penalty because the expression variation is distributed across the hundreds of dimensions of $\mathbf{z}$.

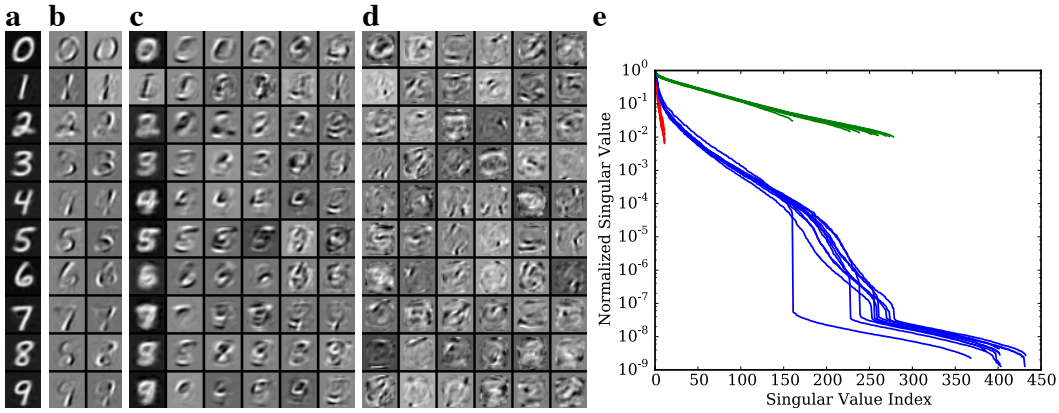

Figure 2.4: **a**: Jacobians were taken at activation values that lead to these images. **z** was set to zero for each digit class. **b**: Gradients of the decoder output with respect to **z**. **c**: Singular vectors from the Jacobian from the activations of the first layer after {**y**, **z**}. **d**: Column vectors from the Jacobian from the the activations of the second layer after {**y**, **z**}. Note that units in the columns for **d** are not neccesarily the same unit. **e**: Plots of the normalized singular values for *red*: the Jacobians with respect to {**y**, **z**}, *blue*: the Jacobians with respect to the activations of the first layer after {**y**, **z**} ($\mathbf{h^{-3}}$ in Figure 2.1), and *green*: the Jacobians with respect to the activations of the second layer after {**y**, **z**} ($\mathbf{h^{-2}}$ in Figure 2.1).

Extrapolating Observed Variables

Previously, we showed the autoencoder learns a smooth continuous latent representation. We find a similar result for the observed expression variables despite only being provided their discrete class labels. In Figure 2.6, we go a step further. We try values for $\hat{\mathbf{y}}$ well beyond those that the encoder could ever output with a softmax activation (0 to 1). We vary the expression variable given to the decoder from -5 to 5. This results in greatly exagerated expressions when set to extreme positive values as seen in Figure 2.6. Remarkably, setting the variables to extreme negative values results in 'negative' facial expressions being displayed. These negative facial expressions are abstract opposites of their positive counterparts. When the eyes are open in one extreme, they are closed in the opposite extreme. This is consistent regardless of the expression label and holds true for other abstract facial features such as open/closed mouth and smiling/frowning face. The decoder has learned a meaningful extrapolation of facial expression structure not explicitly present in the labeled data, creating a smooth semantically sensible space for values of the observed variables completely absent from the class labels.

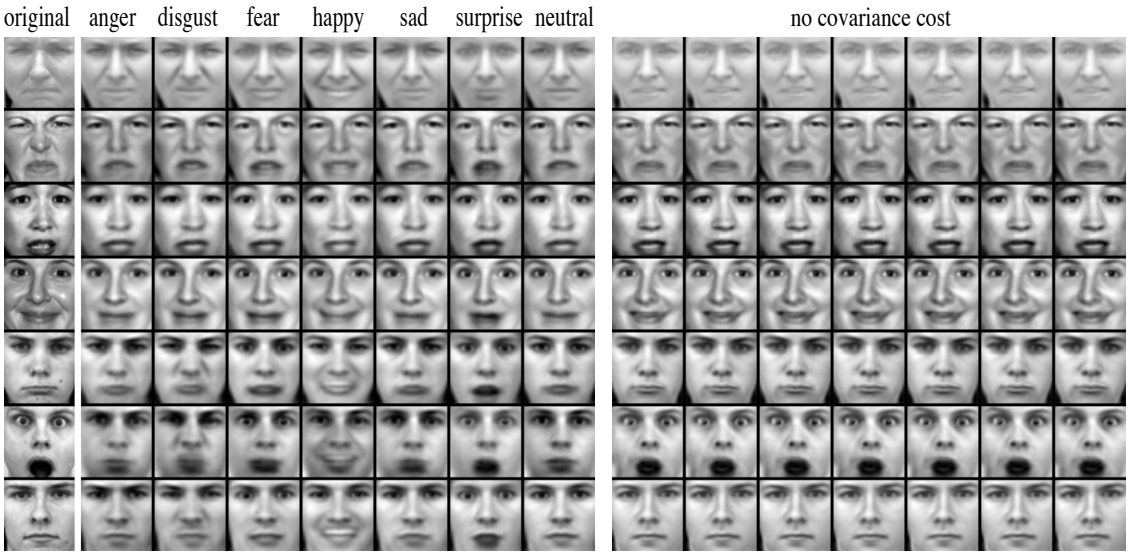

Figure 2.5: Left column: Samples from the test set displaying each of the 7 expressions. The expression-labeled columns are generated by keeping the latent variables $\mathbf{z}$ constant and changing $\mathbf{y}$ (expression). The rightmost set of faces are from a model with no covarriance cost and showcase the importance of the cost in disentangling expression from the latent $\mathbf{z}$ variables.

Manipulating Multiple Factors of Variation

For Multi-PIE, we use two sets of observed factors (camera pose and illumination). As shown in Table 2.1, we have two softmax layers at the end of the encoder. The first encodes the camera pose of the input image and the second the illumination condition. Due to the increased complexity of these images, we made this network substantially deeper (9 layers).

In Figure 2.7, we show the images generated by the decoder while iterating through each camera pose. The network was tied to the illumination and latent variables of images from the test set. Although blurry, the generated images preserve the subject's illumination and identity (i.e. shirt color, hair style, skin tone) as the camera pose changes. In Figure 2.8, we instead fix the camera position and iterate through different illumination conditions. We also find it possible to interpolate between camera and lighting positions by simply linearly interpolating the $\hat{\mathbf{y}}$ between two neighboring camera positions supporting the inherent continuity in the class labels.

2.3 Conclusion

With the addition of a supervised cost and an unsupervised cross-covariance penalty, an autoencoder can learn to disentangle various transformations using standard feedforward

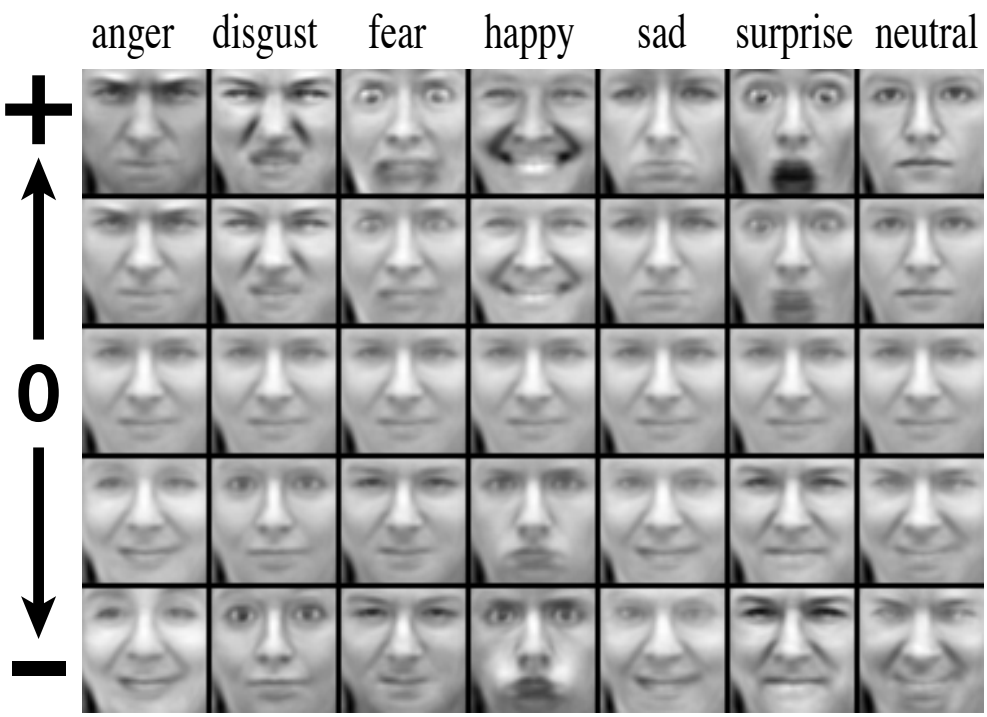

Figure 2.6: For each column, **y** is set to a one-hot vector and scaled from 5 to -5 from top to bottom, well outside of the natural range of [0,1]. 'Opposite' expressions and more extreme expressions can be made.

neural network components. The decoder implicitly learns to generate novel manipulations of images on multiple sets of transformation variables. We show deep feedforward networks are capable of learning higher-order factors of variation beyond the observed labels without the need to explicitly define these higher-order interactions. Finally, we demonstrate the natural ability of these deep networks to learn a continuum of higher-order factors of variation in both the latent and observed variables. Surprisingly, these networks can extrapolate intrinsic continuous variation hidden in discrete class labels. These results gives insight in the potential of deep learning for the discovery of hidden factors of variation simply by accounting for known variation. This has many potential applications in exploratory data analysis and signal denoising.

Figure 2.7: Left column: Samples from test set with initial camera pose. The faces on the right were generated by changing the corresponding camera pose.

Figure 2.8: Left column: Samples from test set. Illumination transformations are shown to the right. Ground truth lighting for the first face in each block is in the first row.

Chapter 3

Learning sensory input structure

We describe a neural attention model with a learnable retinal sampling lattice. The model is trained on a visual search task requiring the classification of an object embedded in a visual scene amidst background distractors using the smallest number of fixations. We explore the tiling properties that emerge in the model's retinal sampling lattice after training. Specifically, we show that this lattice resembles the eccentricity dependent sampling lattice of the primate retina, with a high resolution region in the fovea surrounded by a low resolution periphery. Furthermore, we find conditions where these emergent properties are amplified or eliminated providing clues to their function.

A striking design feature of the primate retina is the manner in which images are spatially sampled by retinal ganglion cells. Sample spacing and receptive fields are smallest in the fovea and then increase linearly with eccentricity, as shown in Figure 3.1. Thus, we have highest spatial resolution at the center of fixation and lowest resolution in the periphery, with a gradual fall-off in resolution as one proceeds from the center to periphery. The question we attempt to address here is, *why* is the retina designed in this manner - i.e., how is it beneficial to vision?

The commonly accepted explanation for this eccentricity dependent sampling is that it provides us with both high resolution and broad coverage of the visual field with a limited amount of neural resources. The human retina contains 1.5 million ganglion cells, whose axons form the sole output of the retina. These essentially constitute about 300,000 distinct 'samples' of the image due to the multiplicity of cell types coding different aspects such as 'on' vs. 'off' channels [65]. If these were packed uniformly at highest resolution (120 samples/deg, the Nyquist-dictated sampling rate corresponding to the spatial-frequencies admitted by the lens), they would subtend an image area spanning just 5x5 deg^2. Thus we would have high-resolution but essentially tunnel vision. Alternatively if they were spread out uniformly over the entire monocular visual field spanning roughly 150 deg^2 we would have wide field of coverage but with very blurry vision, with each sample subtending 0.25 deg (which would make even the largest letters on a Snellen eye chart illegible). Thus, the primate solution makes intuitive sense as a way to achieve the best of both of these worlds. However we are still lacking a quantitative demonstration that such a sampling strategy

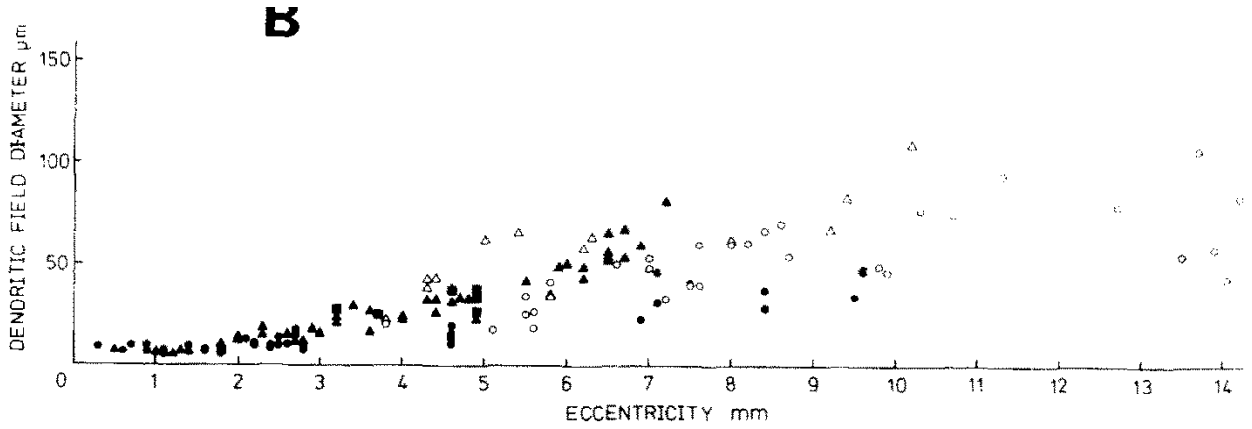

Figure 3.1: Receptive field size (dendritic field diameter) as a function of eccentricity of Retinal Ganglion Cells from a macaque monkey (taken from Perry, Oehler, and Cowey [49]).

emerges as the optimal design for subserving some set of visual tasks.

Here, we explore what is the optimal retinal sampling lattice for an (overt) attentional system performing a simple visual search task requiring the classification of an object. We propose a learnable retinal sampling lattice to explore what properties are best suited for this task. While evolutionary pressure has tuned the retinal configurations found in the primate retina, we instead utilize gradient descent optimization for our in-silico model by constructing a fully differentiable dynamically controlled model of attention.

Our choice of visual search task follows a paradigm widely used in the study of overt attention in humans and other primates [10]. In many forms of this task, a single target is randomly located on a display among distractor objects. The goal of the subject is to find the target as rapidly as possible. Itti and Koch [26] propose a selection mechanism based on manually defined low level features of real images to locate various search targets. Here the neural network must learn what features are most informative for directing attention.

While 'neural attention' models have been applied successfully to a variety of engineering applications [2, 28, 70, 15], there has been little work in relating the properties of these attention mechanisms back to biological vision. An important property which distinguishes neural networks from most other neurobiological models is their ability to learn *internal* (latent) features directly from data.

But existing neural network models specify the input sampling lattice *a priori*. Larochelle and Hinton [36] employ an eccentricity dependent sampling lattice mimicking the primate retina, and Mnih, Heess, Graves, et al. [47] utilize a multi scale 'glimpse window' that forms a piece-wise approximation of this scheme. While it seems reasonable to think that these design choices contribute to the good performance of these systems, it remains to be seen if this arrangement emerges as the optimal solution.

We further extend the learning paradigm of neural networks to the *structural* features of the glimpse mechanism of an attention model. To explore emergent properties of our

learned retinal configurations, we train on artificial datasets where the factors of variation are easily controllable. Despite this departure from biology and natural stimuli, we find our model learns to create an eccentricity dependent layout where a distinct central region of high acuity emerges surrounded by a low acuity periphery. We show that the properties of this layout are highly dependent on the variations present in the task constraints. When we depart from physiology by augmenting our attention model with the ability to spatially rescale or 'zoom' on its input, we find our model learns a more uniform layout which has properties more similar to the 'glimpse window' proposed in Jaderberg, Simonyan, Zisserman, et al. [28] and Gregor et al. [17]. These findings help us to understand the task conditions and constraints in which an eccentricity dependent sampling lattice emerges.

3.1 Retinal Tiling in Neural Networks with Attention

Attention in neural networks may be formulated in terms of a differentiable feedforward function. This allows the parameters of these models to be trained jointly with backpropagation. Most formulations of visual attention over the input image assume some structure in the kernel filters. For example, the recent attention models proposed by Jaderberg, Simonyan, Zisserman, et al. [28], Mnih, Heess, Graves, et al. [47], Gregor et al. [17], and Ba, Mnih, and Kavukcuoglu [1] assume each kernel filter lies on a rectangular grid. To create a learnable retinal sampling lattice, we relax this assumption by allowing the kernels to tile the image independently.

Generating a Glimpse

We interpret a glimpse as a form of routing where a subset of the visual scene U is sampled to form a smaller output glimpse G. The routing is defined by a set of kernels $k[\bullet](s)$, where each kernel i specifies which part of the input $U[\bullet]$ will contribute to a particular output $G[i]$. A control variable s is used to control the routing by adjusting the position and scale of the entire array of kernels. With this in mind, many attention models can be reformulated into a generic equation written as

$$G[i] = \sum_{n}^{H} \sum_{m}^{W} U[n, m] k[m, n, i](s) \tag{3.1}$$

where m and n index input pixels of U and i indexes output glimpse features. The pixels in the input image U are thus mapped to a smaller glimpse G.

Retinal Glimpse

The centers of each kernel filter $\acute{\mu}[i]$ are calculated with respect to control variables s_c and s_z and learnable offset $\mu[i]$. The control variables specify the position and zoom of the entire glimpse. $\mu[i]$ and $\sigma[i]$ specify the position and spread respectively of an individual kernel

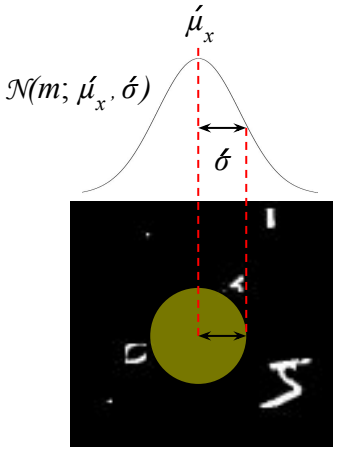

Figure 3.2: Diagram of single kernel filter parameterized by a mean $\acute{\mu}$ and variance $\acute{\sigma}$.

$k[-,-,i]$. These are parameters are learned during training with backpropagation. We describe how the control variables are computed in the next section. The kernels are thus specified as follows:

$$\acute{\mu}[i] = (s_c - \mu[i])s_z \tag{3.2}$$
$$\acute{\sigma}[i] = \sigma[i]s_z \tag{3.3}$$
$$k[m,n,i](s) = \mathcal{N}(m; \acute{\mu}_x[i], \acute{\sigma}[i])\mathcal{N}(n; \acute{\mu}_y[i], \acute{\sigma}[i]) \tag{3.4}$$

We assume kernel filters factorize between the horizontal m and vertical n dimensions of the input image. This factorization is shown in equation 3.4, where the kernel is defined as an isotropic gaussian $\mathcal{N}$. For each kernel filter, given a center $\acute{\mu}[i]$ and scalar variance $\acute{\sigma}[i]$, a two dimensional gaussian is defined over the input image as shown in Figure 3.2. These gaussian kernel filters are used to approximate the sensitivity profile of retinal ganglion cells in primates [65].

While this factored formulation reduces the space of possible transformations from input to output, it can still form many different mappings from an input U to output G. Figure 3.3B shows the possible windows which an input image can be mapped to an output G. The yellow circles denote the central location of a particular kernel while the size denotes the standard deviation. Each kernel maps to one of the outputs $G[i]$.

Positional control s_c can be considered analgous to the motor control signals which executes saccades of the eye in biology. In contrast, *training* defines *structural* adjustments to individual kernels which include its position in the lattice as well as its variance. These adjustments are only possible during training and are fixed afterwards. Training adjustments can be considered analagous to the layout of the retinal sampling lattice which is directed by evolutionary pressures in biology.

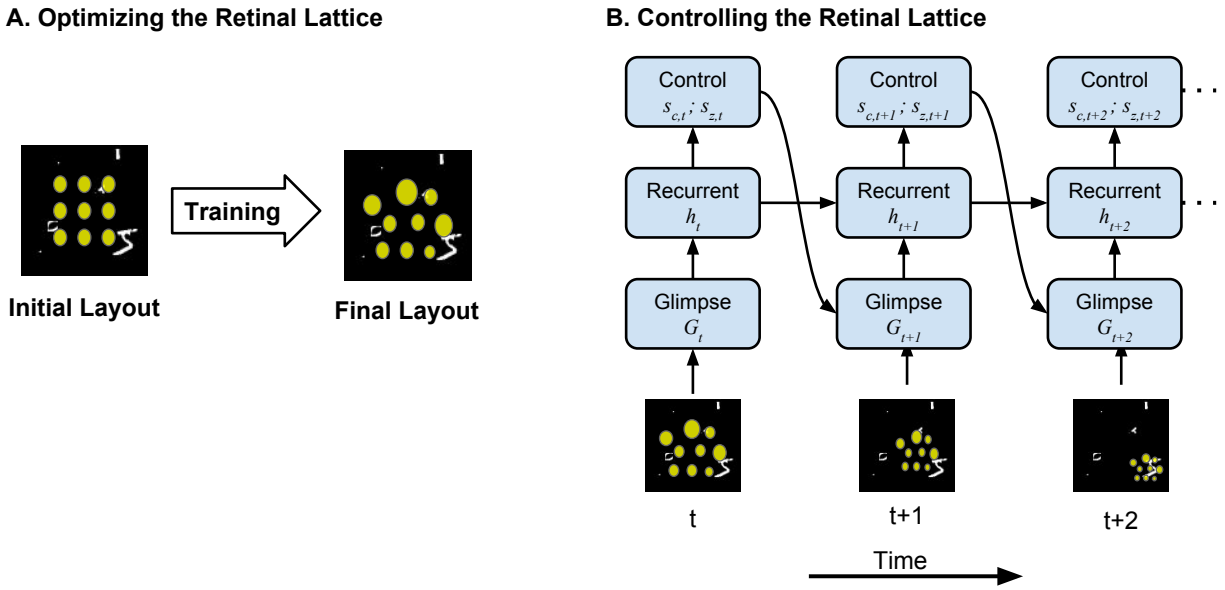

Figure 3.3: **A**: Starting from an initial lattice configuration of a uniform grid of kernels, we learn an optmized configuration from data. **B**: Diagram of our attention model unrolled for two timepoints after training during inference.

3.2 Recurrent Neural Architecture for Attention

A glimpse at a specific timepoint, G_t, is processed by a fully-connected recurrent network $f_{rnn}()$.

$$h_t = f_{rnn}(G_t, h_{t-1}) \tag{3.5}$$

$$[s_{c,t}; s_{z,t}] = f_{control}(h_t) \tag{3.6}$$

The global center $s_{c,t}$ and zoom $s_{z,t}$ are predicted by the control network $f_{control}()$ which is parameterized by a fully-connected neural network.

In this work, we investigate three variants of the proposed recurrent model:

- **Fixed Lattice:** The kernel parameters $\mu[i]$ and $\sigma[i]$ for each retinal cell are *not* learnable. The model can only translate the kernel filters $s_{c,t} = f_{control}(h_t)$ and the global zoom is fixed $s_{z,t} = 1$.
- **Translation Only:** Unlike the fixed lattice model, $\mu[i]$ and $\sigma[i]$ are learnable (via backpropagation).
- **Translation and Zoom:** This model follows equation 3.6 where it can both zoom and translate the kernels.

Table 3.1: Variants of the neural attention model

Ability	Fixed Lattice	Translation Only	Translation and Zoom
Translate retina via $s_{c,t}$	✓	✓	✓
Learnable $\mu[i]$, $\sigma[i]$		✓	✓
Zoom retina via $s_{z,t}$			✓

A summary for comparing these variants is shown in Table 3.1.

Prior to training, the kernel filters are initialized as a 12x12 grid (144 kernel filters), tiling uniformly over the central region of the input image and creating a retinal sampling lattice as shown in Figure 3.5 before training. Our recurrent network, f_{rnn} is a two layer traditional recurrent network with 512-512 units. Our control network, $f_{control}$ is a fully-connected network with 512-3 units (x,y,zoom) in each layer. Similarly, our prediction networks are fully-connected networks with 512-10 units for predicting the class. We use ReLU non-linearities for all hidden unit layers.

Our model as shown in Figure 3.3C are differentiable and trained end-to-end via back-propagation through time. Note that this allows us to train the control network indirectly from signals backpropagated from the task cost. For stochastic gradient descent optimization we use Adam [31] and construct our models in Theano [3].

3.3 Datasets and Tasks

Modified Cluttered MNIST Dataset

Example images from of our dataset are shown in Figure 3.4. Handwritten digits from the original MNIST dataset [38] are randomly placed over a 100x100 image with varying amounts of distractors (clutter). Distractors are generated by extracting random segments of non-target MNIST digits which are placed randomly with uniform probability over the image. In contrast to the cluttered MNIST dataset proposed in [47], the number of distractors for each image varies randomly from 0 to 20 pieces. This prevents the attention model from learning a solution which depends on the number 'on' pixels in a given region. In addition, we create another dataset (Dataset 2) with an additional factor of variation: the original MNIST digit is randomly resized by a factor of 0.33x to 3.0x. Examples of this dataset are shown in the second row of Figure 3.4.

Visual Search Task

We define our visual search task as a recognition task in a cluttered scene. The recurrent attention model we propose must output the class $\hat{c}$ of the single MNIST digit appearing in

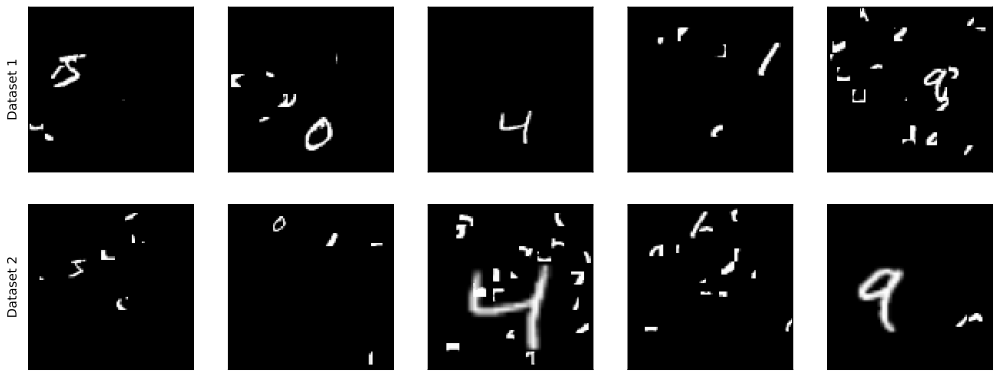

Figure 3.4: *First row* Examples from our variant of the cluttered MNIST dataset (a.k.a Dataset 1). *Second row* Examples from our dataset with variable sized MNIST digits (a.k.a Dataset 2).

the image via the prediction network $f_{predict}()$. The task loss, $\mathcal{L}$, is specified in equation 3.8. To minimize the classification error, we use cross-entropy cost:

$$\hat{c}_{t,n} = f_{predict}(h_{t,n}) \tag{3.7}$$

$$\mathcal{L} = \sum_{n}^{N} \sum_{t}^{T} c_n log(\hat{c}_{t,n}) \tag{3.8}$$

Analolgous to the visual search experiments performed in physiological studies, we pressure our attention model to accomplish the visual search as quickly as possible. By applying the task loss to every timepoint, the model is forced to accurately recognize and localize the target MNIST digit in as few iterations as possible. In our classification experiments, the model is given $T = 4$ glimpses.

3.4 Results

Figure 3.5 shows the layouts of the kernel filters for a Translation Only model. The filters are smoothly transforming from a uniform grid of kernels to an eccentricity dependent lattice. Furthermore, the kernel filters spread their individual centers to create a sampling lattice which covers the full image. This is sensible as the target MNIST digit can appear anywhere in the image with uniform probability.

When we include variable sized digits as additional factor in the dataset, the translation only model shows an even greater diversity of variances for the kernel filters. This is shown

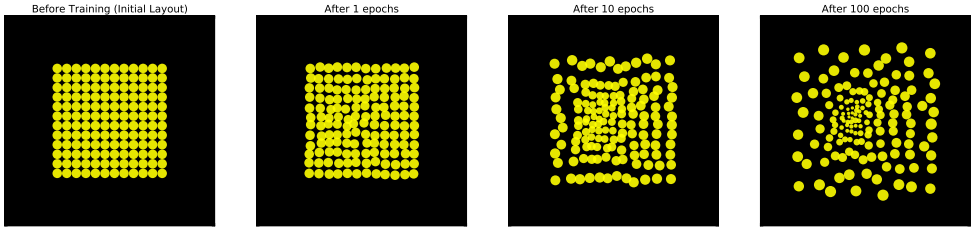

Figure 3.5: Scatter plot of the centers of each kernel filter during training for a Translation Only model. The radius of each point corresponds to the standard deviation σ_i of the kernel.

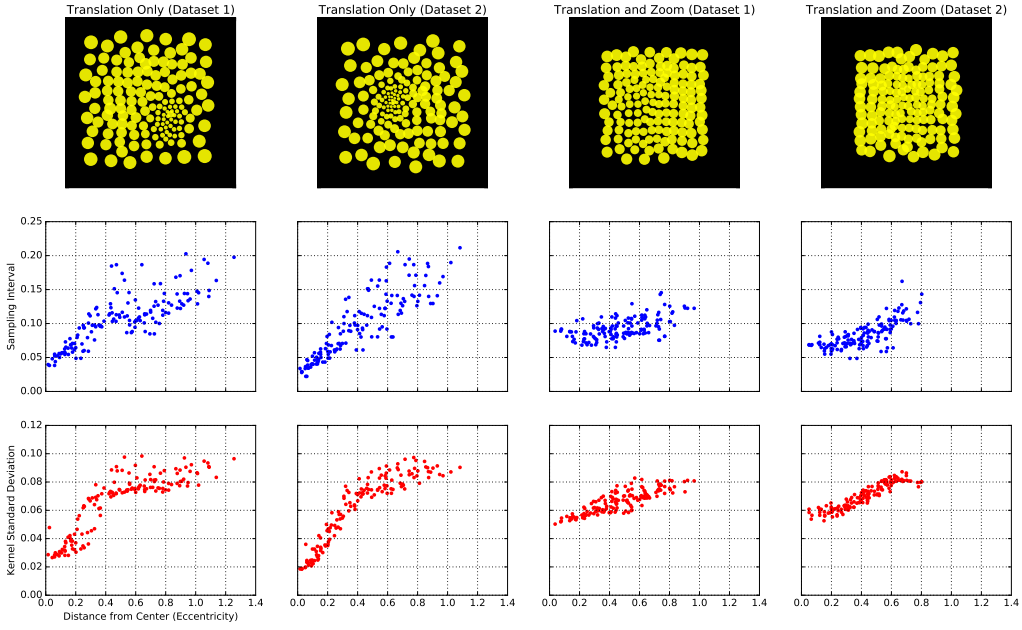

Figure 3.6: **A**: Histograms of the retinal sampling lattice for each model configuration. **B**: Resolution (sampling interval) as a function of eccentricity for Translate Only (RR) model configuration.

Table 3.2: Classification Error on Cluttered MNIST

Sampling Lattice Model	Dataset 1 (%)	Dataset 2 (%)
Fixed Lattice	11.8	31.9
Translation Only	5.1	24.4
Translation and Zoom	4.0	24.1

visually in the first row of Figure 3.6. Furthermore, the second row shows a highly dependent relationship between the sampling interval and standard deviatoin of the retinal sampling lattice and eccentricity from the center. This dependency increases when training on variable sized MNIST digits (Dataset 2). This relationship has also been observed in the primate visual system [49, 65].

When the proposed attention model is able to zoom its retinal sampling lattice, a very different layout emerges. There is much less diversity in the distribution of kernel filter variances as evidenced in Figure 3.6. Both the sampling interval and standard deviation of the retinal sampling lattice have far less of a dependence on eccentricity. As shown in the last column of Figure 3.6, we also trained this model on variable sized digits and noticed no significant differences in sampling lattice configuration.

Figure 3.7 shows how each model variant makes use of its retinal sampling lattice after training. The strategy each variant adopts to solve the visual search task helps explain the drastic difference in lattice configuration. The translation only variant simply translates its high acuity region to recognize and localize the target digit. The translation and zoom model both rescales and translates its sampling lattice to fit the target digit. Remarkably, Figure 3.7 shows that both models detect the digit early on and make minor corrective adjustments in the following iterations.

Table 3.2 compares the classification performance of each model variant on the cluttered MNIST dataset with fixed sized digits (Dataset 1). There is a significant drop in performance when the retinal sampling lattice is fixed and not learnable, confirming that the model is benefitting from learning the high-acuity region. The classification performance between the Translation Only and Translation and Zoom model is competitive. This supports the hypothesis that the functionality of a high acuity region is similar to that of zoom.

3.5 Conclusion

Remarkably, the kernels in the translation only model tile in a similar fashion to those found in the primate retina [6, 65]. This layout is composed of a high acuity region at the center surrounded by a wider region of low acuity. Van Essen and Anderson [65] postulate that the linear relationship between eccentricity and sampling interval leads to a form of scale invariance in the primate retina. Our results of the Translation Only model with variable sized digits supports this conclusion. Additionally, we observe that zoom appears to supplant

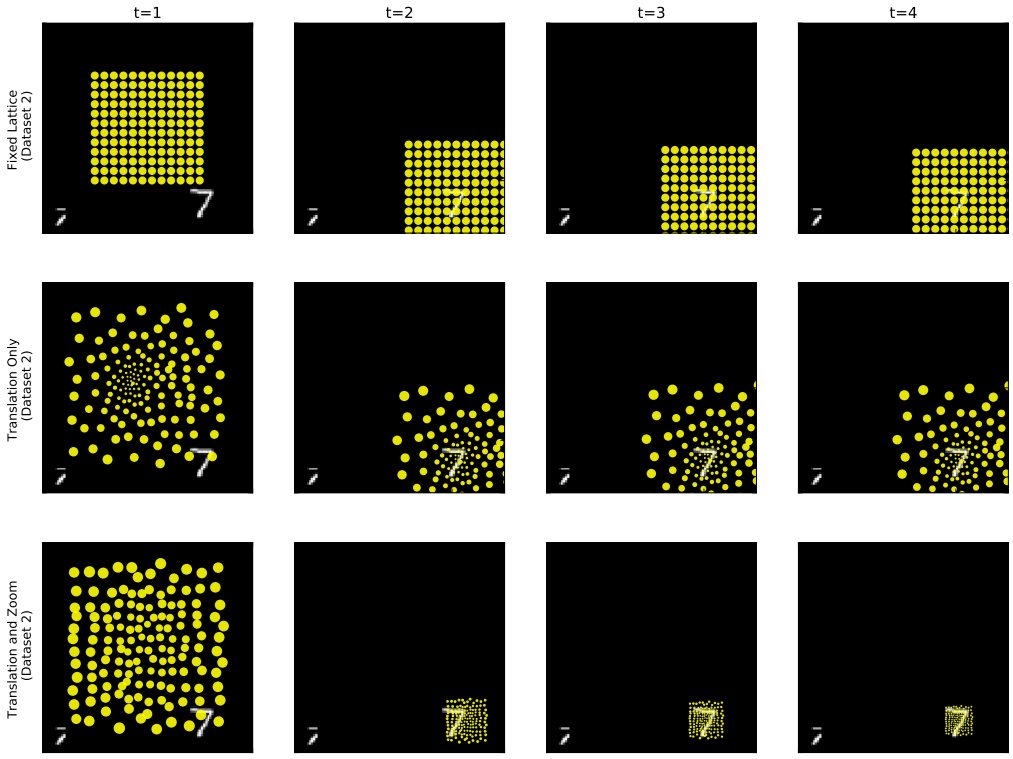

Figure 3.7: Temporal rollouts of the retinal sampling lattice attending over a test image from Cluttered MNIST (Dataset 2) after training.

the need to learn a high acuity region for the visual search task. This implies that the high acuity region serves a purpose resembling that of a rescalable sampling lattice while being more biologically plausible. The low acuity periphery is used to detect the search target and the high acuity 'fovea' more finely recognizes and localizes the target. These initial results indicate the possibility of using deep learning as a mechanism to discover the optimal tiling of retinal ganglion cells in a data driven and task-dependent manner.

Chapter 4

Relaxing learning constraints

Backpropagation is a key component in training neural network models which have been successfully applied to many perceptual tasks including vision and audio. Despite this success, an analogous learning mechanism has yet to be discovered in biology. The feedback alignment algorithm relaxes the constraints imposed by the backpropagation procedure while still demonstrating successful learning in feedforward neural networks. In this work, we further loosen the constraints of these learning algorithms by removing the directionality constraint of the forward and backward paths. We show these paths can be operated in a *bidirectional* manner to train multiple networks without interference even when the networks are solving different tasks.

Neural networks have been shown to perform remarkably well on a variety of machine learning tasks in various domains. They have made substantial improvements in speech recognition [23], image classification [34] and machine translation [62]. Despite the wide variety of architectures used for these applications, these models all share the same fundamental learning algorithm known as *backpropagation* [55]. The success of this training algorithm has raised the question of whether biological neurons could be employing a similar procedure [58].

One hope of finding an appropriate analogy between artificial neural networks and their biological counterparts is the possibility of discovering more robust and efficient learning mechanisms. Recent work has investigated the properties of learning as departures are taken from the standard backpropagation algorithm [27, 40, 7]. Dean et al. [7] showed weight updates could be performed asynchronously. Going further, Jaderberg et al. [27] decoupled the relative ordering of forward and backward passes.

Feedback alignment [41] has been a significant milestone in the path to finding a learning algorithm that has comparable performance with backpropagation in training multilayer neural networks while also being less restrictive. In standard backpropagation, the error signal is propagated through the feedback path via multiplications by the transpose of the feedforward weights. This requires a precise coupling between the weights in the feedforward and feedback paths of the neural network, where the weights along the feedback path must be exact symmetric copies of the weights along the feedforward path. This coupling is known

as the *weight transport problem* and is believed to be biologically implausible [20], because it requires that the learned feedforward weights be "transported" to the feedback path in order to generate learning signals for the neural network.

Feedback alignment offers a solution to the weight transport problem in which the weights along the feedback path of a neural network are randomly initialized and static. This simple solution removes the coupling constraint while still demonstrating successful learning.

In this work, we go further by making the backward weights dynamic and having them serve multiple purposes. Specifically, the backward weights serve:

- their original purpose as the backward weights for a neural network
- a secondary purpose as the forward weights for a different neural network

In contrast to the original feedback aligment algorithm, the secondary purpose changes the backward weights over time.

Our hypothesis is that given the additional flexibility of feedback alignment in training a neural network, we can utilize the backward weights to function as the forward weights for another neural network. In our work, we show that despite the multiple uses of the backward weights in a network, each network still trains without interference from the other network.

4.1 Overview of Feedback Alignment

Feedback alignment has been empirically shown to train fully-connected neural networks to a similar performance to standard backpropagation. The normal backpropagation formulation for a given weight matrix is:

$$\frac{\partial \mathcal{L}}{\partial W^l} = \frac{\partial \mathcal{L}}{\partial z^l} h^{(l-1)T} \tag{4.1}$$

which for a standard feedforward neural network with non-linearity $\sigma()$ is written out as:

$$z^l = W^l h^{l-1} + b^l \tag{4.2}$$

$$h^l = \sigma(z^l) \tag{4.3}$$

$$\frac{\partial \mathcal{L}}{\partial z^l} = (W^{(l+1)T} \frac{\partial \mathcal{L}}{\partial z^{(l+1)}}) \odot \sigma'(z^l) \tag{4.4}$$

where $\odot$ is element-wise multiplication.

In feedback alignment, the transposed weight matrices $W^{(l+1)T}$ in equation 4.4 for each layer are replaced by:

$$\frac{\partial \mathcal{L}}{\partial z^l} = (B^{(l+1)T} \frac{\partial \mathcal{L}}{\partial z^{(l+1)}}) \odot \sigma'(z^l) \tag{4.5}$$

where $B^{(l+1)T}$ is a fixed randomly initialized weight matrix. This random matrix relieves the constraint of backpropagation where the transpose of the forward weight matrix is normally needed on the feedback path of the network.

4.2 Bidirectional Feedback Alignment

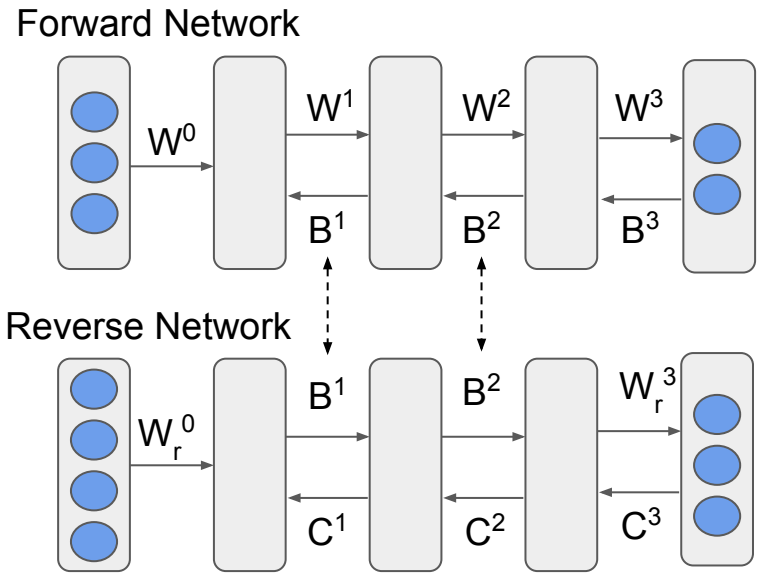

Figure 4.1: Diagram of our bidirectional feedback method. Only weights of the hidden layers are coupled to match in number of units.

Recent work has shown that a significant fraction of the parameters of a neural network can be removed after training without compromising accuracy [22, 9]. The work of Frankle and Carbin [9] suggests that many of the parameters in large neural networks are only utilized for increasing the chances of sampling a good initialization. Otherwise, these weights are left unused during learning.

With this large fraction of unused parameters in mind, we hypothesize that the backward weights of a neural network trained with feedback alignment could be used for multiple purposes without impacting the overall learning process of the forward weights. If only a small random fraction of the backward weights are used for one purpose, there is a low chance that they would interact when co-opted for another purpose. Rather than having the feedback weights carry only gradient information to the lower layers of a neural network, we investigate using these weights as the feedforward weights of a different neural network:

$$z_r^l = B^l h_r^{l-1} + b_r^l \tag{4.6}$$
$$h_r^l = \sigma(z_r^l) \tag{4.7}$$

In contrast to the original feedback alignment procedure, the backward weights B^l here are not kept fixed throughout learning. These weights are updated by another feedback alignment procedure to train a *reverse network* whose components are denoted by the subscript r.

Backward and Forward Coupling

In the previous section, the reverse network possessed its own backward matrices C^l which are randomly initialized and fixed (see Figure 4.1). We go further to investigate whether coupling the backward matrix of the reverse network to the forward network has any impact on learning using feedback alignment:

$$\frac{\partial \mathcal{L}_r}{\partial z_r^l} = (W^{(l+1)T} \frac{\partial \mathcal{L}_r}{\partial z_r^{(l+1)}}) \odot \sigma'(z_r^l) \tag{4.8}$$

4.3 Experiments

We train two distinct 5 layer fully connected neural networks with 1024 units in each hidden layer. One network is trained on the CIFAR-10 task while the other is trained on the MNIST task. To improve convergence speed, we center the pre-activations across the feature dimension. This centering is equivalent to multiplying by a fixed symmetric matrix in the forward and backward directions:

$$C = \begin{bmatrix} 1 - \frac{1}{N} & -\frac{1}{N} & -\frac{1}{N} & \dots & -\frac{1}{N} \\ -\frac{1}{N} & 1 - \frac{1}{N} & -\frac{1}{N} & \dots & -\frac{1}{N} \\ \dots & \dots & \dots & \dots & \dots \\ -\frac{1}{N} & -\frac{1}{N} & -\frac{1}{N} & \dots & 1 - \frac{1}{N} \end{bmatrix} \tag{4.9}$$

$$z_{centered} = Cz \tag{4.10}$$

Aside from improving learning speed, we found this centering did not impact our results or conclusions.

4.4 Results

We compare the accuracy during learning of standard feedback alignment with bidirectional feedback alignment. In standard feedback alignment, the forward and reverse networks are independently learning over the CIFAR-10 and MNIST datasets respectively. Figure 4.2

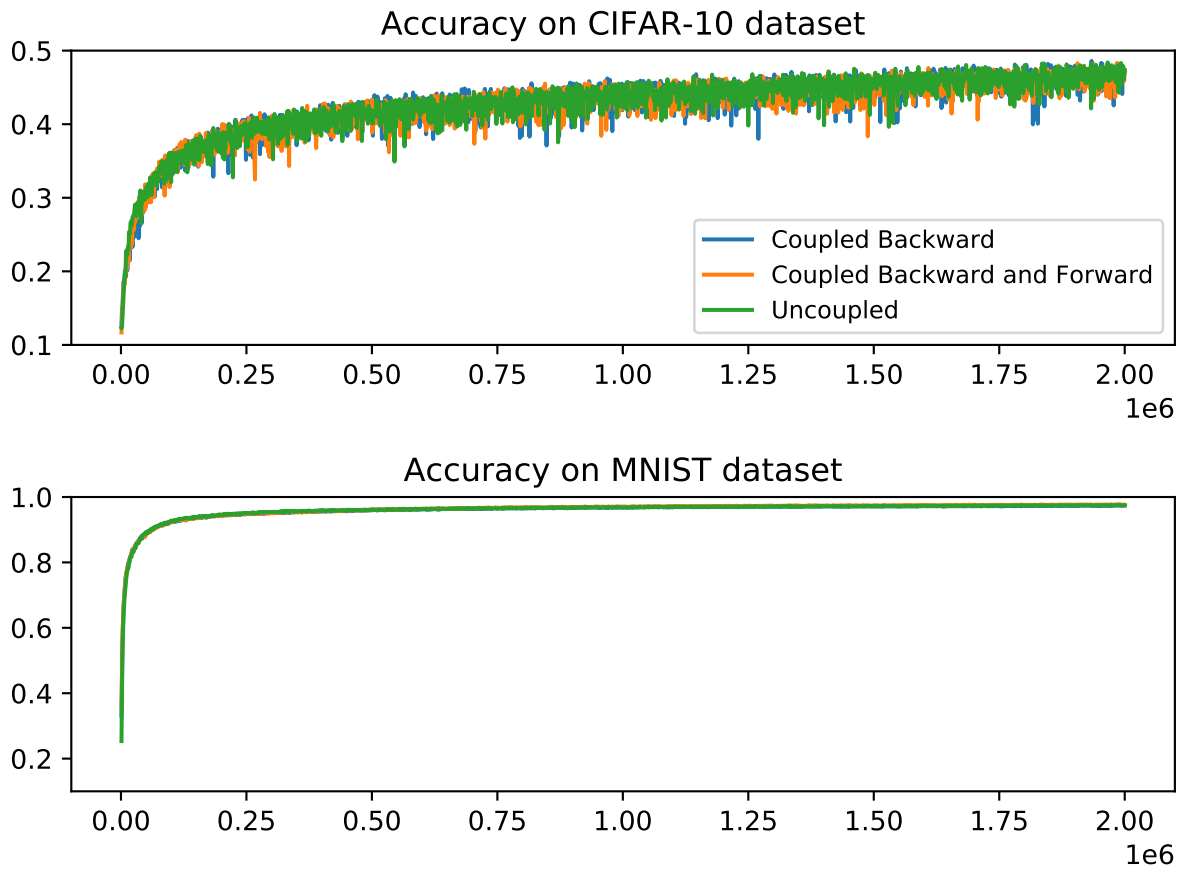

Figure 4.2: Test accuracy during training of standard Feedback Alignment (green), Coupled Backward (blue), and Coupled Backward and Forward (orange) on the CIFAR-10 (top) and MNIST (bottom) datasets.

shows that coupling the backward matrix of the forward network with the reverse network converges at the same rate for both datasets. Furthermore, fully coupling the forward and reverse networks (Backward and Forward coupling) also converges at the same rate as standard feedback alignment.

To determine if there are any interactions while simultaneously training the forward and reverse networks, we compute the cosine similarity of the weight matrices of each coupled layer which is shown in Figure 4.3. Unsurprisingly, when the networks are uncoupled (standard feedback alignment), the weights in all layers have nearly zero cosine similarity. In contrast, when the forward and reverse networks are coupled by the backward matrix, the alignment between the weights increases as training progresses. When the forward and reverse networks are fully coupled, we see the greatest alignment during training. Interestingly the alignment is larger in the earlier layers when compared to the downstream layers.

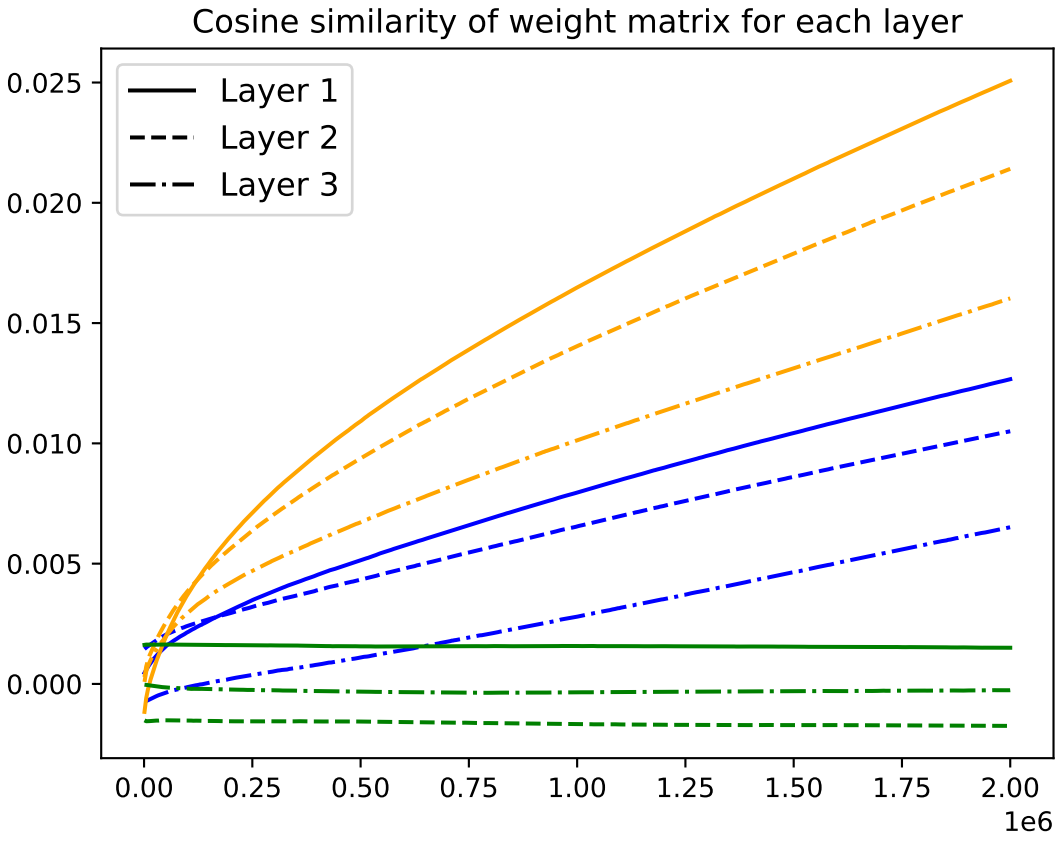

Figure 4.3: Cosine similarity between weights in the forward and reverse networks during training iterations for standard Feedback Alignment (green), Coupled Backward (blue), and Coupled Backward and Forward (orange).

4.5 Discussion

We show preliminary results demonstrating that the feedforward and feedback pathways in a neural network can travel along the same connections, further relaxing the constraints which were initially thought necessary for training neural networks architectures. Instead of keeping the backward weights fixed during learning, we allow the backward matrix to be updated to solve a different task. We show that this does not impact learning performance of the original feedback alignment algorithm.

We believe that tempering the restrictions of when learning is feasible will lead to algorithms which are more resilient to unforeseen changes and may make it easier to find analogies in biology.

Chapter 5

Guiding learning for multiple tasks

We present a method for storing multiple models within *a single set of parameters*. Models can coexist in *superposition* and still be retrieved individually. In experiments with neural networks, we show that a surprisingly large number of models can be effectively stored within a single parameter instance. Furthermore, each of these models can undergo thousands of training steps without significantly interfering with other models within the superposition. This approach may be viewed as the *online* complement of compression: rather than reducing the size of a network after training, we make use of the unrealized capacity of a network *during training*.

Connectionist models have enjoyed a resurgence of interest in the artificial intelligence community. In particular, *neural network* models have demonstrated remarkable performance on many tasks across the domains of vision, text and speech. But in practice, a separate model is dedicated to each of these tasks. Drawing inspiration from another classic connectionist model, the *associative memory*, we develop a framework for combining multiple distinct models into *one superposition of models*. By utilizing the dormant capacity already present in neural networks, it is possible to reliably store and retrieve models that are dynamically changing due to learning.

We propose a modification to a fundamental operation performed across many representation learning models: the *linear transformation*. With a simple change to this nearly ubiquitous operation, we convert the linear transformation itself into a memory. Stored within this memory are multiple linear transformations in a state of superposition. Context information is used to recall an individual linear transformation from this superposition. An overview of this process is shown in Figure 5.1.

Instead of trying to share a single model across multiple tasks, individual models for each task can coexist with one another in superposition. Context information dynamically 'routes' an input towards a specific model within this superposition. The ability to route inputs to models *during training* opens up many possibilities in online learning and learning in memory constrained environments.

The goal of this paper is to introduce a general framework for parameter superposition and to empirically demonstrate both its features and limitations. By establishing a foundation

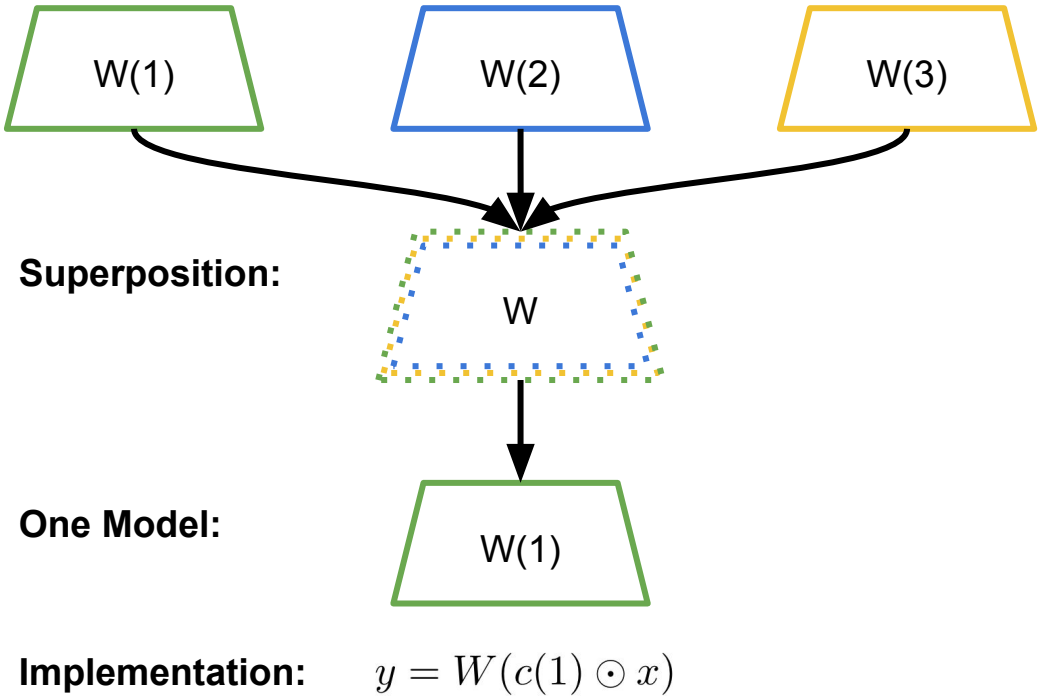

Figure 5.1: Models $W(1)$, $W(2)$ and $W(3)$ are superimposed in W. Context $c(1)$ is a vector (e.g. random $\{-1, 1\}$ values) which implicitly retrieves model $W(1)$ via $y = W(c(1) \odot x)$.

for this new framework, we embrace extensions into these domains and beyond.

5.1 Background

An associative memory is a form of content-addressable memory where recall is based on similarity to the content of stored patterns [42]. Unlike traditional computer memory which is accessed by an explicit addressing scheme where the address must be exact, associative memories are robust to corruptions during recall. This resilience makes them particularly useful in representation learning where training data is imprecise and the training process itself is stochastic.

Associative memories are categorized by the way in which memories are retrieved. An *auto-associative memory* is a memory where an approximate (or partial) memory is used to recover the complete memory of itself, with the Hopfield Network [25] being a notable example. In contrast, a *hetero-associative memory* uses content which is different from the memory recovered by that content. From a data structures perspective, this is similar in spirit to a hash-map where a key can be used to retrieve a corresponding value. In this work, we refer to this key as *context information*.

A *holographic reduced representation (HRR)* is a hetero-associative memory proposed by Plate [50] for storing compositional structures. In contrast to Hopfield networks, which store

individual items, HRRs store pairs of items (v_k, c_k) in superposition. The act of forming these pairs is called *binding*. One component of the pair can be retrieved using the other component as a cue. For example, c_k can be used to restore v_k from memory m:

$$m = \sum_s v_s \odot c_s^{-1}$$
$$v_k \approx m \odot c_k$$

where $\odot$ is a binding operator. This is akin to a bidirectional hash-map where a key can be used to retrieve a corresponding value and vis-versa (i.e. reverse lookup). Due to the superposition however, noise is introduced by this retrieval process, and so a 'clean-up' process is necessary to restore the retrieved memory to its original state. Plate [50] suggest using a separate auto-associative memory for clean-up.

5.2 Motivation

Kanerva [29] proposed a *hyperdimensional computing* framework which utilizes simple arithmetic operations to manipulate hetero-associative memories like HRRs. Storage, addressing and erasure are accomplished by addition, multiplication and subtraction operations respectively. Drawing inspiration from this idea, we develop *parameter superposition*, a memory framework which stores the parameters of learning systems as memories. We relax the requirement of a clean-up process which greatly simplifies our framework. To justify this relaxation, we posit that the parameters of these learning systems are robust to the noise caused by retrieval.

Robustness to noise requires some amount of redundancy. Signs of redundancy can manifest themselves in opportunities for compression. Previous work has shown that the parameters of a neural network can be drastically compressed *after training* [67, 21].

Han, Mao, and Dally [21] show a majority of trained parameters in networks can be pruned with a simple magnitude-based thresholding procedure. More surprisingly, Frankle and Carbin [9] and Liu et al. [43] show the pruning mask acquired after training can be applied to the same network before training with little to no adverse impact on learning. Li et al. [39] found the intrinsic dimensionality of most tasks to be a small fraction of the dimensionality of the parameters. Such results imply that parameters are not being used efficiently in neural networks. This leads us to ask, can we take advantage of any redundancies during training?

Through experiments on multiple datasets, tasks and types of neural networks, we show parameter superposition can indeed exploit the excess capacity present in these models. Furthermore, our framework can exploit this capacity *during training*.

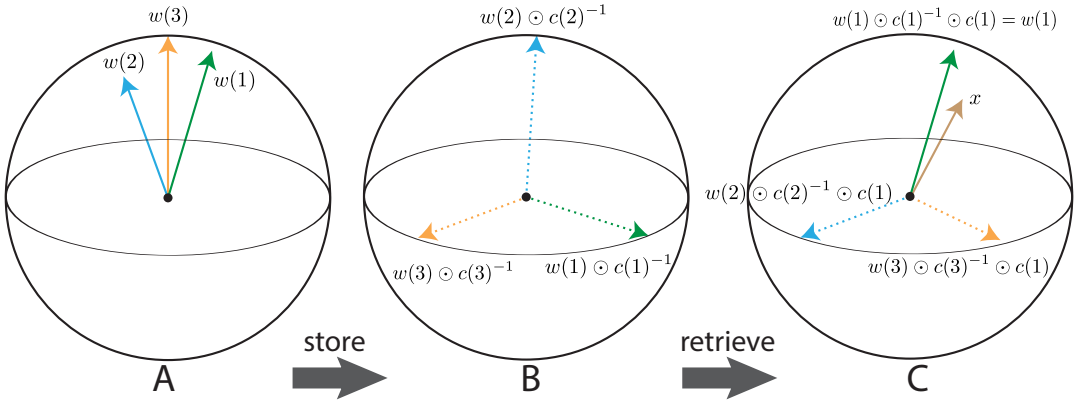

Figure 5.2: **A** Parameter vectors $w(s), s \in \{1, 2, 3\}$ have high similarity. **B (Store)** $w(s) \odot c(s)^{-1}$ will rotate each $w(s)$ making them nearly orthogonal. **C (Retrieve)** $w(s) \odot c(s)^{-1} \odot c(k)$ will restore $w(k)$ but other $s \neq k$ will remain nearly orthogonal, reducing interference during learning.

Online Learning

Training typically requires a large dataset of labelled examples, from which small batches of data are sampled to evaluate the loss function and consequently update the neural network parameters. For successful training, it has been found critical to construct batches by sampling data uniformly at random. Such random sampling ensures that the training data is independent and identically distributed.

Past attempts to address this problem have employed various mechanisms to overcome this issue: the use of replay buffers in reinforcement learning [46], using separate networks for separate tasks [56, 64], or heuristics to selectively identify which weights can be changed during training [32, 72]. The core issue is to come up with a formulation where it is possible to remember the past while maintaining the ability to learn from new data.

5.3 Parameter Superposition

With inspiration from the superposition principle of linear systems, we propose a method called *Parameter Superposition (PSP)* to store many models simultaneously into one set of parameters. We augment the standard linear transformation $y = Wx$ with a binding operation:

$$y = W(c(k) \odot x) \tag{5.1}$$

Context information in the form of $k \in S$ generates a context vector $c(k) \in \mathbb{C}^M$. The matrix $W \in \mathbb{C}^{N \times M}$ represents the parameters of the linear transformation. The symbol $\odot$ refers to element-wise multiplication. By multiplying input vector $x \in \mathbb{C}^M$ with a context vector $c(k)$, x is 'rotated' to a particular model $W(k)$ stored within W.

Since multiplication is associative, we can instead view this as a rotation of a particular model $W(k)$ stored in the rows of W towards x. This rotation is illustrated in the retrieval process shown in Figure 5.2. An overview of ways to generate these rotations is provided in Section 5.3.

Conceptually, one can think of the parameters W as a superposition of parameters $W(s)$ where $s \in S$ is the dimension of superposition:

$$W_{ij} = \sum_s W(s)_{ij} c(s)_j^{-1} \tag{5.2}$$

This is similar to the superposition principle in Fourier analysis where a signal is represented as a superposition of sinusoids. Each sinusoid can be considered as a context vector. For the Inverse Fourier transform, the dimension of superposition is the frequency of those sinusoids:

$$w(t) = \int_{-\infty}^{\infty} \hat{w}(s) e^{2\pi i s t} \mathrm{d}s$$

By using the complement (inverse) of a particular Fourier basis (context vector), we can retrieve a stored amplitude value. In our framework, when set S is discrete and finite, element $k \in S$ can be used to recover individual $W(k)$ from W by generating a corresponding context vector $c(k)$.

Properties of this recovery process are more clearly illustrated by substituting Equation 5.2 into Equation 5.1. By unpacking the inner product operation, Equation 5.1 can be rewritten as the sum of two terms which is shown in Equation 5.3.

$$\begin{aligned} y_i &= \sum_j \sum_s W(s)_{ij} c(s)_j^{-1} c(k)_j x_j \\ &= \sum_j W(k)_{ij} x_j + \sum_j \sum_{s \neq k} W(s)_{ij} c(s)_j^{-1} c(k)_j x_j \end{aligned}$$

which is written more concisely in matrix notation as:

$$\begin{aligned} y &= W(k)x + \epsilon \\ \epsilon &= \sum_{s \neq k} W(s)(c(s)^{-1} \odot c(k) \odot x) \end{aligned} \tag{5.3}$$

The first term, $W(k)x$, is the recovered linear transformation and the second term, ϵ, is a residual. For particular formulations of the set of context vectors $c(S)$, ϵ is a summation of terms which interfere destructively.

Analysis of residual ϵ

We establish properties of the destructive interference which make it possible to recover a linear transformation from the superposition.

Proposition 1 *ϵ in expectation is unbiased, $E_s[\epsilon] \to 0$.*

Proposition 1 states that, in expectation, other models within the superposition will not introduce a bias to the recovered linear transformation.

Proposition 2 *For $x,\ w \in \mathbb{C}^M$, when we bind a random $c \in \mathbb{C}^M$ with w, $\frac{\mathrm{Var}\,[\langle c \odot w, x\rangle]}{\|w\|^2\|x\|^2} \approx \frac{1}{M}$ under mild conditions. For $x,\ w \in \mathbb{R}^M$, let C_k be a random orthogonal matrix s.t. $C_k w$ has a random direction. Then $\frac{\mathrm{Var}\,(\langle C_k w, x\rangle)}{\|w\|^2\|x\|^2} \approx \frac{1}{M}$. In both cases, let $|\langle w, x\rangle| = \|w\|\|x\|\eta$. If η is large, then $|\langle c \odot w, x\rangle|$ and $|\langle C_k w, x\rangle|$ will be relatively small compared to $|\langle w, x\rangle|$.*

If we assume that $\|w(k)\|$'s are equally large and denote it by γ, then $\epsilon \propto \frac{K-1}{M}|\langle w, x\rangle|^2$. When $\frac{K-1}{M}$ is small, the residual introduced by other superimposed models will stay small. Binding with the random keys roughly attenuates each model's interference by a factor proportional to $\frac{1}{\sqrt{M}}$.

Propositions 1 and 2 show that we can superimpose individual models after training and the interference should stay small. In next proposition, we describe having individual models in superposition during training.

Proposition 3 *Denote the cost function of the network with PSP as J_{PSP} used in context k, and the cost function of the k^{th} network without as J_k.*

1. *For the complex context vector case, $\frac{\partial}{\partial W} J_{PSP} \approx \left(\frac{\partial}{\partial W(k)} J_k\right) \odot c(k)$, where $c(k)$ is the context vector used in J_{PSP} and $W(k)$ are the weights of the J_k network.*
2. *For the general rotation case in real vector space, $\frac{\partial}{\partial W} J_{PSP} \approx (\frac{\partial}{\partial W(k)} J_k) C_k$, where C_k is the context matrix used in J_{PSP} and $W(k)$ are the weights of the J_k network.*

Proposition 3 shows parameter updates of an individual model in superposition is approximately equal to updates of that model trained outside of superposition. The gradient of parameter superposition creates a *superposition of gradients* with analogous destructive interference properties to Equation 5.1. Therefore, memory operations in parameter superposition can be applied in an online fashion. A proof of the propositions above is provided in the appendix.

In the following sections, we describe multiple formulations of context vectors in the superposition framework which promote this destructive interference.

A more general framework

By replacing element-wise multiply with matrix multiply, Equation 5.1 can be generalized:

$$y = WC(k)x \tag{5.4}$$

where $C(k) \in \mathbb{C}^{M \times M}$. The left action of $C(k)$ on x can also be viewed as a right action of $C(k)$ on the rows of W. From this perspective, parameter superposition rotates a row of parameters $w \in \mathbb{C}^M$ with a unitary transformation $C(k)$.

$C(k) : \mathbb{C}^M \to \mathbb{C}^M$ is an operator which rotates w into a different region of feature space. As $k \in S$ generates $C(k)$, k directs where w is rotated. Through this rotation, k controls the degree of overlap between different $w(s)$ in superposition. In turn, this controls the amount of interference during learning between different $w(s)$. Figure 5.2 is a visualization of this binding process. In the following sections, we review various ways to generate these unitary transformations.

Complex Superposition

In Equation 5.1, we described *complex superposition*, a form of parameter superposition where complex vectors are used to efficiently generate unitary transformations, $C(k) = diag(c(k))$.

$$c(k)_j = e^{i\phi_j(k)} \tag{5.5}$$

where each component $c(k)_j$ is on the complex unit circle $\mathbb{T}$. The phase $\phi_j(k) \in [-\pi, \pi]$ for all j is sampled with uniform probability density $p(\phi) = \frac{1}{2\pi}$.

This form of parameter superposition is particularly relevant for the predominant form of neural network models. The differentiability with respect to phase ϕ enables gradient-based learning through the context. Moreover, the size of the unitary transformation scales linearly with the dimensionality M of the input vector.

Binary Superposition

Constraining the phase to two possible values $\phi_j(k) \in \{0, \pi\}$ is a special case of complex superposition. The context vectors become $c(k)_j \in \{-1, 1\}$. We refer to this formulation as *binary superposition*. The low-precision of the context vectors in this form of superposition has both computational and memory advantages. Furthermore, binary superposition is directly compatible with both real-valued and low-precision linear transformations.

Rotational Superposition

Binary and complex superposition are special subgroups of the larger group of $M \times M$ unitary matrices (i.e. unitary group). For completeness, the superposition principle can also extend to a broader class of orthogonal matrices $C(k) \in \mathbb{R}^{M \times M}$. This class includes rotations which are not possible with diagonal matrices (e.g. permutation). We refer to this formulation as *rotational superposition*. These rotations are sampled uniformly from the orthogonal group $O(M)$ (Haar distribution)[1].

For each type of superposition, Figure 5.3 provides the geometry of the rotations which can be applied to parameters w. This illustrates the topology of the embedding space of superimposed models. The choice of how to parameterize the unitary transformation in superposition depends on the specific application. For example, a continuous topology

[1] we use `scipy.stats.ortho_group`.

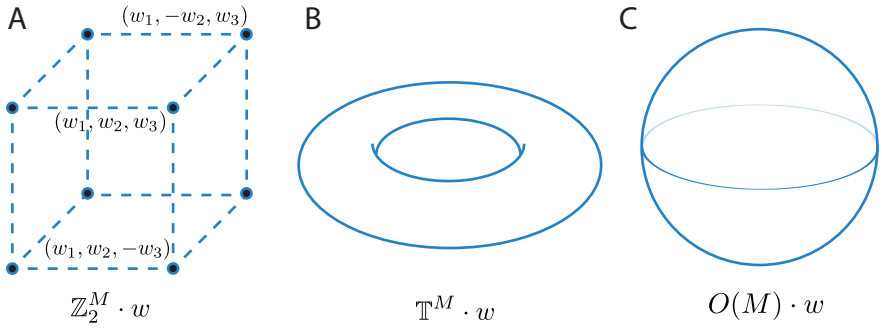

Figure 5.3: The topology of all context operators acting on a vector w, e.g. $\mathbb{T}^M \cdot w = \{c \odot w : c \in \mathbb{T}^M\}$. **A** binary operates on a lattice **B** complex operates on a torus **C** rotational operates on a sphere.

enables gradient-based learning of the context operator. Without loss of generality, we describe ideas in the following sections from the perspective of complex superposition.

5.4 From superposition to composition

While a context is an operator on parameter vectors w, the context itself can also be operated on. Analogous to the notion of a group in abstract algebra, new contexts can be constructed from a composition of existing contexts under a defined operation. For example, the context vectors in complex superposition form a Lie group under complex multiplication. This enables parameters to be stored and recovered from a composition of contexts:

$$c^{a+b} = c^a \odot c^b \tag{5.6}$$

By creating functions $c(k)$ over the superposition dimension $k \in S$, we can generate new context vectors in a variety of ways. To introduce this idea, we describe two basic compositions.

Mixture of contexts

The continuity of the phase ϕ in complex superposition makes it possible to create mixtures of contexts to generate a smoother transition from one context to the next. One basic mixture is an average window over the previous, current and next context:

$$c(k) = e^{i\frac{\phi(k-1)+\phi(k)+\phi(k+1)}{3}} \tag{5.7}$$

The smooth transitions reduces the orthogonality between neighboring context vectors. Parameters with neighboring contexts can 'share' information during learning which is useful for transfer-learning settings and continual learning settings where the domain shift is smooth.

	# parameters	+1 model
No Superposition	MN	MN
Binary	$M(N+1)$	M
Complex	$2M(N+0.5)$	M
Rotational	$M(N+M)$	M^2
OnePower (Complex)	$2M(N+0.5)$	1

Table 5.1: Parameter count for superposition of a linear transformation of size $M \times N$. '+1 model' refers to the number of additional parameters to add a new model.

Powers of a single context

A particularly memory efficient way of generating new contexts is raising a given context to a power (exponentiation). For complex superposition, a given context vector $c \in \mathbb{C}^M$ can be used to create new contexts by:

$$c(k) = e^{ik\phi} \tag{5.8}$$

Superposition with powers has the advantage of a constant-sized memory footprint even as new models are added in superposition. This enables efficient communication of context changes with a single value k without the need to store a set of unique contexts.

In complex superposition, context vectors exist in the topology of a complex M-dimensional torus. Many other functions $c(k)$ can be defined over this smooth manifold. When these functions are differentiable, gradient-based optimization (e.g. backpropagation) can be used to learn $c(k)$ from data. In this work, we focus on first introducing basic operations over contexts vectors and leave such extensions for future work.

5.5 Neural Network Superposition

We outlined multiple formulations of parameter superposition which involves a simple modification of the standard linear transformation. This transformation is a fundamental operation utilized by most neural network models. We can extend these formulations to entire neural network models by applying superposition (Equation 5.1) to the linear transformation of all layers l of a neural network:

$$x^{(l+1)} = g(W^{(l)}(c(k)^{(l)} \odot x^{(l)})) \tag{5.9}$$

where $g()$ is a non-linear (activation) function.

Convolutional Networks

For neural networks applied to vision tasks, convolution is currently the dominant operation in a majority of layers. Since the dimensionality of convolution parameters is usually much smaller than the input image, it makes more sense computationally to apply context to

the weights rather than the input. By associativity of multiplication, we are able reduce computation by applying a context tensor $c(k) \in \mathbb{C}^{M \times H_w \times W_w}$ to the convolution kernel $w \in \mathbb{C}^{N \times M \times H_w \times W_w}$ instead of the input image $x \in \mathbb{C}^{M \times H_x \times W_x}$:

$$y_n = (w_n \odot c(k)) * x \tag{5.10}$$

where $*$ is the convolution operator.

5.6 Experiments

While superposition is a general memory framework for storing parameters and has many potential applications, our goal in this paper is to demonstrate the *capacity* and *robustness* of recovering models from the parameter superposition despite thousands of updates to these parameters.

Learning interference occurs when the distribution of training data shifts during training. The problem is so acute that continual learning literature often refers to it as *catastrophic interference* or *forgetting*. A network trained on multiple consecutive tasks will suddenly 'forget' or perform poorly on earlier tasks. This can be considered a form of overfitting where the model *temporally overfits* to the data it is currently presented, generalizing poorly to data at other timepoints.

We present experiments on multiple types on non-stationary data to illustrate the ability of online learning using the super-position framework. Table 5.1 describes the number of parameters required as a reference for the parameter cost of each formulation of superposition.

Input Interference

A common scenario in online learning is when the input data distribution changes over time (for e.g. as visual properties change from day to night.) Previous works have used datasets such as the The permuting MNIST dataset [11] is a variant of the MNIST dataset [38] where the pixels of the input are randomly permuted during online learning. Each new permutation is considered a new task. Since the labels remain the same for each task, there is no change in the output distribution. Though permutation is an unrealistic distribution shift, we believe training on this dataset can provide a good demonstration of the capacity of a model during online learning.

Network size

In previous work, relatively large networks ($\sim$2000 units) are trained on 10 tasks (10 consecutive permutations) [72]. We would like to investigate the impact of the dimensionality of the linear transformation on the superposition framework. To amplify these differences, we train networks of two hidden layers on 50 tasks (1000 steps per task) and vary the number of units (128 to 2048). A new context vector is generated when transitioning to a new

Figure 5.4: Binary superposition using networks with differing number of units (128 to 2048). Each line is permuting MNIST task 1 accuracy as a function of training on 50 tasks total.

task. We use binary superposition (*pspBinary*) to minimize the differences from the baseline architecture (standard).

In Figure 5.4, we see consistent improvements as the dimensionality of the feature space grows for parameter superposition but not a standard network of the same architecture. This shows that the superposition framework is making better use of the additional capacity present in the larger models.

Parameter efficiency

If parameter superposition better utilizes the training capacity of a neural network, we expect to see a corresponding increase in parameter utilization after training. Following previous work on network compression [21], we compare how prune-able the weights of a network are after training on permuting MNIST (50 tasks). In Figure 5.5, we see the presence of higher magnitude weights in a network with superposition than without. This suggests that superposition makes the network less compressible with magnitude based thresholding (pruning). We see this trend is more pronounced the more downstream the layer. We conjecture the permutation operation in permuting MNIST is a rotation operation making it behave similar to a context rotation in superposition at the early layers.

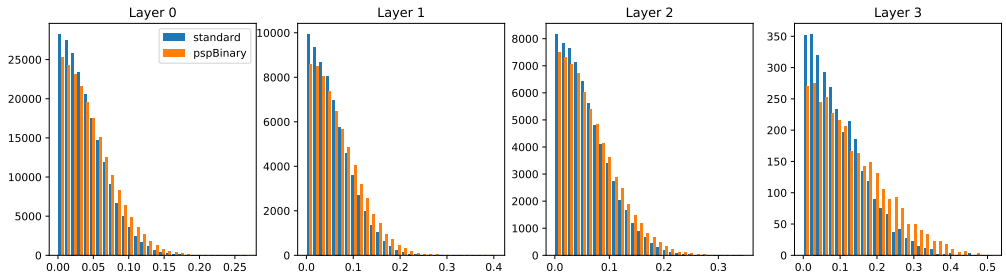

Figure 5.5: Histogram of weight magnitudes for each layer after training on permuting MNIST (50 tasks) using binary superposition (orange) and no superposition (blue).

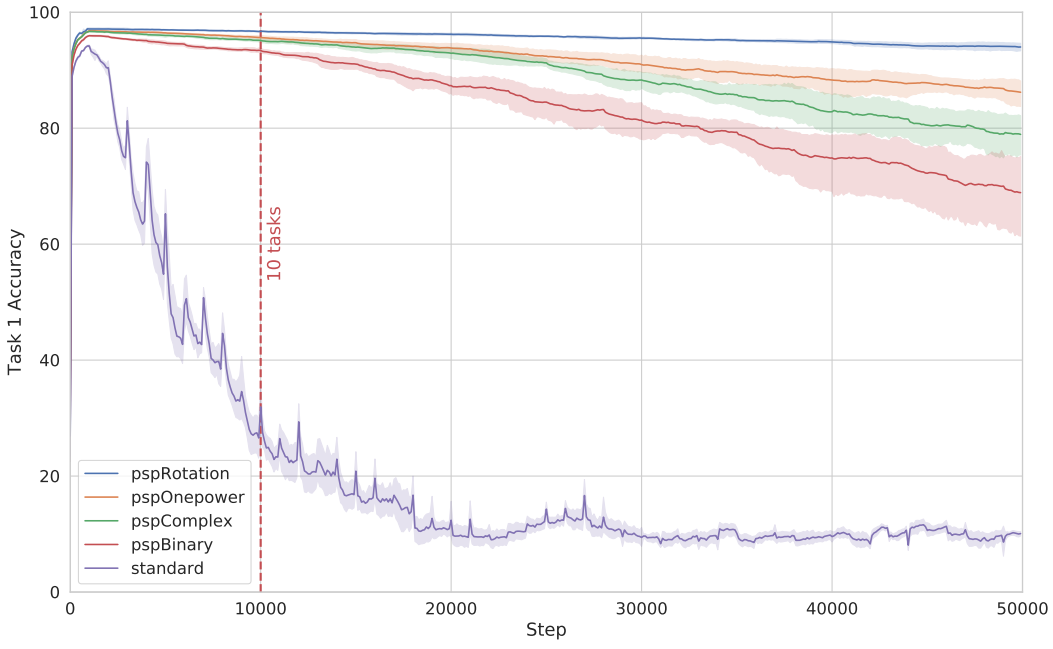

Figure 5.6: Permuting MNIST task 1 accuracy on test set as a function of training step. At 1 task per 1000 steps, each network has seen 50 tasks in sequence at the end of training.

	Avg. Accuracy (%)
[32]*	97.0
[72]*	97.2
No Superposition	61.8
Binary	97.6
Complex	97.4
OnePower (Complex)	97.2

Table 5.2: Average accuracy of a 2000 unit two hidden layer network across 10 permuting MNIST tasks at the end of training. *Taken from Figure 4 in Zenke, Poole, and Ganguli [72]

Types of superposition

To compare the different formulations of parameter superposition, we train two hidden layers with 256 units on 50 tasks (1000 steps per task). In Figure 5.6, we see that rotational superposition (*pspRotation*) performs consistently better than other variations of superposition. As rotational superposition is the broadest class of unitary transformations among those tested, it should be expected to make the most use of the available capacity. Furthermore, rotational introduces significantly more parameters than other superposition methods with each additional context (Table 5.1). Surprisingly, using powers of a single context (*pspOnepower*) performs at the same level if not better than using independent context vectors. Binary superposition (*pspBinary*) does not perform as well as others likely because it is the most constrained unitary transform among those tested.

Superposition does not accumulate learning constraints like most continual learning methods Kirkpatrick et al. [32]. To verify previous models stored in superposition do not hinder learning subsequent tasks, we compute the average accuracy across all tasks after training. To compare to previous work, we limit to training on 10 tasks and match network architectures. Results are shown in Table 5.2.

Output Interference

Output interference occurs when there is also a shift in the output (e.g. label) distribution of the training data. For example, this occurs when transitioning from one classification task to another. The incremental CIFAR (iCIFAR) dataset [51, 72] is a variant of the CIFAR dataset [33] where the first task is the standard CIFAR-10 dataset and subsequent tasks are formed by taking disjoint subsets of 10 classes from the CIFAR-100 dataset.

Even for continual learning methods, the outputs of the last layer are normally modularized with separate distinct parameters for each task (i.e. multi-head network) to prevent interference due to the large output shift [72]. To demonstrate the robustness of our approach, we learn on iCIFAR using *a single output layer* using superposition avoiding the need for a network with multiple output heads.

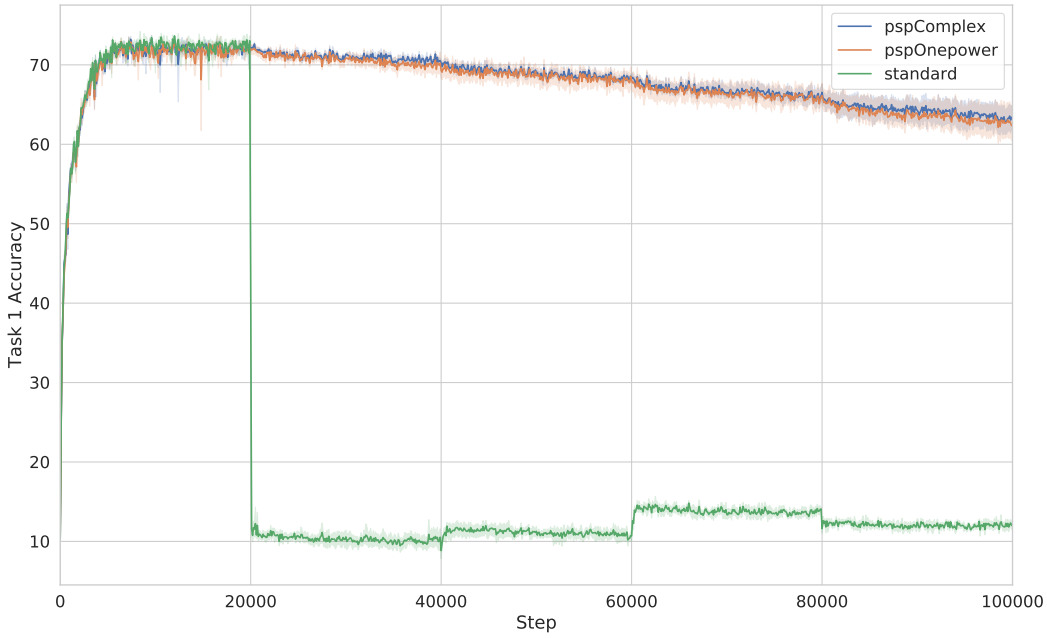

Figure 5.7: iCIFAR task 1 (CIFAR-10) accuracy on test set as a function of training step. All networks use a single output layer across all 5 tasks.

We train 6 layer convolutional networks using the superposition formulation in Equation 5.10 for each convolution layer and Equation 5.9 for every fully-connected layer. In Figure 5.7, we see a surprisingly small degradation in performance despite the drastic change in output interference. This demonstrates the ability to implement modular neural networks without instantiating new parameters for new modules.

Continuous Domain Shift

Most methods including those proposed in continual learning are formulated where the distribution shift is discrete (e.g. permuting MNIST, iCIFAR). But this may be a poor reflection of distribution shift which occurs naturally in online data gathered from the real world. For example, day gradually becomes night and summer gradually becomes winter. To simulate a continuous domain shift, we propose *rotating-MNIST* and *rotating-FashionMNIST* which are variants of the original MNIST and FashionMNIST [69] datasets. At each step of training, a two-dimensional rotation is applied to the input images of the dataset. After a sufficient amount of time (i.e steps) has passed, one revolution is completed and the input distribution will return to the starting distribution and another cycle begins. Figure 5.8 shows examples from these two rotating datasets as a function of time.

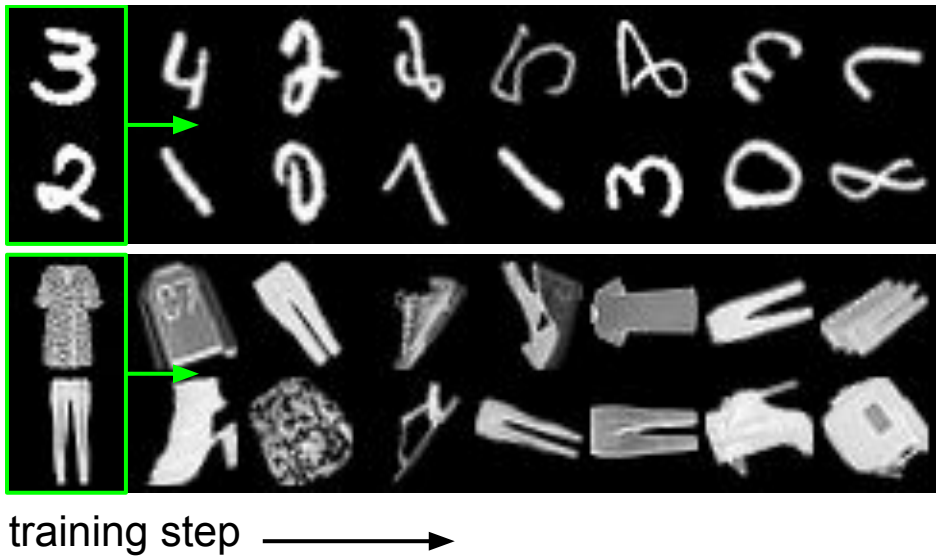

Figure 5.8: Samples of rotating-MNIST (top) and rotating-FashionMNIST (bottom) datasets. At each training step, the training distribution (green box) shifts by counter-clockwise rotation.

First, we compare the performance of models utilizing context vectors similar to previous experiments. After every 100 steps of training, we transition to the next context. Therefore, the context transition speed is 10 transitions per cycle where a cycle is one full revolution (1000 steps) of the input dataset. In Figure 5.9, the oscillations in model accuracy are significantly reduced using superposition for both rotating-MNIST and rotating-FashionMNIST.

In Figure 5.10, we compare the types of superposition for the rotating-FashionMNIST dataset (top row). *pspRotation* has the highest peak performance but also shows largest performance drops when other contexts are trained. Again, this is likely because this form of superposition has by far the most parameters. Similar to experiments on permuting MNIST, *pspComplex* and *pspOnePower* show the most stable behavior while *pspBinary* performs slightly worse.

Context functions

For continuous domain shift, the generation of context information $c(k)$ becomes more important to avoid abrupt changes in model performance during learning. We compare various methods for shifting between contexts.

pspFast refers to a context transition speed of 1000 transitions per cycle. This is effectively storing 1000 models in superposition and we can see the learning performance has noticeably deteriorated. If we incorporate more prior knowledge into these 1000 contexts by taking a mixture of three contexts (*pspFastLocalMix*) described in Equation 5.7, we notice an improvement. By further incorporating a slower rate of context changes to 10 transitions

per cycle (*pspLocalMix*), we notice a slight improvement over *pspComplex* which uses the same rate of context changes.

5.7 Related Work

As an online memory for parameters, superposition reduces interference in a fundamentally different way than previous methods. It does not explicitly constrain the learning of any model within the superposition. Most methods in continual learning utilize a constraining loss function [32]. Others freeze and grow parts of the network [64, 56, 72] to actively preserve previous models during training. As a consequence, the ability to learn new tasks becomes more and more limited as these loss constraints and computational costs accumulate. By acting as a memory, superposition provides control in how parameter capacity is allocated at any given moment during learning.

Therefore, like any other memory framework [15, 68], parameter superposition requires a controller in the form of a context function to store and recall memories (i.e. parameters). Sukhbaatar, Weston, Fergus, et al. [60] developed a differentiable form of the slot-based memory presented in Weston, Chopra, and Bordes [68] allowing the controller to be learned without direct supervision. In multiple forms of superposition we describe, the context selection function is also differentiable and learnable via backpropagation.

5.8 Discussion

By making use of redundancies already present in neural network models, we introduce a memory framework which operates on their parameters. Entire neural networks are memories which can be stored or recalled with context information which is far smaller in size than the network itself. When using powers of a context vector, that information can be as small as a single scalar value. Furthermore, model parameters can be retrieved in a discrete or continual fashion which accommodates for different types of distribution shift during training.

The flexibility of this approach in addressing learning interference opens the door to many new *online* applications. This is particularly useful in domains where memory resources are low both during training and at inference time. On the flip side, much larger modular networks can be trained with the same amount of memory resources [59]. In future work, we can take advantage of the continuity of superposition and define objectives to learn the context functions which control the parameter memory.

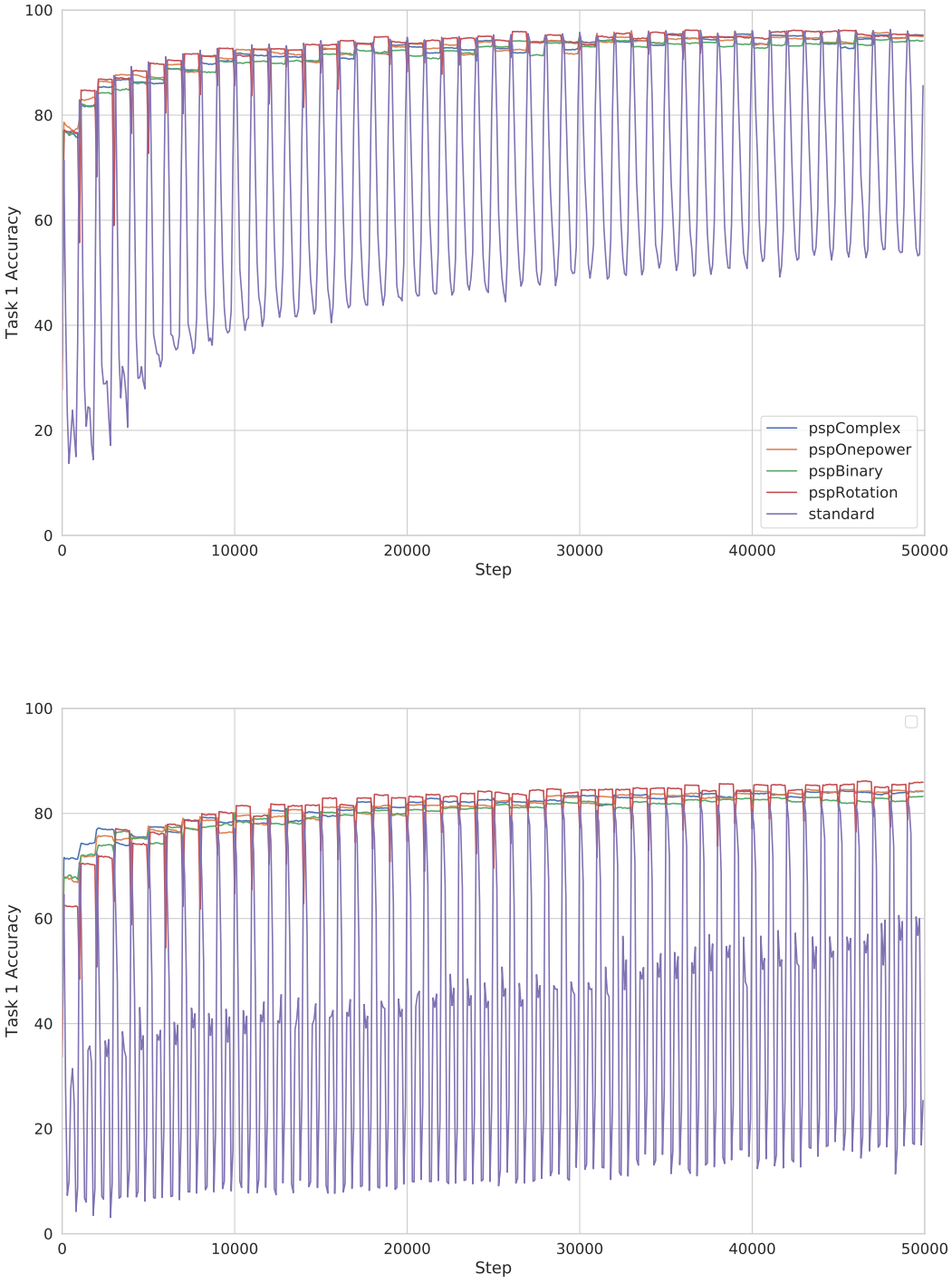

Figure 5.9: rotating-MNIST (top) and rotating-FashionMNIST (bottom) test accuracy at angle 0s a function of training step. One full rotation occurs every 1000 steps.

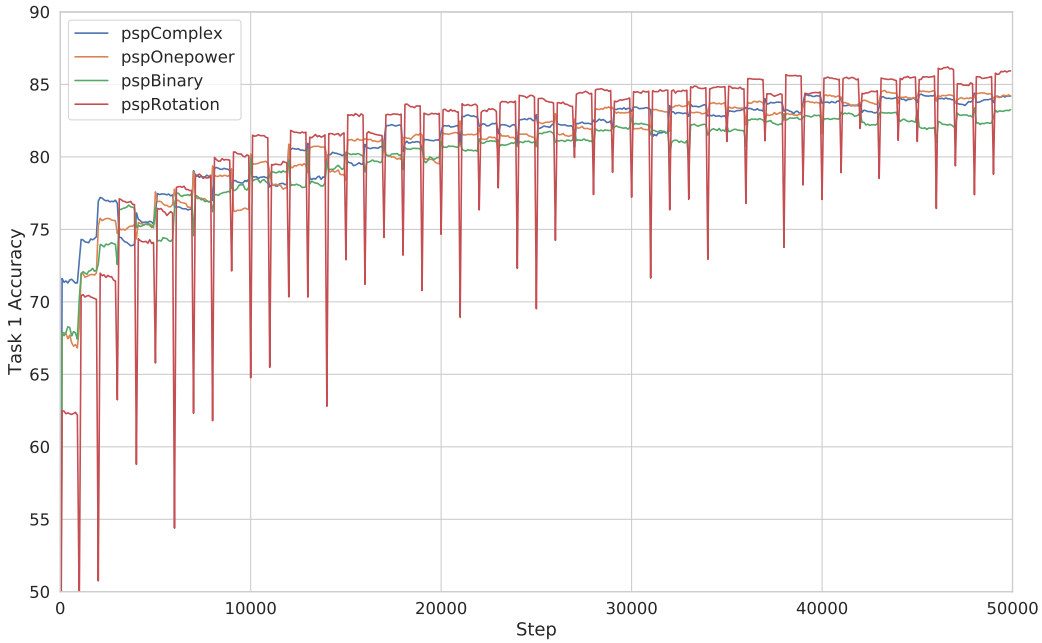

Figure 5.10: Closer comparison of each form of parameter superposition on the rotating-FashionMNIST task at angle 0

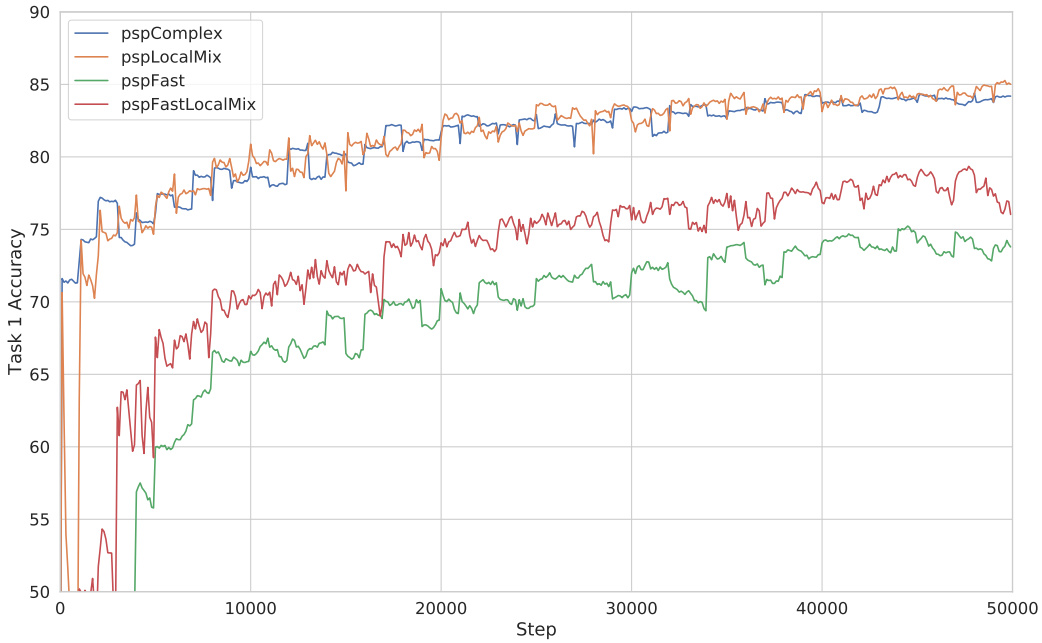

Figure 5.11: Comparison of different context selection functions on the rotating-FashionMNIST task at angle 0

Chapter 6

Learning from the past

In the previous chapters, we demonstrated the ability of simple learning algorithms to adapt internal variables from data. We showed that this learning can be guided in various ways to prevent previously acquired adaptations from being forgotten. In this chapter, we discuss future work which moves beyond simply preserving what has been learned towards reusing that knowledge to improve learning in the future. From this perspective, continual learning can serve as a benefit for a learning system over traditional i.i.d-based learning.

6.1 Temporal Overfitting

Currently, research focuses on addressing catastrophic forgetting by treating it as a memory problem. Most work in the past has treated this as a write-only memory which perpetually accumulates knowledge as it arrives. This strongly biases learning to occur only within a local time window. This greedy strategy will not scale as more and more data becomes available to a learning system. As mentioned in the previous chapter, we refer to this behavior as *temporal overfitting*, where a model overfits to the data it is currently presented, generalizing poorly to data at other times.

A mechanism to retrospectively consolidate previously learned knowledge will make more effective use of the finite capacity of a model. And more importantly, this consolidation will support knowledge transfer between different time windows both forwards and backwards. This is not unlike curriculum learning where a model first learns simple skills and then composes these to learn more complex skills.

Learning Causality

Forgetting occurs because knowledge acquired in the past cannot be applied to data in the present. In gradient descent learning, the gradient encourages forgetting because the utility of what was learned before does not help in adapting to information in the present. In many online learning settings, this makes sense if future data has no relationship with past data.

But in many real world environments, there is some degree of causal relationship between consecutive events. If there is an underlying cause which makes the current data a result of past data, learning the causal relationship enables knowledge to transfer from the past towards data in the future. This increases the utility of past knowledge which will naturally reduce the degree of forgetting.

Temporal Knowledge Transfer

If past knowledge is useful for the current task, even temporally local optimization methods like gradient descent would favor preserving previously acquired knowledge. But in continual learning, this becomes a chicken and the egg situation. At the time of knowledge acquisition, it is unknown how or if it will be applied in the future. An episodic memory can remedy this issue by storing past data or experiences for retrospective consolidation. Given a finite memory capacity, determining when to store past experiences is an open research direction. Unlike simple reservoir sampling, this determination is likely dependent on what has already been learned in the past.

6.2 Towards Natural Learning

While current learning algorithms have already demonstrated compelling abilities in using data to accomplish tasks and objectives, there are still many aspects which distinguish it from a more ideal system. These algorithms still require tedious curation and manual specification of the learning signal. In the future, we hope to relax these constraints to bring artificial learning systems closer to counterparts which we observe in nature. To make progress towards this, we must understand how every natural system learns in an inherently continual environment.

Bibliography

[1] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. "Multiple object recognition with visual attention". In: *arXiv preprint arXiv:1412.7755* (2014).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[3] Frédéric Bastien et al. "Theano: new features and speed improvements". In: *arXiv preprint arXiv:1211.5590* (2012).

[4] Pietro Berkes, Richard E Turner, and Maneesh Sahani. "A structured model of video reproduces primary visual cortical organisation". In: *PLoS computational biology* 5.9 (2009), e1000495.

[5] Charles F Cadieu and Bruno A Olshausen. "Learning intermediate-level representations of form and motion from natural movies". In: *Neural computation* 24.4 (2012), pp. 827–866.

[6] Christine A Curcio and Kimberly A Allen. "Topography of ganglion cells in human retina". In: *Journal of Comparative Neurology* 300.1 (1990), pp. 5–25.

[7] Jeffrey Dean et al. "Large scale distributed deep networks". In: *Advances in neural information processing systems*. 2012, pp. 1223–1231.

[8] David Eigen, Dilip Krishnan, and Rob Fergus. "Restoring an Image Taken through a Window Covered with Dirt or Rain". In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, pp. 633–640.

[9] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Training pruned neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[10] Wilson S Geisler and Lawrence Cormack. "Models of overt attention". In: *Oxford handbook of eye movements* (2011), pp. 439–454.

[11] Ian J Goodfellow et al. "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *arXiv preprint arXiv:1312.6211* (2013).

[12] Ian J Goodfellow et al. "Maxout networks". In: *Proceedings of the 30th International Conference on Machine Learning*. ACM. 2013, pp. 1319–1327.

[13] Ian J Goodfellow et al. "Pylearn2: a machine learning research library". In: *arXiv preprint arXiv:1308.4214* (2013).

[14] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2672–2680.

[15] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines". In: *arXiv preprint arXiv:1410.5401* (2014).

[16] Henry Gray. *Gray's anatomy*. 1910.

[17] Karol Gregor et al. "DRAW: A recurrent neural network for image generation". In: *arXiv preprint arXiv:1502.04623* (2015).

[18] David B Grimes and Rajesh PN Rao. "Bilinear sparse coding for invariant vision". In: *Neural computation* 17.1 (2005), pp. 47–73.

[19] Ralph Gross et al. "Multi-pie". In: *Image and Vision Computing* 28.5 (2010), pp. 807–813.

[20] Stephen Grossberg. "Competitive learning: From interactive activation to adaptive resonance". In: *Cognitive science* 11.1 (1987), pp. 23–63.

[21] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[22] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[23] Awni Hannun et al. "Deep speech: Scaling up end-to-end speech recognition". In: *arXiv preprint arXiv:1412.5567* (2014).

[24] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. "Transforming auto-encoders". In: *Artificial Neural Networks and Machine Learning–ICANN 2011*. Springer, 2011, pp. 44–51.

[25] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[26] Laurent Itti and Christof Koch. "A saliency-based search mechanism for overt and covert shifts of visual attention". In: *Vision research* 40.10 (2000), pp. 1489–1506.

[27] Max Jaderberg et al. "Decoupled neural interfaces using synthetic gradients". In: *arXiv preprint arXiv:1608.05343* (2016).

[28] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. "Spatial transformer networks". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2008–2016.

[29] Pentti Kanerva. "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors". In: *Cognitive Computation* 1.2 (2009), pp. 139–159.

[30] Diederik P Kingma et al. "Semi-supervised learning with deep generative models". In: *Advances in Neural Information Processing Systems*. 2014, pp. 3581–3589.

[31] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[32] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* (2017), p. 201611835.

[33] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[35] Dharshan Kumaran, Demis Hassabis, and James L McClelland. "What learning systems do intelligent agents need? Complementary learning systems theory updated". In: *Trends in cognitive sciences* 20.7 (2016), pp. 512–534.

[36] Hugo Larochelle and Geoffrey E Hinton. "Learning to combine foveal glimpses with a third-order Boltzmann machine". In: *Advances in neural information processing systems*. 2010, pp. 1243–1251.

[37] Quoc V Le. "Building high-level features using large scale unsupervised learning". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 8595–8598.

[38] Yann LeCun and Corinna Cortes. *The MNIST database of handwritten digits*. 1998.

[39] Chunyuan Li et al. "Measuring the intrinsic dimension of objective landscapes". In: *arXiv preprint arXiv:1804.08838* (2018).

[40] Qianli Liao, Joel Z Leibo, and Tomaso A Poggio. "How important is weight symmetry in backpropagation?" In: *AAAI*. 2016, pp. 1837–1844.

[41] Timothy P Lillicrap et al. "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nature communications* 7 (2016), p. 13276.

[42] Christiane Linster. *Neural Associative Memories*. `https://courses.cit.cornell.edu/bionb330/readings/Associative\%20Memories.pdf`.

[43] Zhuang Liu et al. "Rethinking the Value of Network Pruning". In: *arXiv preprint arXiv:1810.05270* (2018).

[44] James L McClelland, Bruce L McNaughton, and Randall C O'reilly. "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." In: *Psychological review* 102.3 (1995), p. 419.

[45] Roland Memisevic and Geoffrey E Hinton. "Learning to represent spatial transformations with factored higher-order boltzmann machines". In: *Neural Computation* 22.6 (2010), pp. 1473–1492.

[46] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[47] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. "Recurrent models of visual attention". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2204–2212.

[48] Bruno A Olshausen et al. "Bilinear models of natural images". In: *Electronic Imaging 2007*. International Society for Optics and Photonics. 2007, pp. 649206–649206.

[49] VH Perry, R Oehler, and A Cowey. "Retinal ganglion cells that project to the dorsal lateral geniculate nucleus in the macaque monkey". In: *Neuroscience* 12.4 (1984), pp. 1101–1123.

[50] Tony A Plate. "Holographic reduced representations". In: *IEEE Transactions on Neural networks* 6.3 (1995), pp. 623–641.

[51] Sylvestre-Alvise Rebuffi et al. "icarl: Incremental classifier and representation learning". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 5533–5542.

[52] Scott Reed et al. "Learning to Disentangle Factors of Variation with Manifold Interaction". In: *Proceedings of The 31st International Conference on Machine Learning*. ACM. 2014, pp. 1431–1439.

[53] Salah Rifai et al. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 833–840.

[54] Salah Rifai et al. "Disentangling factors of variation for facial expression recognition". In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 808–822.

[55] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), p. 533.

[56] Andrei A Rusu et al. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[57] Ruslan Salakhutdinov and Geoffrey E Hinton. "Learning a nonlinear embedding by preserving class neighbourhood structure". In: *International Conference on Artificial Intelligence and Statistics*. 2007, pp. 412–419.

[58] Benjamin Scellier and Yoshua Bengio. "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation". In: *Frontiers in computational neuroscience* 11 (2017), p. 24.

[59] Noam Shazeer et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer". In: *arXiv preprint arXiv:1701.06538* (2017).

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

[61] J. Susskind, A. Anderson, and G. E. Hinton. *The Toronto Face Database*. Tech. rep. University of Toronto, 2010.

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

[63] Joshua B Tenenbaum and William T Freeman. "Separating style and content with bilinear models". In: *Neural computation* 12.6 (2000), pp. 1247–1283.

[64] Alexander V. Terekhov, Guglielmo Montone, and J. Kevin O'Regan. "Knowledge Transfer in Deep Block-Modular Neural Networks". In: *Living Machines 2015: Biomimetic and Biohybrid Systems*. 2015, pp. 268–279.

[65] David C Van Essen and Charles H Anderson. "Information processing strategies and pathways in the primate visual system". In: *An introduction to neural and electronic networks* 2 (1995), pp. 45–76.

[66] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *The Journal of Machine Learning Research* 11 (2010), pp. 3371–3408.

[67] Li Wan et al. "Regularization of neural networks using dropconnect". In: *International Conference on Machine Learning*. 2013, pp. 1058–1066.

[68] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory networks". In: *arXiv preprint arXiv:1410.3916* (2014).

[69] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[70] Kelvin Xu et al. "Show, attend and tell: Neural image caption generation with visual attention". In: *arXiv preprint arXiv:1502.03044* (2015).

[71] Matthew D Zeiler. "ADADELTA: An adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[72] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual learning through synaptic intelligence". In: *arXiv preprint arXiv:1703.04200* (2017).