Exploring Topological Structures in the Multinomial Machine

Linghai Liu

Department of Applied Mathematics

Brown University

Thesis Advisor

Stuart Geman

Reader

Basilis Gidas

In partial fulfillment of the requirements for the degree of

Bachelor of Science in Applied Mathematics - Computer Science

April 28, 2023

Abstract

In this thesis, we define and build a network architecture termed the "multinomial machine" with types of reusable blocks. With computational experiments, we demonstrate that with a shallow network of only several hundred parameters, the network can perform well in binary classification tasks on MNIST and learn interpretable kernels that capture features. Then we introduce "perturbation" to enhance topological structures and point out current bottlenecks of the network. Finally, we discuss the idea of a top-down structure that can potentially promote topological structures while maintaining good performance.

Acknowledgements

I would like to express my gratitude to my advisor, Professor Stuart Geman, who inspired and introduced me to explore *science* and provided help and guidance spanning my undergraduate studies. I would like to thank my thesis reader Professor Basilis Gidas, whose teaching and advising helped me a lot while learning statistics.

Thank you to those who also participated in this project - Xinjie (Jayden) Yi, Jose Urruticoechea, Chen (Danny) Li, Derik Kauffman, and Jiayuan (Jay) Sheng. I really enjoyed working with you all.

Lastly, I would like to express my deepest gratitude to my parents for their love and support, and Yuan Pu for her love and company.

Contents

Chapter 1

Introduction

Humans have been relying on vision for a long time. Visual perception is the process of acquiring knowledge from objects surrounding us based on physical properties of light [21]. We perform it continuously, effortlessly, and efficiently without even noticing it. However, to build a machine or a system that achieves the same level of performance is by no means easy.

Vision, as it appears, is a task that involves information processing. As humans, reflected light stimulates the cone cells and rod cells on our retina, and we use this information to think, infer, and act. But it is more than that. If a system, e.g., our brain, can understand the relative location among many objects and backgrounds, then it must be capable of representing this information in some form. Hence, what we want eventually is a machine that not only processes information but also represents it in an appropriate manner.

With this duality in mind, Marr outlined the three levels necessary for one to claim a complete understanding of an information processing device [19]: its computational theory, the representation and algorithm that it uses, and how to implement this device on existing hardware. Computational theory involves figuring out the goals of the entire computation and justifying the approach. The representation and the algorithm consider how we represent the input and output, and how the transformation from input to output is carried out. Finally, we realize the entire process physically.

One famous attempt was the Summer Vision Project at MIT [22], where the summer workers aimed to segment a given image into the smallest building blocks such as balls, bricks, and cylinders by July, and extend this line of work into August to an understanding the background texture and noise. With these components and knowledge of the relationship between the "building blocks" and the background (or merely noise), we could use compositional rules to group them into meaningful scenes.

Heuristics as in the summer project are helpful, but they elicit important problems to solve. First of all, we have to figure out whether we have structures that are sensitive to the "building blocks" of the scene that we see. Fortunately, we do have specialized cells that are very sensitive to some shapes and luminosity changes. For example, Wiesel and Hubel discovered simple cells and complex cells [8, 9] in the cat's visual cortex. Both types of cells respond to features such as oriented line segments and corners, but complex cells possess spatial invariance in the input to some extent.

Once we have broken down an image into supposedly meaningful parts, how does the visual system, be it biological or artificial, know how to group them together using some learned relationships (the "binding problem")? It is not feasible to encode all the relational information into some units because there is an infinite amount of such relationships, and we are given limited capacity either biologically or computationally.

How to detect whether such features could be recognized when they are presented with a noisy background? This leads us to the *dilemma of invariance versus selectivity.* If a visual system aims to recognize a pattern whenever it appears, then it would suffer from high false alarm rates, possibly because of noisy surroundings and contaminated inputs. On the other hand, if the system is very selective, i.e., it only responds when it sees almost exactly the same visual cue, then it would seldom detect such a feature.

A possible solution to this dilemma lies in the very nature of the visual cells in the brain. Overlapping receptive fields carry information, which is passed through a hierarchy of neurons and used for inference [6]. To build this hierarchy, which consists of disjunctions of conjunctions, we start with a "multinomial machine" that mimics an idealized biological neuron, then embed this discrete-state system into a continuous-state system (§3.1) wherein current toolboxes such as autograd and deep learning models can be exploited (§4.1).

Chapter 2

Related Work

2.1 Nonlinearity of Dendrites

Dendrites are the tree-like structure of a biological neuron that receive inputs (spikes) from upstream neurons. These inputs are integrated locally and may result in various non-linearities, such as dendritic spikes [17, 4]. Typically, these non-linearities can be described as super-additive, in the sense that a temporal coincidence of signals that arrive within the same region of the dendrite will evoke a larger response than the sum of the responses evoked by either stimulus alone. In the case of dendritic spikes, these responses might remain local in the dendrite, or they might trigger outputs from the neuron to which the dendrite belongs. These outputs are, in turn, transmitted to downstream neurons as presynaptic inputs, where they too can evoke a nonlinear response. The structure of the hierarchy of biological neurons, together with models that include some form of super-additivity in their dendrites, e.g., the Rectified Linear Units (ReLU) and the softmax function defined in §3.2.3, has inspired and contributed to the prosperity of engineered systems and artificial intelligence in general.

2.2 Advances in AI

In the past decade, huge advances have taken place in machine learning. Among these, deep learning [15] is widely used as a universal approximation to functions that can perform tasks such as image classification. Convolutional machine learning methods struggled before the deep learning era because extreme expertise was needed to build feature extractors. However, deep learning models can be fed with raw data, and with their numerous layers, they automatically discover a suitable abstract representation.

A convolution layer is parametrized by a filter (or usually called a "kernel") $\mathcal{F} \in \mathbb{R}^{k \times k}$ with a fixed kernel size k upon initialization. Given an input 2D image $\mathcal{I}$,

$$\text{2D convolution:} \quad \mathcal{H}(m,n) := \sum_{k,\ell} \mathcal{F}(k,\ell)\mathcal{I}(m-k,n-\ell)$$

for each coordinate (m,n) in the resulting image $\mathcal{H}$. 2D convolution has the merit of *translational equivariance* during feature extraction, which is one of the desired properties when building feature extractors.

Convolutional neural networks (CNNs) are based on many such layers. Training a CNN amounts to optimizing model parameters such as the kernels $\mathcal{K}$'s. This requires *huge* datasets for complex downstream tasks. With the advent of ImageNet [5], we observe an explosion of CNNs - AlexNet[13, 14], ZFNet[27], GoogLeNet[25], VGGNet[24], ResNet[7] and so forth - that perform better and better in image classification tasks and beyond.

Rather than feature engineering with deep learning, another line of work called neuromorphic computing is inspired by the structure in our brain. Neurons are connected by about 10^{16} synapses and they perform complex operations way more efficiently and consume much less energy than computational machines [20]. To mimic what neurons are doing, we can choose to design our models so that the computations are local and we can try to incorporate the underlying physics taking place right at the synapses.

Spiking Neural Networks (SNNs) have differential equations modeling the underlying flows of ions and membrane potential. When the integrated potential over time exceeds the threshold of a unit, a spike will occur and will be passed on to units in the next layer. Hence, the input of a unit would be spike trains (time series of 0's and 1's), as shown in Figure 2.1. One well-known example is the Izhikevich model [11] that is both biologically plausible and computationally efficient. It can also model the firing patterns of cortical neurons with suitable parameters, as outlined in [10].

Training SNNs is quite challenging. SNNs encounter gradient vanishing or explosion because spike trains are non-differentiable signals. Some attempts include converting artificial neural networks (from the first line of work) to SNNs, but this process increases computational complexity because of the large number of time steps. Other work includes

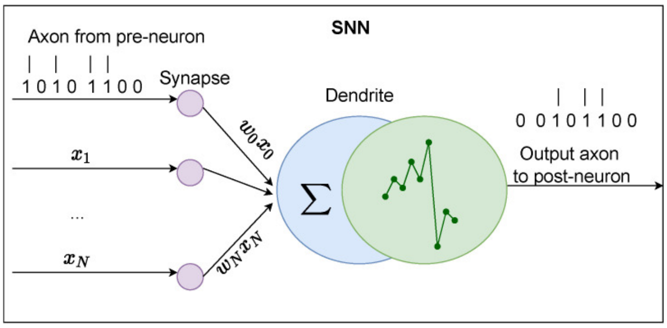


Figure 2.1 Demonstration of an SNN [26]

searching for suitable SNN architectures using meta-learning techniques [12].

2.3 Functional Common Input

A solution to the *dilemma of invariance versus selectivity* mentioned in Chapter 1 means more than just building up feature detectors and then using them to capture the features in a given image. *How* we capture them matters as well. For example, if we are given two units that can learn to detect vertical lines, then in both case (a) and (b) in Figure 2.2, the two units will report a detection. However, the underlying signals are structured completely differently. At downstream units (neurons), we have no idea of the exact location of the detected lines. We lost this information during the bottom-up forward propagation.

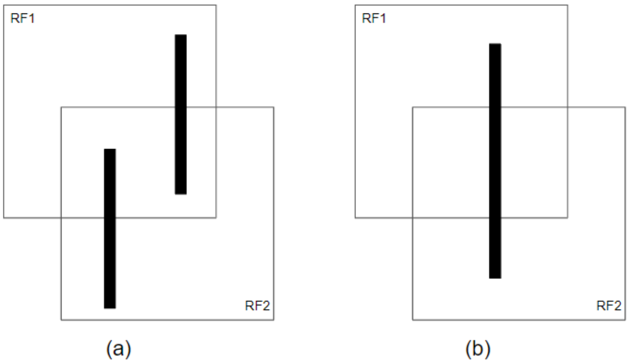


Figure 2.2 Demonstration of two line detectors. RF1 and RF2 denote the receptive field, the region from which a unit receives input from the entire scene, of unit 1 and 2 respectively.

To capture this information, Geman proposed functional common input (fci) and used it to obtain lower false alarm rates in a simple face-detection algorithm [6]. The idea was formulated as follows: take two feature detectors N_1 and N_2 with kernels $\mathcal{K}_1$

and $\mathcal{K}_2$ and an image $\mathcal{I}$. An image patch $\mathcal{I}_\lambda$ has a top-left corner at λ and the same size as the kernel to be convolved with (by taking the inner product). When taking the convolution, we align the top-left corner of the kernels with each image patch within the image, thereby modeling available inputs to feature detectors N_1 and N_2. Let λ_1 and λ_2 be two image locations, and let $\langle \mathcal{K}_1, \mathcal{I}_{\lambda_1} \rangle$ and $\langle \mathcal{K}_2, \mathcal{I}_{\lambda_2} \rangle$ designate the correlations (inner products) between the image patches and respective templates. The functional common input (fci) to feature detectors N_1 and N_2 is the product of three components, two of which are their respective correlations. The third amounts to a proxy for the proper alignment of the parts:

$$\sum_{i \in \mathcal{I}_{\lambda_1} \cap \mathcal{I}_{\lambda_2}} (\mathcal{K}_{1,i} \cdot \mathcal{K}_{2,i})$$

where, for each i in the intersection of the image patches, $\mathcal{K}_{1,i}$ and $\mathcal{K}_{2,i}$ are the corresponding entries of $\mathcal{K}_1$ and $\mathcal{K}_2$, respectively. In words, we use a sum of products, one for each location i in the common input. Each product is between the partial derivatives of the inputs to N_1 and N_2 with respect to the pixel strength at location i. It is this term that recovers the relative positioning of the parts that would otherwise be lost by virtue of the invariance of the individual feature detectors. All together,

$$\text{fci} = \langle \mathcal{K}_1, \mathcal{I}_{\lambda_1} \rangle \cdot \langle \mathcal{K}_2, \mathcal{I}_{\lambda_2} \rangle \sum_{i \in \mathcal{I}_{\lambda_1} \cap \mathcal{I}_{\lambda_2}} (\mathcal{K}_{1,i} \cdot \mathcal{K}_{2,i})$$

By varying thresholds, an ROC curve is swept out. The ROC performance is substantially improved by including the proxy for common input.

The success of this idea is also supported by work in neurophysiology. Reinagel and Reid showed that a large portion of the information needed to predict the firing of cortical neurons lies in the precise timings of spikes in pairs of neurons extending from the lateral geniculate nucleus (LGN), and that this precise timing typically reflects common input to the LGN cells from the retina [23]; Alonso et al. found that synchronized thalamic spikes from the LGN promote input to cortical cells [1]; Azouz and Gray demonstrated the contribution to selectivity by synchronous synaptic inputs [2]. Since common input promotes synchrony, it is possible to construct a top-down process to infer the input given the firing patterns of downstream neurons, thus leading to a reusable generative framework. The structure in Figure 3.2 derives from the same vein of thought, and we propose a measure of it in §3.3.

2.4 The Sufficiency Trick

In Chang et al [3], the authors proposed a generative method to learn features. The idea is simple: if we define some distributions on low-dimensional features S that are specific to image categories and assume that there exists a universal (background) distribution on the values on the pixels y given the features, where y denotes an observation of input pixels, then we will have a category-specific distribution on pixels that is a function of the features themselves:

$$\mathbb{P}_Y(y) = \mathbb{P}_S(s(y;\phi);\theta)\ \mathbb{P}^0_Y(y \,|\, S = s(y;\phi)) \tag{$*$}$$

where $\mathbb{P}^0_Y(\cdot \,|\, S)$ is the background distribution (with no parameters), S can be the correlation of a kernel (with parameters ϕ determining it to be estimated) and the image, and θ parametrizes the distribution of feature S. If we are given many independent examples $y_1, y_2, \ldots, y_N$ and happen to know the kernels ϕ, then the feature S is sufficient for θ.

We will present in §3.3 a formulation of such S dependent on the hierarchical structure of our network outlined in §3.1 and §3.2. This S will then be used to “perturb” the original hierarchy of probability distributions of states of units and the conditional probability distribution over the categories we train the network to classify.

Chapter 3

Methods

3.1 The Multinomial Machine

Let α be an artificial neuron with n^α many dendrites. Let

$$X^\alpha \in \{0, 1, \ldots, n^\alpha\}$$

be a multinomial random variable with a single trial, with $\mathbb{P}(X^\alpha = i) = \varepsilon_i^\alpha$ for all $i \in \{0, 1, \ldots, n^\alpha\}$.

We also define

$$x_i^\alpha := \mathbb{1}_{X^\alpha = i}$$

be an indicator of the event $\{X^\alpha = i\}$ for all $i \in \{0, 1, \ldots, n^\alpha\}$.

Since X^α is defined to be a multinomial random variable, we have

$$\sum_{i=0}^{n^\alpha} x_i^\alpha = 1 \tag{3.1}$$

Note that in order to determine the values of $n^\alpha + 1$ elements that sum up to 1, we only need n^α of them, i.e., for each neuron α, there is n^α degrees of freedom. Here we use the last n^α of them, so

$$x_0^\alpha = \sum_{i=1}^{n^\alpha} x_i^\alpha$$

X^α indicates which dendrite has a number 1, which means *that* dendrite is firing (a spike occurs) if $X^\alpha \neq 0$. We deem $\{X^\alpha = 0\}$ as the event where neuron α is *not* firing.

Hence, it would be interesting to investigate the probability that neuron α fires, and we do this by embedding this multinomial machine defined above into the continuous space $[0,1]$ in the following way:

Since the random variables x_i^α's satisfy (3.1), they lie on (and actually are vertices of) an n^α-dimensional probability simplex

$$\mathcal{D}^{n^\alpha} := \{v \in \mathbb{R}^{n^\alpha+1} : \sum_{i=0}^{n^\alpha} v_i \text{ and } v_i \geq 0 \text{ for } i = 0, 1, \ldots, n^\alpha\}$$

Currently, we can write the probability density function of X^α as: for $x^\alpha \in \{0, 1, \ldots, n^\alpha\}$,

$$\begin{aligned} \mathbb{P}(x^\alpha) &= \prod_{i=0}^{n^\alpha} (\varepsilon_i^\alpha)^{x_i^\alpha} \\ &= \exp\left\{\sum_{i=0}^{n^\alpha} x_i^\alpha \log \varepsilon_i^\alpha\right\} \end{aligned} \tag{3.2}$$

With (3.2), we can extend $\mathbb{P}$ from the vertices of $\mathcal{D}^{n^\alpha}$ to all its elements by modeling each x_i^α as the probability that dendrite i *is* firing and x_0^α as the probability that neuron α is *not* firing.

Hence, it is natural to introduce $z^\alpha \in \{0,1\}$ for neuron α as follows:

$$z^\alpha := \begin{cases} 0 & \text{if } x_0^\alpha = 1 \\ 1 & \text{if } x_0^\alpha = 0 \end{cases}$$

in the discrete case, and

$$z^\alpha := \sum_{i=1}^{n^\alpha} x_i^\alpha = 1 - x_0^\alpha$$

when we embed the multinomial machine into the continuous space $[0,1]$ as above. In the discrete case, z^α indicates whether neuron α is firing. When embedded, z^α stands for the probability that neuron α fires.

3.2 The Network

Our network aims to solve image classification tasks. In this project, we use the MNIST dataset, which contains images featuring digits 0 through 9. The strokes are white and the background is black. Our network takes in an image from the MNIST dataset and outputs a classification in $\{0, 1, \ldots, 9\}$ as the predicted label of the input image.

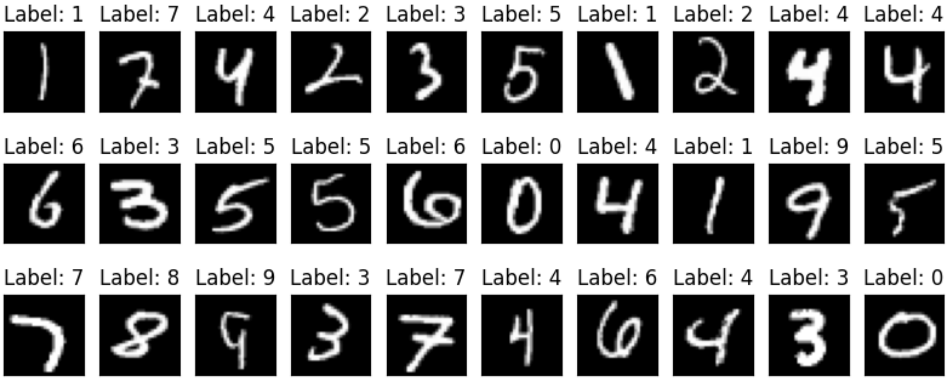


Figure 3.1 A subset of the MNIST dataset [16]

3.2.1 Units

Since we are involved in some vision tasks, we will naturally have three types of idealized neurons in our network:

- Pixel neurons: these neurons are actually pixels, which constitute an input layer to our network. Without loss of generality, we will work with gray-scale images, since each channel of a colored image can also be viewed as a layer of gray-scale image.

  Let $\mathcal{A}$ be the layer of pixel neurons (the bottom-most layer) and $\alpha \in \mathcal{A}$ be a specific pixel. For neurons to which α is connected, they observe z^α given $\omega \in \Omega$. As shorthand, we will refer to z^α as "the output of α".

- Hidden neurons: let $\mathcal{B}$ be the set of hidden neurons and $\beta \in \mathcal{B}$ be a hidden neuron. Neuron β has n^β dendrites, and the level of activation of each dendrite drives neuron β to fire. For simplicity, we use the embedded $z^\beta \in [0, 1]$ as the output of β.

- Decision neuron(s): our network seeks to solve a classification task, so a decision neuron γ is in the topmost layer $\mathcal{C}$. Its output is a probability distribution over all the classes from which we can make a classification given $\omega \in \Omega$.

3.2.2 Weights

- For each hidden neuron $\beta \in \mathcal{B}$,

  $$W_i^\beta \in \mathbb{R}^{|\mathcal{A}|}$$

  is the weight vector for the i-th dendrite of β, where $i = 1, 2, \cdots, n^\beta$.

- For a decision neuron γ,

  $$W_i^\gamma \in \mathbb{R}^{|\mathcal{B}|}$$

  is the weight vector for the i-th dendrite of γ, where $i = 1, 2, \cdots, c$ and c is the number of classes we need to classify.

Note that when defining the weight W^γ, we used the notation $|\mathcal{B}|$, but this is not accurate. Here $\mathcal{B}$ is the last layer prior to the topmost layer (where neuron γ resides), but $\mathcal{B}$ can be more general - it can denote the set of all hidden neurons. In that case, we would need another symbol, e.g., $\mathcal{B}'$, to denote the last hidden layer.

3.2.3 Activation

- For each pixel $\alpha \in \mathcal{A}$, the activation is $z^\alpha \in [0, 1]$, the value of the pixel.

  From now on, denote $z^\mathcal{A}$ as the activation of layer $\mathcal{A}$.

- Each hidden neuron $\beta \in \mathcal{B}$ receives the activation from the previous layer, e.g., $z^\mathcal{A}$ from layer $\mathcal{A}$, as input.

  The softmax activation of dendrite i is defined as

  $$x_i^\beta := \frac{\exp\left\{\langle W_i^\beta, z^\mathcal{A}\rangle + b^\beta\right\}}{1 + \sum_{j=1}^{n^\beta} \exp\left\{\langle W_j^\beta, z^\mathcal{A}\rangle + b^\beta\right\}} = \frac{e^{\ell_i^\beta}}{1 + \sum_{j=1}^{n^\beta} e^{\ell_j^\beta}}$$

  where $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors, and

  $$\ell_i^\beta := \langle W_i^\beta, z^\mathcal{A}\rangle + b^\beta$$

  is defined as the linear readout of dendrite i for $i = 1, 2, \ldots, n^\beta$.

Additionally, we define

$$x_0^\beta := \frac{1}{1 + \displaystyle\sum_{j=1}^{n^\beta} \exp\left\{\langle W_j^\beta, z^{\mathcal{A}}\rangle + b^\beta\right\}} = \frac{1}{1 + \displaystyle\sum_{j=1}^{n^\beta} e^{\ell_j^\beta}}$$

and we interpret it as the probability that the hidden unit β is *not* firing.

Note that the vector

$$x^\beta = (x_0^\beta, x_1^\beta, \cdots, x_{n^\beta}^\beta)$$

is a distribution.

Then the *activation* of hidden unit β is defined as

$$z^\beta := 1 - x_0^\beta = \sum_{j=1}^{n^\beta} x_j^\beta = \frac{\displaystyle\sum_{j=1}^{n^\beta} e^{\ell_j^\beta}}{1 + \displaystyle\sum_{j=1}^{n^\beta} e^{\ell_j^\beta}}$$

which can be interpreted as the probability that the hidden unit β *is* firing.

- Similar to hidden neurons, a decision neuron $\gamma \in \mathcal{C}$ receives the activation from the previous hidden layer, $z^{\mathcal{B}}$, as input.

  The linear readout of γ is

  $$\ell^\gamma = \langle W^\gamma, z^{\mathcal{B}}\rangle + b^\gamma \in \mathbb{R}^c$$

  To obtain the activation of γ, we reuse the softmax function again:

  $$x_i^\gamma = \frac{e^{\ell_i^\gamma}}{1 + \displaystyle\sum_{j=1}^{c} e^{\ell_j^\gamma}}, \quad i = 1, 2, \cdots, c$$

  and

  $$x_0^\gamma = \frac{1}{1 + \displaystyle\sum_{j=1}^{c} e^{\ell_j^\gamma}}$$

  so that $x^\gamma = (x_0^\gamma, x_1^\gamma, \ldots, x_c^\gamma)$ forms a distribution.

Note that normally x^γ is used to perform some downstream tasks, such as image classification in our case. Hence, we do not need to define the activation of γ. But for completeness, we define it in the same manner as for hidden neurons:

$$z^\gamma = 1 - x_0^\gamma = \sum_{j=1}^{c} x_j^\gamma$$

3.2.4 Output

Given an input activation $z^{\mathcal{A}}$, layer $\mathcal{B}$ has an activation $z^{\mathcal{B}}$, which is passed on to the decision neuron γ so that we can obtain the vector x^γ.

x^γ is the output of the Network. Hence, it can be viewed that the Network is trying to model the conditional probability $\mathbb{P}(c \,|\, z^{\mathcal{A}})$ for each class c.

The classification of the given image underlying $z^{\mathcal{A}}$ is then

$$\hat{C} = \underset{i \in \{1,2,\cdots,c\}}{\arg\max}\, x_i^\gamma$$

3.2.5 Parameters

- For each hidden unit $\beta \in \mathcal{B}$, the parameters are elements of a fixed kernel $\mathcal{K}^\beta \in \mathbb{R}^{n_k \times n_k}$ and a bias term $b^\beta \in \mathbb{R}$, where n_k is the size of β's convolution kernel.

  This is different from what we mentioned in §3.2.2, where a weight vector of the hidden unit β is

  $$W_i^\beta \in \mathbb{R}^{|\mathcal{A}|} \quad \text{for } i = 1, 2, \cdots, n^\beta$$

  Since convolution is linear, these weights could be collected in a matrix $W^\beta \in \mathbb{R}^{n^\beta \times |\mathcal{A}|}$. This could be done by writing the kernel $\mathcal{K}^\beta$ into a Toeplitz matrix.

  Then, given an array of pixels $z^{\mathcal{A}}$ (an image from the pixel layer $\mathcal{A}$), we can reshape it into a 2D array $z'^{\mathcal{A}} \in \mathbb{R}^{n_r \times n_c}$ such that $n_r \times n_c = |\mathcal{A}|$, we have:

  $$\texttt{flatten}(\mathcal{K}^\beta * z'^{\mathcal{A}}) + b^\beta \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \langle W^\beta, z^{\mathcal{A}} \rangle + b^\beta \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

  where n_r is the number of rows and n_c is the number of columns of pixels in the

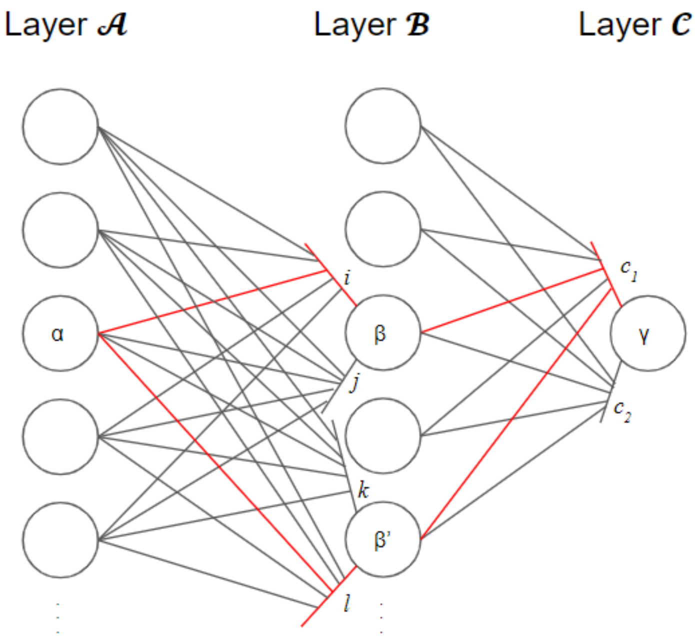


Figure 3.2 A cycle in red, starting from dendrite 1 of γ and ending at neuron α

image, $*$ denotes the 2D convolution operation, and the function `flatten` takes in an array of any shape and turns it into a column vector.

- For the decision neuron γ in the top-most layer, the parameters consist of a matrix $W^\gamma \in \mathbb{R}^{c \times |\mathcal{B}|}$ and a bias vector $b^\gamma \in \mathbb{R}^c$.

The number of parameters in the entire network is then:

$$|\mathcal{B}| \cdot \underbrace{(k^2 + 1)}_{\mathcal{K}^\beta, b^\beta} + \underbrace{(|\mathcal{B}| + 1)}_{W_i^\gamma, b_i^\gamma} \cdot c$$

3.3 Topology

Take an instance of our network with three layers: the bottom layer $\mathcal{A}$ of pixels, the hidden layer $\mathcal{B}$ of hidden units, and the top-most layer $\mathcal{C}$, which contains only a single decision neuron γ. This is the shallowest network we can find that can perform our MNIST classification task while possessing topological structures, i.e., *cycles*, that we are presenting here.

In Figure 3.2, starting from dendrite 1 of $\gamma \in \mathcal{C}$ and ending at pixel neuron $\alpha \in \mathcal{A}$, we can find many different paths through the hidden neurons in layer $\mathcal{B}$. We color two of those paths in red. The first path starts from dendrite 1 of γ, goes to neuron β, and arrives at neuron α via the i-th dendrite of β. Similarly, we have the other path passing through another neuron β'. We term this structure as a *cycle*, which can be useful in understanding the decision-making by γ.

Note that dendrite 1 of γ corresponds to the first class of digits we are trying to classify and dendrite i of β corresponds to the i-th location on which β performs the 2D convolution operation on $z^{\mathcal{A}}$.

More cycles imply more overlap by the hidden units in layer $\mathcal{B}$, which increases the functional common input in §2.3. This in turn suggests the hidden neurons have a lot of coincidences while detecting the features they are interested in, respectively. We encourage this phenomenon, as it can suggest better detection, per the face detection experiment in [6]. But calculating the number of cycles, or the strengths of these cycles, is not easy. Relying on the fact that taking a partial derivative involves traversing every possible path to the input, we formulate it in the following manner:

$$\frac{\partial \ell_i^\gamma}{\partial z^\alpha} = \sum_{\beta \in \mathcal{B}} \sum_{m=1}^{n^\beta} \sum_{k=1}^{n^\beta} \frac{\partial \ell_i^\gamma}{\partial z^\beta} \frac{\partial z^\beta}{\partial x_m^\beta} \frac{\partial x_m^\beta}{\partial \ell_k^\beta} \frac{\partial \ell_k^\beta}{\partial z^\alpha}$$

We calculate each part in order:

$$\begin{aligned}
\frac{\partial \ell_i^\gamma}{\partial z^\beta} &= \frac{\partial}{\partial z^\beta}\left(\sum_{s \in \mathcal{B}} W_{is}^\gamma z^s + b_i^\gamma\right) = W_{i\beta}^\gamma \\
\frac{\partial z^\beta}{\partial x_m^\beta} &= \frac{\partial}{\partial x_m^\beta}\left(\sum_{s=1}^{n^\beta} x_s^\beta\right) = 1 \\
\frac{\partial x_m^\beta}{\partial \ell_k^\beta} &= \frac{\partial}{\partial \ell_k^\beta}\left(\frac{e^{\ell_m^\beta}}{1 + \sum_{s=1}^{n^\beta} e^{\ell_s^\beta}}\right) = x_m^\beta \mathbb{1}_{m=k} - x_m^\beta x_k^\beta \\
\frac{\ell_k^\beta}{\partial z^\alpha} &= \frac{\partial}{\partial z^\alpha}\left(\sum_{s \in \mathcal{A}} W_{ks}^\beta z^s + b^\beta\right) = W_{k\alpha}^\beta
\end{aligned}$$

Hence, the partial derivative in question is equal to

$$\begin{aligned}\frac{\partial \ell_i^\gamma}{\partial z^\alpha} &= \sum_{\beta\in\mathcal{B}}\sum_{m=1}^{n^\beta}\sum_{k=1}^{n^\beta}\frac{\partial \ell_i^\gamma}{\partial z^\beta}\frac{\partial z^\beta}{\partial x_m^\beta}\frac{\partial x_m^\beta}{\partial \ell_k^\beta}\frac{\partial \ell_k^\beta}{\partial z^\alpha}\\ &= \sum_{\beta\in\mathcal{B}}\sum_{m=1}^{n^\beta}\sum_{k=1}^{n^\beta} W_{i\beta}^\gamma\left(x_m^\beta \mathbb{1}_{m=k} - x_m^\beta x_k^\beta\right) W_{k\alpha}^\beta\end{aligned}$$

The total number of cycles associated with dendrite i would be:

$$\sum_{\alpha\in\mathcal{A}}\frac{\partial \ell_i^\gamma}{\partial z^\alpha}$$

which can also be viewed as a perturbation that can be added to the original linear readout ℓ^γ. The weighted sum of the perturbation and ℓ^γ is then fed into the softmax function to produce a perturbed activation of γ.

With a chosen parameter $\varepsilon > 0$ (as we are encouraging cycles), when we introduce perturbation from cycles, the output of γ would be

$$\tilde{x}_i^\gamma = \frac{e^{\tilde{\ell}_i^\gamma}}{1+\sum_{j=1}^{c} e^{\tilde{\ell}_j^\gamma}}, \quad i = 1, 2, \cdots, c$$

and

$$\tilde{x}_0^\gamma = \frac{1}{1+\sum_{j=1}^{c} e^{\tilde{\ell}_j^\gamma}}$$

where

$$\tilde{\ell}^\gamma = \ell^\gamma + \varepsilon\begin{bmatrix}\sum_{\alpha\in\mathcal{A}}\frac{\partial \ell_1^\gamma}{\partial z^\alpha}\\ \vdots\\ \sum_{\alpha\in\mathcal{A}}\frac{\partial \ell_c^\gamma}{\partial z^\alpha}\end{bmatrix} \tag{3.3}$$

The classification, then, will be

$$\hat{C} = \underset{i\in\{1,2,\cdots,c\}}{\arg\max}\ \tilde{x}_i^\gamma$$

3.4 Propagation of Distribution

Recall that in §3.1, for a neuron α we defined X^α as a multinomial distribution that can take on values in the set $\{0, 1, \dots, n^\alpha\}$, with probabilities ε_i^α obtained as a by-product of the forward propagation.

Hence, we technically have a distribution for each layer, characterized by concatenating the individual multinomial random variables representing the state of each unit.

However, given an image $z^{\mathcal{A}}$ in the input layer, we cannot sample states of the hidden units from each hidden layer $\mathcal{B}$ and integrate in order to obtain the distribution in the decision layer. This is similar to the wave function collapse caused by an observation.

Chapter 4

Experiments

4.1 Implementation

We preprocess the MNIST dataset, build the Network, and optimize the parameters of the Network using PyTorch, an optimized tensor library for deep learning using CPUs and GPUs. GPUs have shown to be very efficient in parallel computing, e.g., the 2D convolution operation.

4.1.1 Preprocessing

The MNIST dataset (Figure 3.1) contains 60,000 grayscale handwritten single-digit images labeled with numbers from 0 to 9 for training and another 10,000 for testing. For example, when we are training the network to perform binary classification, we choose two classes, extract the corresponding images and labels, and normalize the values of the pixels so that we can use them as the pixel layer $z^{\mathcal{A}}$ in §3.2.1 and §3.2.3.

4.1.2 Building the Network

We implemented the network using PyTorch, with the following classes for reusability:

- Unit: we implemented the class of hidden neurons, which contains the parameters described in §3.2.5

- Layer: a wrapping class consisting of instances of Units.

- Network: a wrapping class consisting of instances of Layers.

4.1.3 Optimization

For each image $z^{\mathcal{A}}$ and its corresponding class C, the output of the network (defined in §3.2.4) is a distribution x^γ over all the possible classes. We define the following loss function

$$\mathcal{L}(\theta, z^{\mathcal{A}}, C) := -\log x^\gamma_C$$

associated to the parameters θ (also called the "cross entropy loss") of our network. Our optimization objective is to find the value θ^* that minimizes this loss:

$$\theta^* = \underset{\theta}{\arg\min}\ \mathbb{E}_{(z^{\mathcal{A}}, C)}\left[\mathcal{L}(\theta, z^{\mathcal{A}}, C)\right]$$

where the expectation is taken over the distribution of all such (image, class) pairs.

We approach this optimization problem with mini batch gradient descent. We initialized our model with parameters $\theta^{(0)}$ at random. After preprocessing, we divide the resulting data into mini batches with a pre-fixed size N for each batch. For the i-th mini batch, we calculate the mean loss over the examples:

$$\bar{\mathcal{L}}^{(i)} = \frac{1}{N}\sum_{j=1}^{N} \mathcal{L}(\theta, z^{\mathcal{A}}_j, C_j)$$

and update the parameters as follows:

$$\theta^{(i)} \leftarrow \theta^{(i-1)} - \eta \nabla_\theta \bar{\mathcal{L}}^{(i)}$$

where $\eta > 0$ is a small positive number called the learning rate. The calculation of $\nabla_\theta \bar{\mathcal{L}}^{(i)}$ is done by automatic differentiation implemented by the PyTorch library.

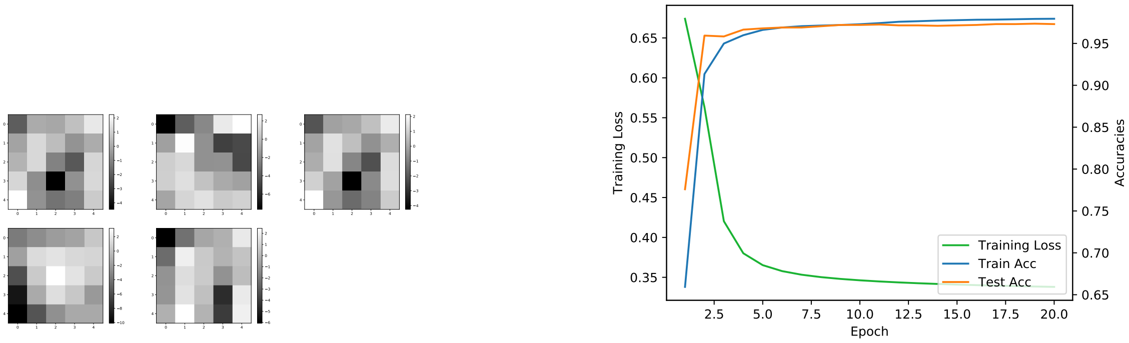
Note that using automatic differentiation by specifying ℓ^γ, the linear readouts of γ, we can also find the partial derivatives $\frac{\partial \ell_i}{\partial z^\alpha}$ for each class i and pixel $\alpha \in \mathcal{A}$. This is indeed how we implemented the perturbation outlined in §3.3.

4.2 Results

4.2.1 Binary Classification

During binary classification, we pick two different digits, e.g., 6 and 7, and extract the corresponding images and correct classes (labels) from the MNIST dataset. Then we initialize and train our network as detailed in §4.1.3. Learning rate η is set as 5×10^{-3} and we train for 20 epochs (the number of times we iterate over the entire dataset).

We started with 5 hidden neurons $\beta \in \mathcal{B}$, with a kernel size of 5. Then we changed the number of kernels and the number of hidden neurons to see their effects.



(a) Kernels of hidden units

(b) Plot of loss and accuracy

Figure 4.1 A network of 5 hidden units with kernel size 5

The loss keeps going down over the epochs and the training/testing accuracies exceed 95%. Then, we change the number of hidden units and the size of kernels, and results are as follows:

# of β's	Kernel Size	Training Accuracy	Testing Accuracy	# of Parameters
2	5×5	0.9683	0.9649	58
5	5×5	0.9794	0.9736	142
5	10×10	**1.0000**	**0.9985**	517

Table 4.1: Training and testing accuracies are the maximum accuracy over 20 epochs

The network performs best when we use more hidden units with larger kernels. For the experiments conducted, we observe a perfect training accuracy of 1 for 5 hidden units with 10×10 kernels, which uses 517 parameters. However, using only 58 parameters can also achieve accuracies over 95% both during training and testing.

To investigate what a kernel $\mathcal{K}$ is actually "looking at", we use reverse correlation method defined as below:

$$\text{reverse\_correlation}(\text{class} = c) = \mathbb{E}_{\mathcal{I}\sim c}\left[\sum_{\lambda}\langle \mathcal{K}, \mathcal{I}_{\lambda}\rangle \cdot \mathcal{I}_{\lambda}\right]$$

where the expectation is taken over all the images $\mathcal{I}$ of class c in the dataset, location λ is summed over all the possible locations on the image, and $\mathcal{I}_{\lambda}$ corresponds to an image patch whose top-left corner is λ (similar to that in §2.3).

The results for networks with 2 hidden neurons with 5×5 kernels and 5 hidden neurons with 5×5 and 10×10 kernels are shown in Figure 4.2, Figure 4.3, and Figure 4.4.

Weights

It is important to note that, by printing out the weights W^{γ} for each model that we trained, their entries are either strictly positive or negative and at about the same scale. For the two entries $W^{\gamma}_{1,\beta}$ and $W^{\gamma}_{2,\beta}$ corresponding to any hidden neuron β, they have different signs, i.e., $W^{\gamma}_{1,\beta}W^{\gamma}_{2,\beta} < 0$. This is why we claim that some neurons favor 6 and some other favor 7 in the caption of Figure 4.2.

Features

In the results from reverse correlation for kernel size 5×5, features like horizontal lines appear (Figure 4.2(b), the second kernel, and Figure 4.3(b), the third kernel). 10×10 kernels are large compared with a 28×28 image, and the kernels aim to capture features from a larger range. Hence, discussing what these kernels are "looking at" by reverse correlation becomes complicated.

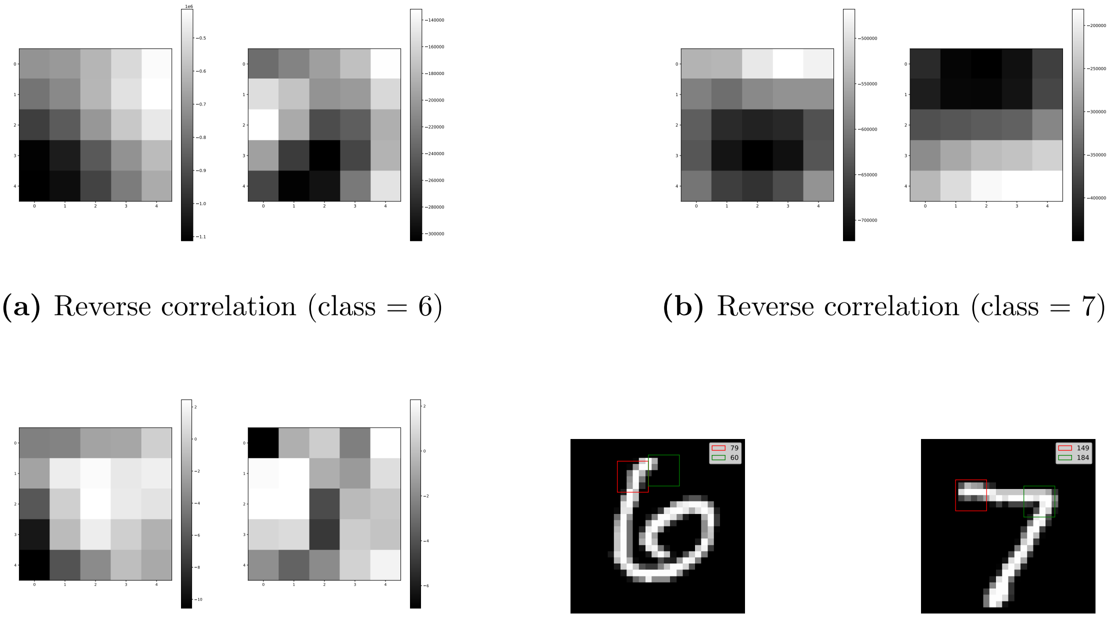
However, for the kernels themselves (as plotted in panel (c) in all three figures), no clear feature can be identified by eyes unless we have a large kernel with size 10×10. In Figure 4.4(c), we see that the third kernel (which favors digit 7) looks like a horizontal line and the fourth and the fifth kernels seem to capture the traits of part of a circle.

Locations of Maximum Activation

In Figure 4.2(e), we observe that the two trained kernels are most activated by the regions in 7, where the red kernel is associated with negative weights and the green kernel is associated with positive weights when classifying a digit 7.

In Figure 4.3(d), we observe that all the green kernels (who favor 7) are quite activated at the inner part of the "loop" of a digit 6. Since these kernels are associated with negative weights during classification in the γ neuron, those parts of a digit 6 discourage the network from classifying the digit as a 7.

In Figure 4.4(d) and (e), the arrangements among those kernels become messy. Especially in Figure 4.4(e), where the presence of a horizontal stoke is what every kernel is relying on. Another thing to note is that kernels with size 10 × 10 may be a little too large for the 28 × 28 images. Moreover, it is weird for the second kernel to detect the (almost) vertical bar of a digit 6 but still activated by the horizontal line of a digit 7.



(a) Reverse correlation (class = 6)

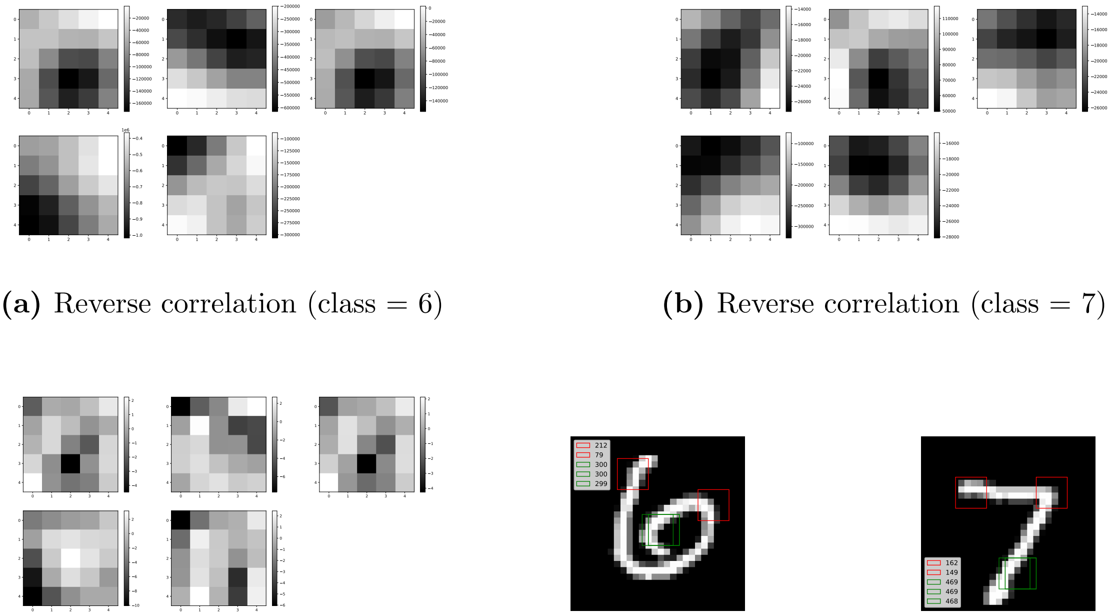
(b) Reverse correlation (class = 7)

(c) Kernels

(d)

(e)

Figure 4.2 Results for 2 hidden units with 5 × 5 kernels. (a)(b) feature the reverse correlation of the two neurons for 6 and 7, respectively. (c) are the kernels. (d)(e) feature the locations of the kernels on 6 and 7 where each kernel has the maximum activation. Red kernels favor 6 and green kernels favor 7. For the hidden neurons, neuron 1 is in red and neuron 2 is in green. A neuron $\beta \in \mathcal{B}$ favors 6 if $W^{\gamma}_{1,\beta} > 0$, and the same criterion applies to 7. The indices in the legend indicate the coordinate of the kernels. The MNIST images are of size 28 × 28. For instance, an index 60 using a 5 × 5 kernel means the pixel at the intersection of the second row and the twelfth column because for each row, there are only $28 - 5 + 1 = 24$ pixels on which we can place a 5 × 5 kernel without exceeding the boundary.



(a) Reverse correlation (class = 6)

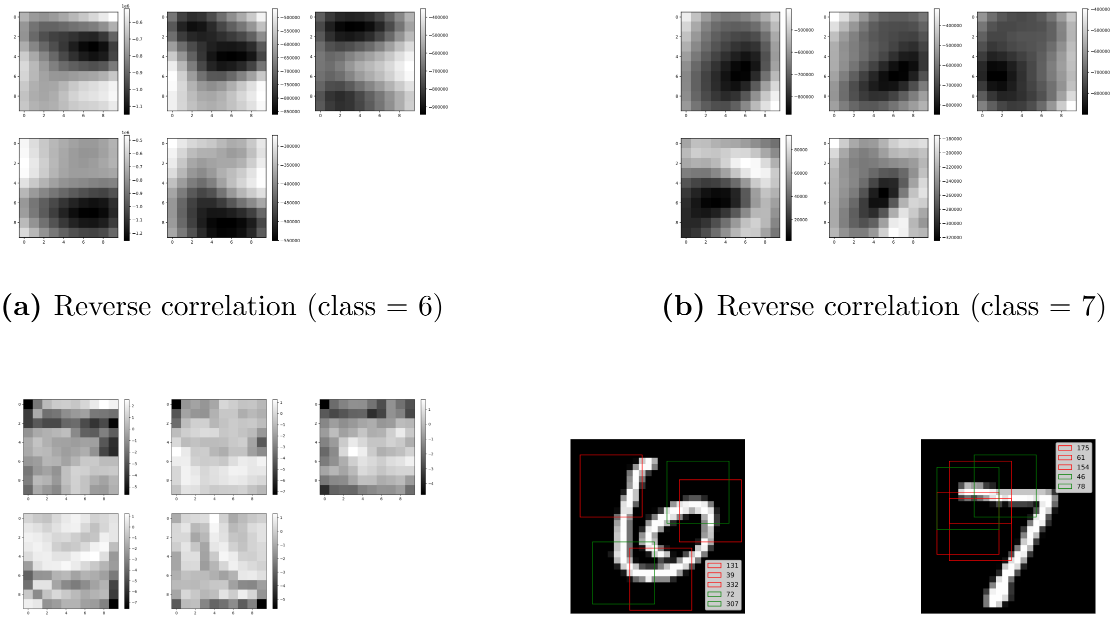
(b) Reverse correlation (class = 7)

(c) Kernels

(d)

(e)

Figure 4.3 Results for 5 hidden units with 5×5 kernels. The descriptions are the same as those for Figure 4.2 For the hidden neurons, neurons 2 and 4 (row-first counting) favor 6 and are in red; the rest of the neurons favor 7 and are in green.



(a) Reverse correlation (class = 6)

(b) Reverse correlation (class = 7)

(c) Kernels

(d)

(e)

Figure 4.4 Results for 5 hidden units with 10×10 kernels. The descriptions are the same as those for Figure 4.2 For the hidden neurons, neurons 1, 2, and 4 (row-first counting) favor 6 and are in red; the rest of the neurons favor 7 and are in green.

4.2.2 Introducing Perturbation

Perturbation, outlined in §4.3 by combining the number of *cycles* in the network, is introduced as a means to encourage functional common input (§2.3) of the network from the inputs. Here we proceed with the same setup, where we have 5 hidden neurons, each equipped with a kernel of size 5 × 5.

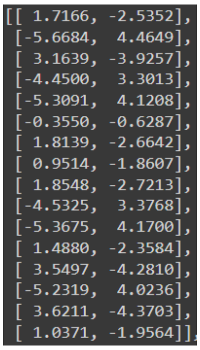
When we add the perturbation, we use the formulation in Equation 3.3 and adjust the value of ε. This value is determined before we train the model. As ε increases, the decision by γ relies more and more on the sufficient statistics S, and the model without perturbation can be viewed as a special case where $\varepsilon = 0$.

ε	Max Train Acc	Mean Train Acc	Max Test Acc	Mean Test Acc
0	0.9794	0.9540	0.9736	0.9602
0.0001	0.9798	0.9537	0.9817	0.9662
0.001	**0.9810**	**0.9575**	0.9802	**0.9719**
0.01	0.9782	0.9495	**0.9822**	0.9597
0.1	0.9664	0.8669	0.9812	0.8905
1.0	0.9652	0.8419	0.9680	0.8640
1.5	0.9738	0.8871	0.9807	0.9114
2.0	0.9762	0.8648	**0.9822**	0.8899

Table 4.2: The maximum and mean training and testing accuracies are the maximum accuracy over 20 epochs. All the models have 5 hidden units with 5 × 5 kernels.

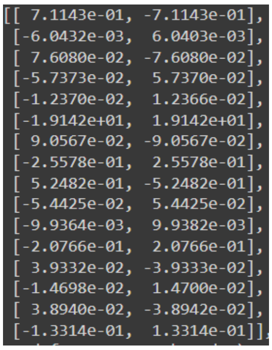
A best ε ?

Among those models, $\varepsilon = 0.001$ performs the best. However, $\varepsilon = 0.001$ is too small to make any difference by the following observation. By taking the model with $\varepsilon = 0.001$ and a batch of 16 images and passing them into the model without updating any parameters, we can obtain the linear readouts ℓ^γ and sufficient statistics S as below:



```
[[ 1.7166, -2.5352],
 [-5.6684,  4.4649],
 [ 3.1639, -3.9257],
 [-4.4500,  3.3013],
 [-5.3091,  4.1208],
 [-0.3550, -0.6287],
 [ 1.8139, -2.6642],
 [ 0.9514, -1.8607],
 [ 1.8548, -2.7213],
 [-4.5325,  3.3768],
 [-5.3675,  4.1700],
 [ 1.4880, -2.3584],
 [ 3.5497, -4.2810],
 [-5.2319,  4.0236],
 [ 3.6211, -4.3703],
 [ 1.0371, -1.9564]]
```

(a)



```
[[ 7.1143e-01, -7.1143e-01],
 [-6.0432e-03,  6.0403e-03],
 [ 7.6080e-02, -7.6080e-02],
 [-5.7373e-02,  5.7370e-02],
 [-1.2370e-02,  1.2366e-02],
 [-1.9142e+01,  1.9142e+01],
 [ 9.0567e-02, -9.0567e-02],
 [-2.5578e-01,  2.5578e-01],
 [ 5.2482e-01, -5.2482e-01],
 [-5.4425e-02,  5.4425e-02],
 [-9.9364e-03,  9.9382e-03],
 [-2.0766e-01,  2.0766e-01],
 [ 3.9332e-02, -3.9333e-02],
 [-1.4698e-02,  1.4700e-02],
 [ 3.8940e-02, -3.8942e-02],
 [-1.3314e-01,  1.3314e-01]]
```

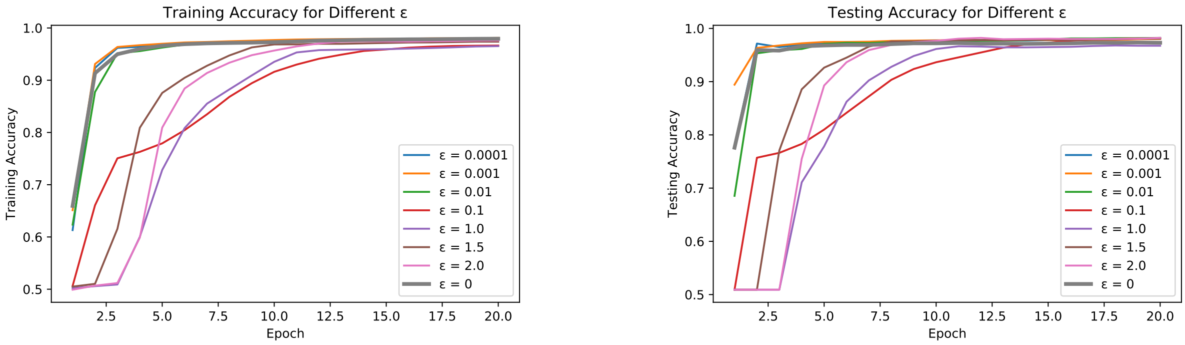
(b)

Figure 4.5 Values of ℓ^γ (panel (a)) and S (panel (b)) for a random batch of 16.

Observe that the values of S are smaller than ℓ^γ by a scale of around $\frac{1}{100}$. When S is multiplied by $\varepsilon = 0.001$ and added to ℓ^γ before passing into the softmax function to output the final probabilities over classes, then the role S plays in this part would be about 10^{-5} of the contribution by ℓ^γ. This is not much different from $\varepsilon = 0$. As for the maximum testing accuracy, it might be the case that the model converges more slowly because of a large ε.

Training Trajectory over Epochs

We are also interested in how the trends of training and testing accuracies are changed when we vary ε. By Figure 4.6, we observe that when ε becomes greater than 0.01, both the training and testing accuracies will converge more slowly than the baseline ($\varepsilon = 0$).



(a) **(b)**

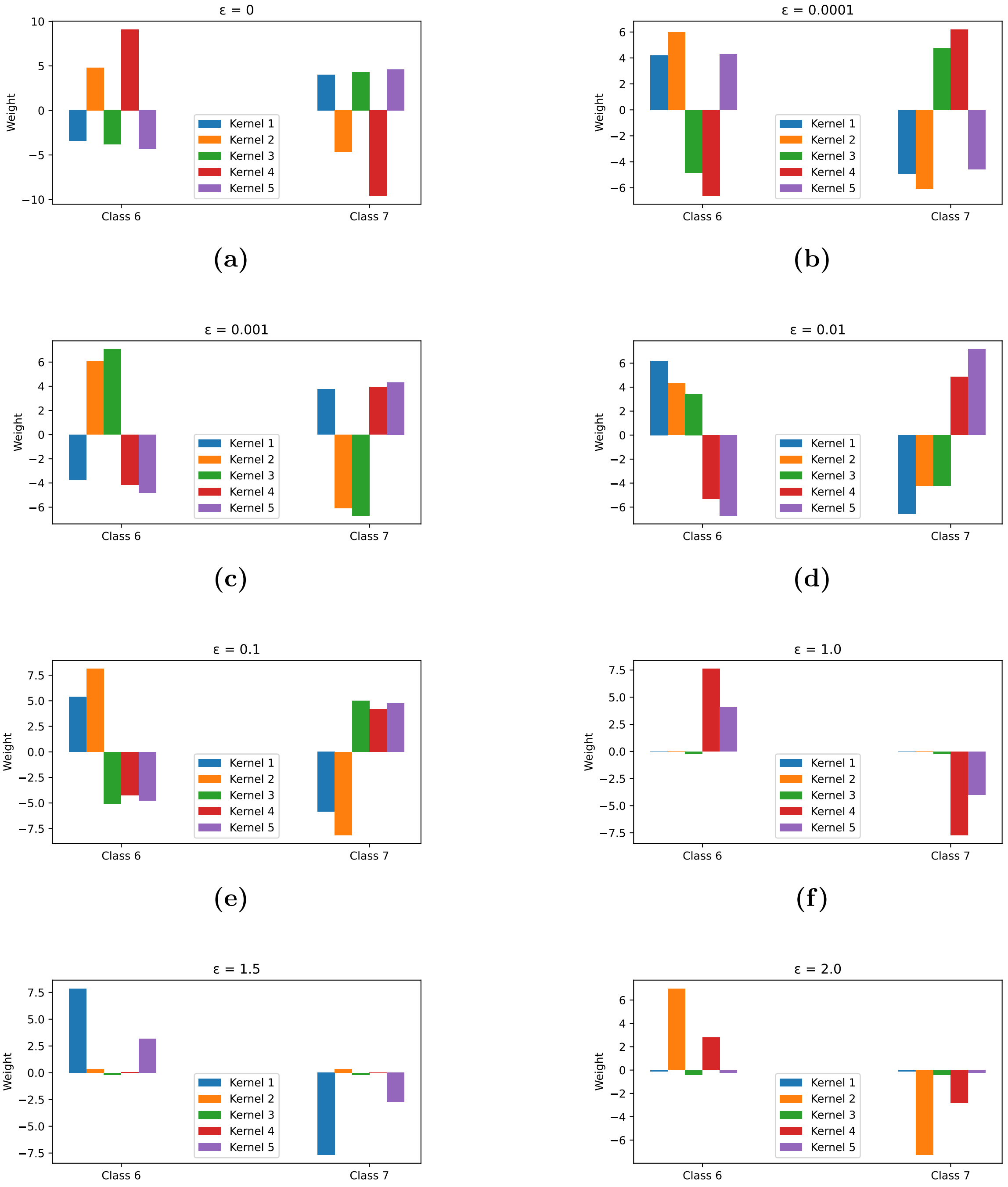
Figure 4.6 Training and Testing Accuracies v.s. Epochs

Units Favoring Either Classes?

For different values of ε's, we plotted their weights from the weight matrix W^γ corresponding to class 6 and 7, in increasing order from panel (a) to (g) in Figure 4.7. If ε is small, then we can observe that some kernels favor class 6 and some others favor class 7. However, when $\varepsilon \geq 1$, some kernels favor class 6 while others are indistinguishable because their corresponding weights in the decision layer are almost zero. It seems that for larger ε's, the models shift to a paradigm where they only learn one of the two classes.

4.2.3 Computing Resources

Parts of the experiments are conducted using the computational resources and services provided by the Center for Computation and Visualization, Brown University.



(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

Figure 4.7 Weights from W^{γ} corresponding to class 6 and 7 for different ε's.

Chapter 5

Discussion and Future Work

Based on our results in §4.2.1, multinomial machines have shown great performance in binary classification tasks on MNIST, despite with a small set of parameters, compared with current deep learning models. With a suitable set of parameters to be optimized for, they learn to extract features automatically. Alternatively, we can view them as models that approximate a function mapping from an image to a conditional distribution over all the classes in the dataset, and the "training" process amounts to optimizing the conditional distribution towards a one-hot vector, where the ground truth label bears a probability of 1 and the rest are all zeros.

However, as we introduce topological structures like cycles to the linear readout in the decision layer, we observe that the performance increase slows down. One possible reason is that when the balancing constant ε becomes larger, the model relies on the sufficient statistics S too much. Although the calculation of this term is a top-down process, i.e., from the output back to the input pixels, we are not optimizing for the configuration of the locations where the kernels receive the highest activation explicitly, thus not forcing the kernels to overlap and creating cycles (functional common inputs). We also notice that for larger ε's, the weights corresponding to some of the kernels will be roughly zero - only some of the kernels are contributing to the output of the model, and these kernels are having positive weights for the same class and negative weights for the other. This means that the model tends towards a 6-detector or 7-detector rather than one that learns *both* the features of 6's and 7's, respectively.

The first step of future work is to build a framework that is top-down. Once we have trained our bottom-up network as outlined in §4.1, we fix the kernels, weight matrices, and biases and seek to optimize the locations where each kernel has the maximum activation. The values determined by the fixed parameters provide us with initial values for those maximizing locations.

When we train the bottom-up network, the model will try to maximize the likelihood of the input pixels by exploring more regions of the image, so they tend to scatter around. When we optimize for the configuration that encourages cycles, the kernels will then come together and overlap, thus creating functional common input, or cycles by our definition. Hence, with the interplay of the two forces, we seek for a sweet spot where the kernels are neither too far from each other nor piled up on top of each other, and we propose that this configuration of kernels can both perform well and encourage topological structures to emerge.

For each neuron β in the hidden layer, it has $(28 - 5 + 1)^2 = 576$ dendrites, which is the number of locations on the image to place its 5×5 kernel. Since we are optimizing the locations for each β, the number of parameters will be around three thousand even if we only use 5 hidden units, which is way more than the 142 parameters calculated in Table 4.1. Interestingly, this is aligned to the presence of numerous feedback connections in the visual cortex, and these connections are thought to learn through experience to shape our visual capabilities. Recent work has shown the potential advantage of feedback connections when representing both in time and space [18].

References

[1] Jose-Manuel Alonso, W Martin Usrey, and R Clay Reid. “Precisely correlated firing in cells of the lateral geniculate nucleus”. In: *Nature* 383.6603 (1996), pp. 815–819.

[2] Rony Azouz and Charles M Gray. “Adaptive coincidence detection and dynamic gain control in visual cortical neurons in vivo”. In: *Neuron* 37.3 (2003), pp. 513–523.

[3] Lo-Bin Chang et al. “Maximum likelihood features for generative image models”. In: *The Annals of Applied Statistics* 11.3 (2017), pp. 1275–1308. DOI: 10.1214/17-AOAS1025. URL: https://doi.org/10.1214/17-AOAS1025.

[4] William M Connelly and Greg J Stuart. “Local versus global dendritic integration”. In: *Neuron* 103.2 (2019), pp. 173–174.

[5] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[6] Stuart Geman. “Invariance and selectivity in the ventral visual pathway”. In: *Journal of Physiology-Paris* 100.4 (2006), pp. 212–224.

[7] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[8] David H Hubel and Torsten N Wiesel. “Receptive fields of single neurones in the cat’s striate cortex”. In: *The Journal of physiology* 148.3 (1959), p. 574.

[9] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962), p. 106.

[10] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007. Chap. 8.2.

[11] Eugene M Izhikevich. "Simple model of spiking neurons". In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.

[12] Youngeun Kim et al. "Neural architecture search for spiking neural networks". In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*. Springer. 2022, pp. 36–56.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[16] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: `http://yann.lecun.com/exdb/mnist/`.

[17] Jeffrey C Magee. "Dendritic integration of excitatory synaptic input". In: *Nature Reviews Neuroscience* 1.3 (2000), pp. 181–190.

[18] Tiago Marques et al. "The functional organization of cortical feedback inputs to primary visual cortex". In: *Nature neuroscience* 21.5 (2018), pp. 757–764.

[19] David Marr. *Vision: A computational investigation into the human representation and processing of visual information*. MIT press, 2010.

[20] Carver Mead. "Neuromorphic electronic systems". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636.

[21] Stephen E Palmer. *Vision science: Photons to phenomenology*. MIT Press, 1999.

[22] Seymour Papert. *The summer vision project. Memo AIM-100*. 1966.

[23] Pamela Reinagel and R Clay Reid. "Precise firing events are conserved across neurons". In: *Journal of Neuroscience* 22.16 (2002), pp. 6837–6841.

[24] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[25] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[26] Kashu Yamazaki et al. “Spiking neural networks and their applications: A Review”. In: *Brain Sciences* 12.7 (2022), p. 863.

[27] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 818–833.