Feature Selectors that Fire Together Wire Together: A Hebbian Based Approach to Backprop-Less Learning in Artificial Neural Networks

Zeyuan Zhu
McGill University
Zeyuan.zhu@mail.mcgill.ca

Shiyan Liu
McGill Univeristy
shiyan.liu2@mail.mcgill.ca

Abstract

Multi-layer perceptrons (MLPs) have long been accepted as a staple of supervised machine learning. However, they suffer from the need for large amounts of labelled data. Simultaneously, it has been known that the most common learning rule in training MLPs, backpropogation (BP) using gradient descent (GD), is not biologically plausible. Hebbian plasticity rules have been regarded as a potential solution to the GD problem due to their localized nature. This paper implements Hebbian learning in a one-hidden-layer winner-takes-all (WTA) MLP, inspired by lateral inhibition (LI) concepts in the human brain. These methods aim to allow GD-less learning in MLPs by imitating the learning process in humans. We show that features learned through Hebbian plasticity are both interpretable and viable, and examine the nature of these features for different LI strengths.

1 Introduction

As deep Artificial Neuron Networks (ANNs) are becoming more popular, we are facing more problems regarding computational resources. ANNs are efficient at generating their outputs through the use of GD, but in doing so they require large amounts of labelled data. The use of GD also requires a back-propogation step to training, which only gets more intensive as ANNs become deeper. Combining this with the increase in size of popular datasets such as ImageNet, ($\sim$ 1.3 million training examples), the need for efficient unsupervised learning algorithms is becoming apparent. One possible avenue for alleviating this issue is through biology. It has been known for that the method of GD is not biologically plausible, as layers of neurons early in the network depend on the information of layers above it in order to learn, which is at odds with widely accepted models of how learning occurs in the brains of humans, which emphasize the locality of learning in neurons. A more important difference however, is that biological learning is almost fully unsupervised, so an interesting question is whether we can emulate the process of unsupervised biological learning in order to improve the efficiency of learning with ANNs.

1.1 Hebbian Learning

Hebbian Learning is a neuro-psychological theory regarding the process of learning in synapses. It is commonly summed up by the adage "neurons that fire together wire together", in that a synapse between a pre- and post-synaptic cell will become more efficient through repeated stimulation through the cells. In the context of ANNs, it is expressed through a local learning rule.

$$\Delta W = \alpha y x \tag{1}$$

Here, W is any weight corresponding to a given neuron in our network, α is a learning rate constant, x is the input to the neuron, i.e. the presynaptic signal, and y is the output, i.e. postsynaptic signal, of the neuron. This learning rule voids the need of calculating the gradient of a global loss function, and instead only modifies a neuron using information that is locally available to it. However, this learning rule runs into the issue of boundedness, as there's nothing preventing the weights from diverging given enough inputs, and in practice this issue prevents this rule from becoming stabilizing during training. The ability of a neuron's efficiency to decrease is noted as one of the six essential properties of Hebbian learning in regards to ANNs (1), which necessitates a change to the learning rule in order to accommodate this requirement.

1.2 Oja Rule

The Oja rule is an extension of the vanilla Hebbian learning rule that aims to increase long term stability by adding a second term that controls the growth of the weights during training.

$$\Delta W = \alpha(xy - y^2 W) \tag{2}$$

The first part of the update is identical to the vanilla rule, but the second term is what makes this more viable for training. In addition to stabilizing the weights, it has a nice interpretation as the ability of the network to 'forget' the prior value of the weight. However, the rule is limited in that it cannot be applied to multiple outputs. Again, we need to extend our learning rule.

1.3 General Hebbian Algorithm

The General Hebbian Algorithm (GHA), also known as Sanger's Rule, is a further extension on the Hebbian learning rule that allows for multiple output neurons to learn differently. Let i, j be indices denoting the output (y) and input dimensions (x) respectively, we write Sanger's Rule as:

$$\Delta W_{ij} = \alpha((y \otimes x)_{ij} - y_i \sum_{k=1}^{i} W_{kj} y_k) \tag{3}$$

The tensor product $(y \otimes x)_{ij}$ can be rewritten as $y_i x_j$ for simplicity. This rule enforces orthogonality of feature selectors across the hidden layer.

1.4 Lateral Inhibition

Lateral inhibition refers to the ability of an excited neuron to suppress the activity of neighbouring lateral neurons. Specifically, it deals with the neuron actively disabling action potentials from propagating in its neighbours, thereby inhibiting their synaptic capacity to fire. In ANNs, this can be expressed through a lateral rule as follows. Let h_μ be a neuron from a given hidden layer in our ANN, and let $\tilde{h}, h$ denote the pre and post-activation values respectively.

$$h_\mu = \frac{\tilde{h_\mu^\lambda}}{\max_\nu \tilde{h_\nu^\lambda}} \tag{4}$$

λ is a hyper parameter that determines the strength of the inhibition. In the limit of $\lambda \to \infty$, only the maximally activated neuron in the layer is stays activated and all the other neurons have zero activation. Inversely, when $\lambda = 1$ there is no inhibition and the relative activation of neurons is not affected by the activation of the other neurons. Changing the value of λ allows us to regulate the strength of inhibition across the layer.

The study of the effectiveness of these unsupervised biological methods have mainly been focused on binary inputs and activation functions. In this paper we will apply these ideas to more complex architectures. This structure of this paper will follow as such. First, we will discuss related work in applying biological learning rules to ANNs. We will implement our own model using BP-less Hebbian learning. Our models will be evaluated on standard datasets, and performance will be compared against previous papers.

2 Related Work

2.1 Hebbian+SGD MLP

Krotov and Hopfield implement the ideas of Hebbian learning and lateral inhibition in (2). Their architecture consists of a 1-hidden layer MLP, where the weights $W_{\mu i}$ from the input layer v_i to the hidden layer h_μ are determined in a completely unsupervised way through a local Hebbian plasticity rule determined by differential equations for the weights $W_{\mu i}$, and the hidden layer neurons h_μ. After determining the $W_{\mu i}$, they freeze the first layer and train the second layer weights through a standard SGD method. They apply this architecture to the MNIST and CIFAR-10 datasets. On MNIST, their biological algorithm matches the established state-of-the-art (SOTA) performance of around 1.6% using MLPs. Their performance on the CIFAR-10 dataset was weaker, reaching a minimum of around 45% after minimizing across hyperparameters.

2.2 Associative Memory

In (3), Krotov and Hopfield investigate another neurobiological concept in associative memory, and its possible implementation in deep learning architectures. The environment that they propose is a system of, not necessarily layered, N binary neurons, denoted by a vector as σ, which are able to store K memories, denoted as ζ^μ, for $\mu = 1...K$. The goal of the system is when presented with an input x, representing some pattern, the system shall find the ζ^μ that best matches the pattern x. This is accomplished through the use of an energy function given as:

$$E = -\sum_{\mu=1}^{K} F(\zeta_i^\mu \sigma_i) \tag{5}$$

Here, $F(x)$ is a smooth function of the input variable, and summation over i is implied. The particular $F(x)$ used is a rectified polynomial dependent on a hyperparameter n.

$$F_n(x) = \begin{cases} 0 & x \leq 0 \\ x^n & x > 0 \end{cases} \tag{6}$$

The connection between rectified polynomial functions and biological processes is the following. One can imagine the rectified function as a sort of threshold potential for ANNs. For inputs $x < 0$, the potential is not reached, and no activation occurs. However, for inputs that are positive, there exists an additional hyperparameter n that controls the strength of the activation. When the system receives an input x, we can iteratively update the neurons in the system using the following simplified learning rule.

$$\sigma_i = g(E_{\sigma_i=1} - E_{\sigma_i=-1}) \tag{7}$$

$g(x)$ is some activation function, the authors use both the sigmoid and tanh functions for different purposes. This rule ensures that the update of each neuron must result in a net energy loss for the system. The authors apply this model to the MNIST dataset, using rectified polynomial functions with various power laws (values of n). Their σ vector consists of a input system, which matches each pixel of an MNIST image, as well as a classifier system, which consists of 10 neurons for each class of the MNIST dataset. Upon feeding the system an image vector v, the input system is changed to match v, and each classifier c_α is updated using the aforementioned rule to determine the class. The K memories are initially drawn from standard Gaussian, but are minimized using a loss function and back-propogation every iteration to achieve the minimum error on the training set. They achieve slightly better than MLP SOTA performance at $n = 3$, at around 1.4% test error. The resulting memories for various n are worth examining. For low n, the memories match specific features of each digit rather than the digit itself, such as certain line strokes. At high n, it is the opposite, the memories appear as prototypes of digits, with the entire digit appearing in the memory.

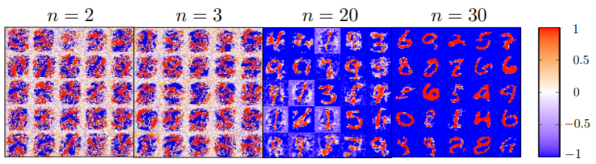


Figure 1: Randomly selected memories for various power laws after training. One can see the clear distinction between the selected memories for low and high n. It appears that the choice of power law significantly changes what information a network learns from a dataset.

This change in information learned can be explained as follows. As n grows, only the most similar memories will contribute to the update rule given in 7. One can imagine that if there were are a larger threshold potential in a biological system, only the stimuli with the most association would be able to perform an activation. The features of the digits individually do not represent a specific digit enough to activate this potential at higher powers, and as such do not contribute to the memories. However, at low n, due to a lower threshold, the similarities between the learned memories and the actual digits do not need to be as high in order for activation to occur, so the memories stored are allowed to have more feature-like properties as opposed to full representations of digits. This power law is akin to the aforementioned lateral inhibition rule, with n taking the place of λ.

2.3 SoftHebb

In (4), Moraitis et.al propose a modification of a WTA network that modifies the "absolute winner" mentality of the original WTA architecture in a similar way to LI. The cornerstone of their network is an Oja-esque plasticity rule that is spatially local in the output layer.

$$\Delta W_{ik} = \alpha y_k (x_i - u_k W_{ik}) \tag{8}$$

The indices i, k represent the input and output dimensions, with x, u, y denoting the input, pre-activation output, and post-activation output respectively.

3 Methods

3.1 MLP

The architecture of our MLP is a two-layer MLP consisting of a feature detector layer and a classifier layer. The hidden layer h is activated according to Equation [4], while the classifier layer is put through a SoftMax operation to obtain probabilities. We first investigate the ability of the MLP to properly learn features using our local learning rule. See Figure.

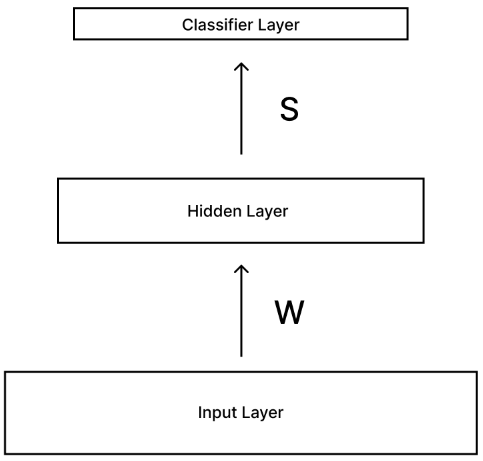


Figure 2: An overview of the general architecture of our model. Both the W_{ij} and S_{ij} weight matrices are learned purely through a feed forward process, though with slightly different learning rules, described in section **cite the section here.**

3.1.1 Exponential Averages

An early issue that was discovered was the presence of many "dead" feature selectors, which persisted as random noise after training. These feature selectors were the result of the y_i pre-factor in the weight update equation for Sanger's rule, as if these output values were too small, the selectors would fail to learn over the course of training. We aimed to fix this tracking the exponential average of the output over training, and adjusting the feature selectors accordingly.

We denote the exponential average vector as a, and update it every training iteration according to the equation:

$$a_i \rightarrow \gamma * a_i + (1 - \gamma) * y_i \tag{9}$$

Here, γ is a hyperparameter that controls the "rigidity" of the exponential average, a higher γ indicates that a is harder to change. y_i is the i-th output, after inhibition. [4]. Using this vector, we define a growth factor f_{ij} for the weight matrix W_{ij}, based on a ratio matrix A_{ij}. See Figure 3.

$$W_{ij} \rightarrow f_{ij} * W_{ij} \tag{10}$$

$$f_{ij} = \epsilon * tanh(-\frac{1}{\epsilon}(|A_{ij}| - 1)) + 1 \tag{11}$$

$$A_{ij} = \frac{a_{ij}}{avg(a)} \tag{12}$$

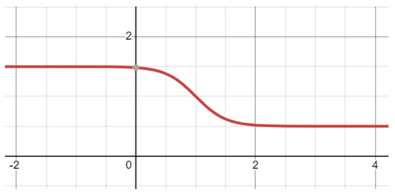


Figure 3: A snapshot of the tanh equation determining the growth factors, with $\epsilon = 0.5$. If the i-th exponential average is much smaller than the average, then it is multiplied by a value of $1 + \epsilon$. A higher ϵ value allows for a strong adjustment of weights, but care has to be taken to prevent overflow.

3.2 Classifier Update

We use a modified learning rule for the weights of our classification layer, as a way of incorporating supervision into our model. There's no requirement for any orthogonality in the classifier weights, so we discard the second term in equation [3], leaving only our vanilla Hebbian learning rule [1].

As we wish for the true label to be a stable point of our system, we use the following modified learning rule based on Equation [1], denoting by t the true label of any given training instance.

$$\Delta S_{ij} = \alpha(t - y_i)x_j \tag{13}$$

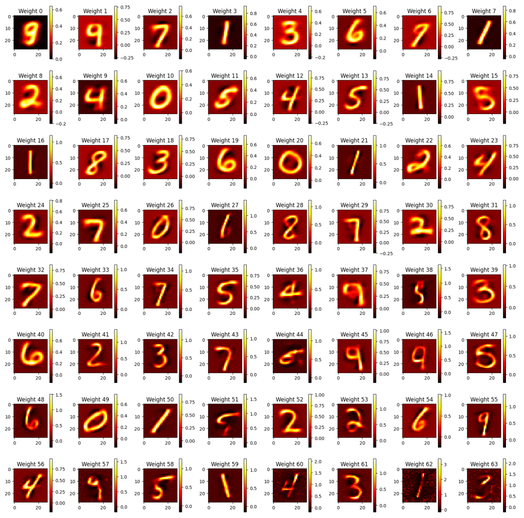
To account for overflow, we also perform a max norm on each classifier weight every iteration. With the nature of our learning rule, the first feature selector will always learn an average of every output, leading to meaningless results. As such, we zero the first element of each classifier weight, to account for this behaviour.

4 Results

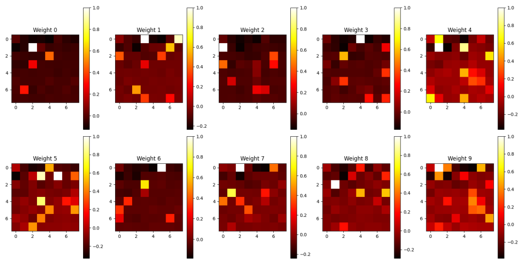
4.1 Preliminary Results

Figure [4] shows preliminary results for a layer with 64 feature selectors, using the standard MNIST dataset. With the high λ value, we see prototypical features being learned, in accordance with expected behaviour.

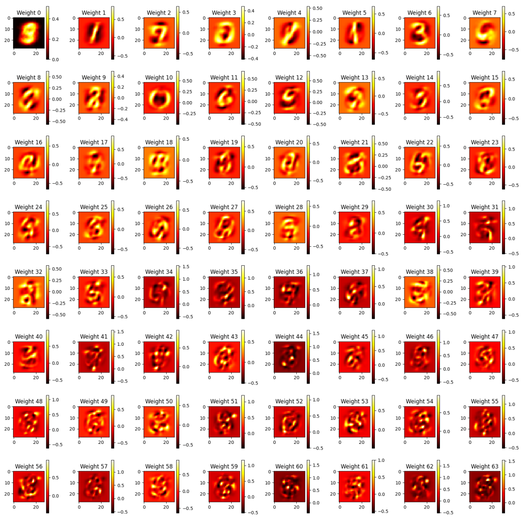
The strongest classifier weights seem to be biased towards the earlier feature selectors for both λ values. This behaviour likely stems from the non-symmetrical nature of the Sanger's learning rule employed in our network, where the selectors with smaller index will learn the most relevant principal components. In Figure 4a for example,when $\lambda = 15$ the feature selectorslearn weights close to prototypical representations of each digit, as opposed to $\lambda = 1$, where these feature selectors are learning weights close to parts of a digit which can be composed together to form a full picture [4c]. The classifier weights [4b] [4d], reflect this, as the most weighted feature selector for each digit classification lies in the first 18 selectors.



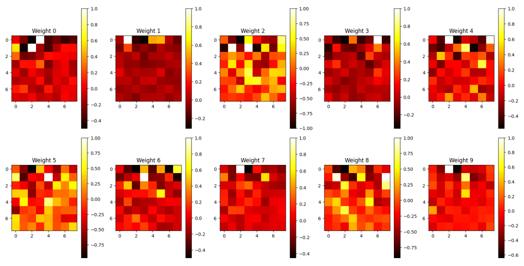
(a)



(b)



(c)



(d)

Figure 4: Preliminary results for feature selectors and classifier layers on MNIST for $\lambda = 15$ [(a), (b)] and $\lambda = 1$ [(c), (d)]. The classifier layers represent the weights of the digits 0-4 and 5-9 left to right across the two rows.

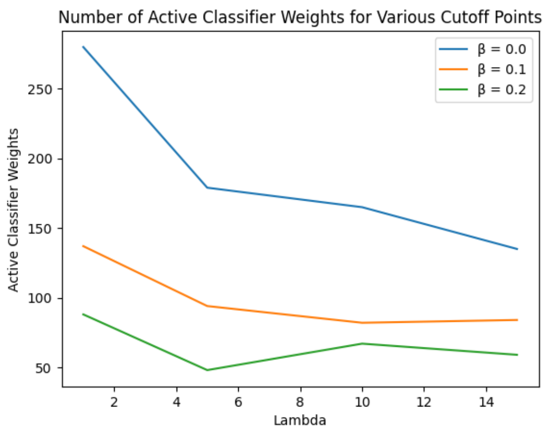


Figure 5: Active selectors for 3 different cutoff points (β).

4.2 Feature Selectors used by classification layer

With prototypical features, we expect that the classifier layers use "less" of the available feature selectors to make a decision. To quantify this, we'll define 3 cutoff points $\beta = 0.0, 0.1, 0.2$ and determine the average number of weights in the classifier layer that are above each β for a few values of λ. See figure [5].

We see a general trend of less prototypical selectors being used for a decision as opposed to their feature-like counterparts [1]. Lower λ values push most of their active feature selectors (> 0), to higher weights. With less prototypical selectors, more selectors are required to obtain a "full picture" of the input, so inevitably there will be more selectors that weighted heavily by the classifier.

λ	Low Weight Selectors (%)
1	31.4
5	26.8
10	40.6
15	43.7

Table 1: Low weight selectors for various λ values.

4.3 Viability of Feature Selectors

It should be the case that the strongest feature selectors for each digit match the general outline of the digit. Figures [6a] and [6b] show this for high and low λ values respectively. We see expected behaviour in terms of shared features for each. For low λ values, we see a fair amount of shared feature selectors, (for example the first selector of 4, 7 and 9), as opposed to the high λ case, where the only shared selector we see is the first selector of 9 and the third of 4. We can see a qualitative match between the features used and the digit learned.

4.4 Hebbian Feature Selectors used in Classification

Our observation on how Hebbian features selectors activated in classification also supports our intuition that

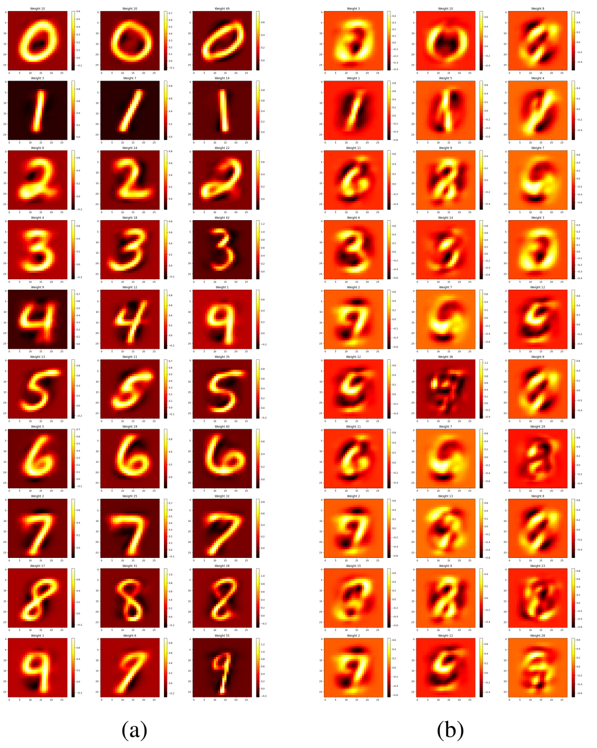


Figure 6: The 3 strongest feature selectors for each digit (by row), for a). $\lambda = 15$, and b). $\lambda = 1$.

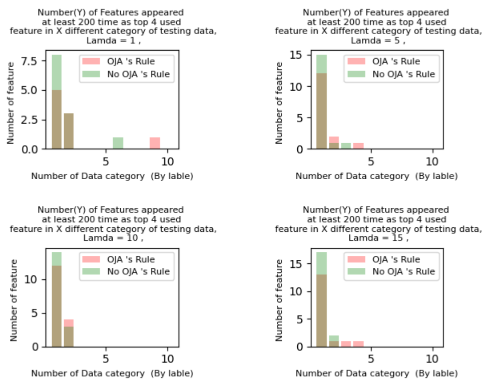


Figure 7: feather widely used by multiple category of classification for $\lambda = 1, 5, 10, 15$.

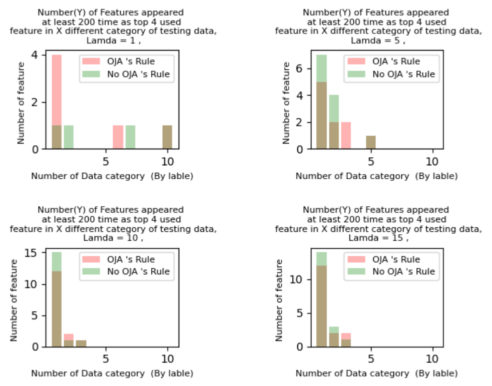


Figure 8: feather widely used by multiple category of classification for $\lambda = 1, 5, 10, 15$.

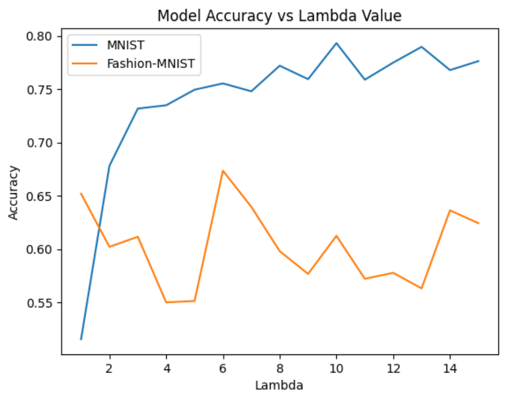


Figure 9: Accuracy values for our MLP experiment for $\lambda = 1 - 15$.

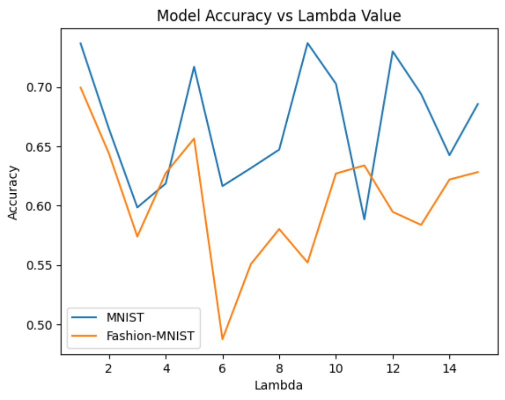


Figure 10: Accuracy values for our MLP experiment for $\lambda = 1 - 15$, using the Oja rule classifier.

model with higher lambda not only learns(showed in section 4.2) but also uses prototype and models with lower lambda uses data feature for classification.

Below is a graph showing the amount of feather widely used by multiple category of classification, for both MNIST [7] and Fashion-MNIST [8]

This graph also shows us some interesting effect about Oja's Rule. It is clear from MINST plot that when Oja's Rule is applied the feature tends to be used by more categories in general, it is also clearly that when Oja's rule is applied a larger number of features are used in scale above the threshold , we can't determine the exact reason for those observation but this may worth further exploration.

4.5 Entropy of outputs

Entropy is calculated by $Entropy = \ln\left(\frac{a}{\hat{a}}\right) \cdot \left(\frac{a}{\hat{a}}\right)$ where a is the output vector of layers and $\hat{a}$ is the sum of a.

We also investigated what output entropy of Hebbian layer and classifier layer looks like , and the result was excellent, a high lambda shrinks the entropy of Hebbian output[12][13] greatly for both dataset. In classifier layer[14][15] the situation is also clear: lambda value doesn't seem relevant to the output entropy because clamping dominating the output weight.

All of these indicates the fact a high lambda can be combined with other manner to achieve an excellent classification.

5 Accuracy

We test our models on the standard MNIST and Fashion-MNIST test sets of 10000 images. Hyperparameters and exact values for accuracies are given in the Appendix.

We receive a maximum accuracy of around 79% for a λ of 10. It appears that our design is more suited towards prototypical features.

In general, the network performs weaker on Fashion-MNIST, achieving around a 70% max accuracy, around 10% lower than on the standard MNIST dataset.

6 Conclusions

Hebbian-based methods are shown to learn viable and interoperable representations of MNIST image data through a BP-less MLP. The features selectors that are learned visually match the patterns of the input data, and the classifier weights for each labelled class are shown to use selectors that match the patterns associated with their class. Max accuracy of 79.32% and 69.95% are achieved using a one-layer Hebbian MLP.

Study can be done into constructing a deeper network, where early layers are imbued with low λ values, which increase with layer. This architecture would allow early layers to learn more feature-like representations of an input, which are combined at deeper layers to construct a full picture of the input. We can also try to find a better way than clamping to use previous layers' output and get better accuracy. How to obtain a best lambda universally or for specific data set is also a way to go.

7 Acknowledgements

We would like to thank Professor Isabeau Premont-Schwarz for his generous supervision and guidance during this project, without which we could not have completed this.

References

[1] E. Kuriscak, P. Marsalek, J. Stroffek, and P. G. Toth, “Biological context of hebb learning in artificial neural networks, a review,” *Neurocomputing*, vol. 152, pp. 27–35, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231214015239

[2] D. Krotov and J. J. Hopfield, “Unsupervised learning by competing hidden units,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, pp. 7723 – 7731, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49479422

[3] ——, “Dense associative memory for pattern recognition,” *CoRR*, vol. abs/1606.01164, 2016. [Online]. Available: http://arxiv.org/abs/1606.01164

[4] T. Moraitis, D. Toichkin, A. Journé, Y. Chua, and Q. Guo, “Softhebb: Bayesian inference in unsupervised hebbian soft winner-take-all networks,” *Neuromorphic Computing and Engineering*, vol. 2, no. 4, p. 044017, Dec. 2022. [Online]. Available: http://dx.doi.org/10.1088/2634-4386/aca710

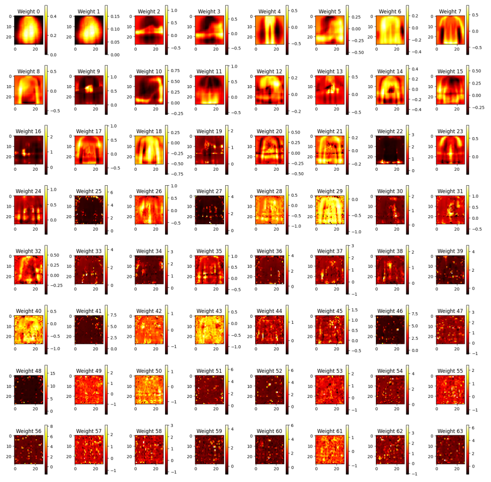
8 Appendix

Hyperparameter	Value
Hebbian Learning Rate (α)	0.01 / 0.005
Num. Epochs	3 / 3
Exponential Average Strength (ϵ)	0.01 / 0.01
Hidden Layer Size	64 /64

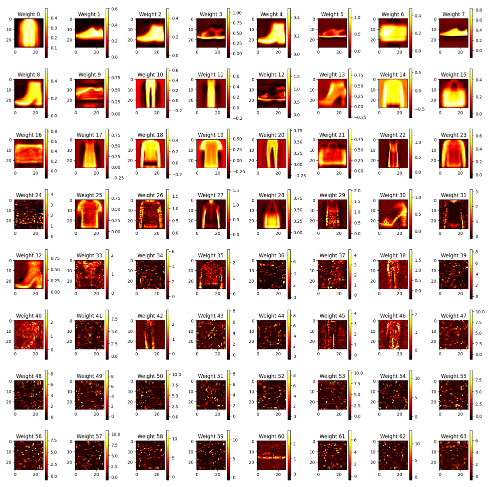
Table 2: Relevant hyperparameters and their values for our MLP experiment. MNIST / Fashion-MNIST

λ	Accuracy (Digit)	Accuracy (Fashion)
1	73.66 / 51.54 %	69.94 / 65.19 %
2	66.46 / 67.81 %	64.38 / 60.21 %
3	59.84 / 73.18 %	57.39 / 61.15 %
4	61.85 / 73.49 %	62.73 / 55.00 %
5	71.69 / 74.95 %	65.64 / 55.13 %
6	61.64 / 75.54 %	48.74 / 67.35 %
7	63.15 / 74.80 %	55.06 / 63.96 %
8	64.71 / 77.20 %	58.02 / 59.80 %
9	73.68 / 75.94 %	55.20 / 57.67 %
10	70.25 / 79.32 %	62.70 / 61.24 %
11	58.83 / 75.89 %	63.38 / 57.21 %
12	72.98 / 77.49 %	59.47 / 57.77 %
13	69.40 / 78.97 %	58.37 / 56.31 %
14	64.24 / 76.79 %	62.19 / 63.63 %
15	68.55 / 77.63 %	62.82 / 62.43 %

Table 3: Exact accuracies for various LI strengths on MNIST test sets with/without the Oja’s rule classifier.



(a)



(b)

Figure 11: Feature selectors on Fashion-MNIST for a). $\lambda = 1$, b). $\lambda = 15$. Less feature selectors are learning meaningful patterns as opposed to the standard MNIST set.

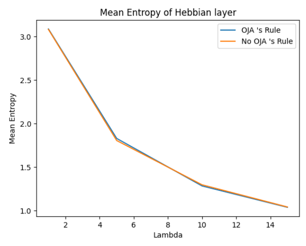


Figure 12: entropy of output from Hebbian layer for $\lambda = 1, 5, 10, 15$. MINST data

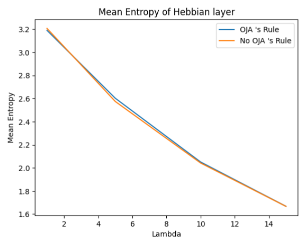


Figure 13: entropy of output from Hebbian layer for $\lambda = 1, 5, 10, 15$.Fashion MINST data

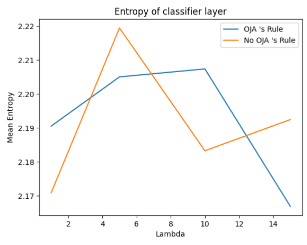


Figure 14: entropy of output from Classifier layer for $\lambda = 1, 5, 10, 15$. MINST data

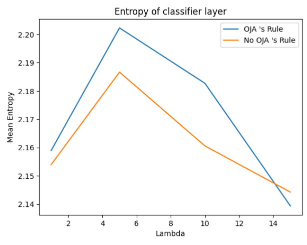


Figure 15: entropy of output from Classifier layer for $\lambda = 1, 5, 10, 15$.Fashion MINST data