Neural Learning: Can we make it a little more bio-inspired!

A Thesis Submitted in the Partial Fulfilment
of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE

By

Ambar Bandyopadhyay
[Roll No: CS1610]

Under the guidance of

Prof. Nikhil R. Pal
Electronics and Communication Sciences Unit (ECSU)

INDIAN STATISTICAL INSTITUTE, KOLKATA
July, 2018

M.TECH(CS) THESIS COMPLETION CERTIFICATE

Student: Ambar Bandyopadhyay (CS1610)
Title: Neural Learning: Can we make it a little more bio-inspired!
Supervisor: Prof. Nikhil R. Pal

This is to certify that the thesis titled "**Neural Learning: Can we make it a little more bio-inspired!**" submitted by **Ambar Bandyopadhyay** in partial fulfillment for the award of the degree of Master of Technology is a bonafide record of work out by him under my supervision. The thesis has fulfilled all the requirements as per the regulations of this Institute and, in my opinion, has reached the standard needed for submission. The results contained in this thesis have not been submitted to any other university for the award of any degree or diploma.

Date	Prof. Nikhil R. Pal

To my parents, my sister, and my well wishers...

Acknowledgements

I am deeply indebted to many people who have contributed in making the completion of this thesis work possible. I would like to express my earnest gratitude and respect to my supervisor, Prof. Nikhil R. Pal, for his guidance, constant support and monitoring throughout the course of this work. I do wish to gratefully acknowledge the continuous cooperation and support provided by Kaustuv Nag, Suvra Jyoti Choudhury, Suchismita Das, Sunanda Das and all other research fellows and project linked personnels of ECSU, Indian Statistical Institute, Kolkata. I convey my gratitude to the teachers of Indian Statistical Institute for there support.

Abstract

Though neural networks are inspired by human brains, recent studies have reported several differences between them in terms of their working principles. Specifically, some investigations on animal brains have shown that, in an animal brain, for different stimuli, different clusters neurons get activated. For example, when an animal visualizes different images, neurons from different parts of the brain gets activated. Being inspired from such an observation, here we propose a multilayered model of neural network that incorporates an idea of local activation of neurons for different group of objects (classes). In order to realize activation of distinct spatial clusters of neurons for different types of stimuli, the proposed model makes an interesting integration of a multi-layer perceptron and a self-organizing map. When compared to a conventional multilayer perceptron, the proposed model produces distinct locally activated regions for different classes and at the same time it learns to discriminate between classes.

Keywords: multi-layer perceptron, self-organizing maps, visual cortex, visual stimuli.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Introduction

Neural networks are one of the most frequently used tools in Machine Learning. One of the main characteristics of neural networks is that it is inspired from brain. There is a study done by Karpathy [1], the director of Artificial Intelligence Tesla, where he showed that almost 10% of all papers submitted to arxiv-sanity database in March 2017 mention Tensorflow and almost 31% of all papers use deep learning frameworks (including Tensorflow). There he also gives the top hot keywords that are used in the papers and some of these are ResNet, Tensorflow, GANs. He used last five years arxiv-sanity database which contains 28303 machine learning papers to do the study [1]. This clearly suggests that neural networks, particularly deep neural networks, is a very active area of research now.

Though neural networks are used in many applications there are some fundamental differences between the way a human brain and a neural network works. In neural networks, we use backpropagation to update the weights of the neurons' axons. There we find the amount it will be updated by multiplying the error signal with every synaptic weight on each neuron's axon and further we go downstream. For this, there needs to be a precise, symmetric backward connectivity pattern, but in a human brain, it is thought to be impossible [2]. Neurons in the human brain fire spikes whenever the cell potential crosses a threshold. When the neurons fire the

potential of the cells continuously rise to a peak and suddenly it falls to undershoot its resting value. It then slowly decays towards its resting value. But in neural networks, we simplified the idea by assuming that when a neuron fires its cell potential value directly goes to the resting value [3]. So we can say that the neural networks that we are using is much simpler version than the actual one.

Recently some interesting observations have been made by scientists by doing experiments on brains of animals including humans. In one experiment different visual stimuli were shown to rats and cats and periodically the activity of the neurons were captured. It has been seen that for different stimuli different group of neurons are activated [4]. In another experiment images of different persons and places were shown to five people and the activity of different parts of the brain were captured. Here it has been seen that for images of different persons or of different places, neurons of different parts of the brain are activated [4].

All these experiments on the brain suggest that depending on the stimuli, different groups of neurons are activated, where members of each group are spatial neighbors. Moreover these groups are distributed over different regions.

Here our objective is to design neural networks that are more strongly influenced by our brains. For this, we attempt to build a model where neurons of different parts of the network will be participating in different classes in the dataset.

1.2 Thesis Outline

The rest of the thesis is organized as follows. In Section 2, we briefly discuss the preliminaries and background related to our work. In Section 3, we describe some of the works that are related to our work. Section 4 describes the detailed construction of our scheme along with the data structures used in the construction. There we show some of the experimental results. And finally in Section 5 we conclude and we discuss some of the possible future works.

Chapter 2

Preliminaries

2.1 Multi-layer Perceptron

Here we consider a Multi-Layer Perceptron (MLP) with a single hidden layer. Figure 2.1 shows the architecture. Let an input pattern be $\mathbf{x} = (x_1, x_2, ..., x_n)^T$. Therefore in the input layer, there will be n fan out nodes for an n dimensional input and a single bias node. We assume that in the hidden layer there are k sigmoidal neurons. Let there be m classes, so in the output layer there are m sigmoidal neurons. Let W^{IH} be the set of weights between the input and the hidden layers, and W^{HO} be the set of weights between the hidden and output layers. Therefore, we can write weight matrices as,

$$W^{IH} = [w_{ih}^{IH}]_{(n+1)\times k} \tag{2.1}$$

$$W^{HO} = [w_{hj}^{HO}]_{(k+1)\times m} \tag{2.2}$$

Now we discuss the general model of an MLP [3]. Here the activation function of a neurons is denoted by $\mathcal{S}(\cdot)$.

1. In the input layer at the i^{th} neuron the computation is,

$$\begin{aligned} &\mathcal{S}(x_0) = 1; \\ &\mathcal{S}(x_i) = x_i; i = 1, 2, \cdots, n. \end{aligned} \tag{2.3}$$

Thus, except the bias node, other input nodes just fan out the inputs.

2. In the hidden layer at the h^{th} neuron the computation is,

$$\mathcal{S}(p_0) = 1; \tag{2.4}$$

$$\mathcal{S}(p_h) = \frac{1}{1+e^{-p_h}}; h = 1, 2, \cdots, k \tag{2.5}$$

where

$$p_h = \sum_{i=0}^{n} w_{ih}^{IO} \times S(x_i); h = 1, 2, \cdots, k. \tag{2.6}$$

3. In the output layer at the j^{th} neuron the computation is,

$$\mathcal{S}(y_j) = \frac{1}{1+e^{-y_j}}; j = 1, 2, \cdots, m, \tag{2.7}$$

where

$$y_j = \sum_{h=0}^{h=k} w_{hj}^{IO} \times S(p_h); j = 1, 2, \cdots, m. \tag{2.8}$$

For an input $\mathbf{x}$, let $\mathbf{o} = (o_1, o_2, ..., o_m)^T$ be the desired output and after feeding the input $\mathbf{x}$ into the network at the output layer we found $\mathbf{y} = (y_1, y_2, ..., y_m)^T$. Our objective is to minimize the difference between the actual output $\mathbf{y}$ and the desired output $\mathbf{o}$. So we want to minimize the error, E, where $E = \frac{1}{2}\sum_{i=1}^{m}(y_i - o_i)^2$.

Now we need to modify the set of weights from input to hidden layer and hidden to output layer so that the difference between the obtained output $\mathbf{y}$ and desired output $\mathbf{o}$ is minimised. Here we assume that we are giving the input vectors one by one. So after feeding each input vector into the network, we find the error and we update W^{IH} and W^{HO} with respect to the error. This strategy is called the online update strategy. One can also use a batch mode update strategy to minimize the error [3]. The gradient descent technique is used to modify the weights.

Let, $\Delta W =$ be the amount of modification necessary towards minimization

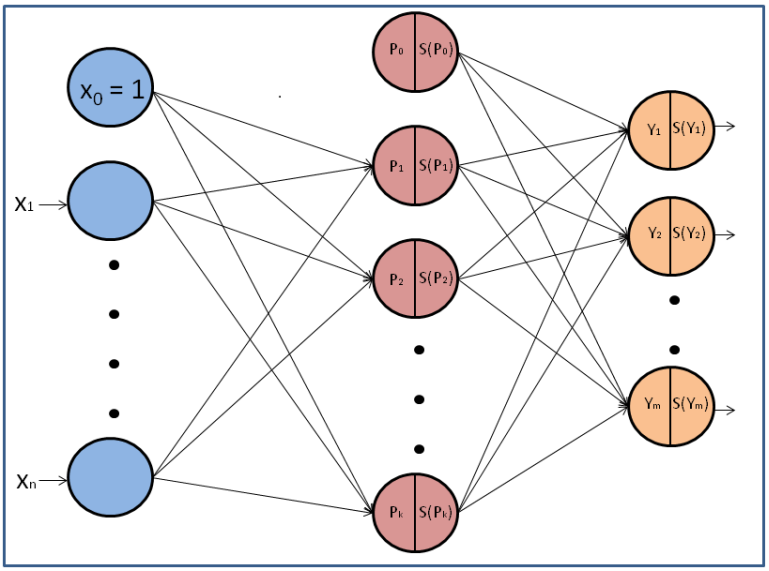

Figure 2.1: Multi-layer Perceptron (MLP) with one hidden layer

of error, E.

$$\Delta W \propto -\frac{\delta E}{\delta W} \tag{2.9}$$

$$\therefore \Delta W = -\eta \frac{\delta E}{\delta W}, \tag{2.10}$$

where η is the learning rate. Now after doing derivative with respect to W^{HO} and W^{IH} we obtain the amount of modification needed for the hidden to output layer links and input to hidden layer links respectively.

The amount of weight update between the l^{th} output layer node and the j^{th} hidden layer node is,

$$\Delta w_{lj} = -\eta(y_l - o_l)y_l(1 - y_l)\mathcal{S}(p_j) \tag{2.11}$$

The amount of weight update between the j^{th} hidden layer node and the i^{th} input layer node is,

$$\Delta w_{ji} = -\eta \sum_{l=1}^{m}(y_l - o_l)y_l(1 - y_l)w_{lj}\mathcal{S}(p_j)(1 - \mathcal{S}(p_j))x_j \tag{2.12}$$

To design our networks we shall make use of some processing similar to

that of Kohonen's self organizing map. Hence, we introduce it next.

2.2 Self-Organizing Maps

The Self-Organizing Maps (SOM) [3] is also a type of artificial neural network (ANN). SOM is trained in an unsupervised way which generates a lower dimensional (usually two or three dimensional) discredited representation of the input training samples, which is called a map. For the learning purpose, unlike error-correction learning which is used in many artificial neural networks, SOM uses competitive learning. Figure 2.2 shows the architecture of a SOM. The visible part of SOM is known as the map space which consists of components called neurons or nodes. Like any other artificial neural network, SOM also works in two modes: Training and mapping. In the "training" mode for input data, it builds the map, whereas in the "mapping" mode it classifies a new input example automatically. The finite two-dimensional map space is defined before it goes to the "training" mode. This is a two-layer network. The first layer is the input layer. If the training data are in an n-dimensional space, the input layer will have n fan-out nodes. The second layer is the competitive layer of nodes. There is complete connection between the two layers. The second layer nodes are arranged on a q-dimensional latice, typically $q = 1, 2$ or 3. With each node, a weight vector is associated which represents a position in the input space, i.e. it has the same dimension as that of the input. The task of "training" is to move the weight vectors towards input data, but without damaging the topology induced from the map space. In the "training" mode the essential processes are given below:

1. **Competition :** For each input pattern, the neurons in the output layer will determine the value of a function. We call that function as discriminant function. So for each neuron in the map space, we compute the discriminant function. This function provides the basis of the competition. So the neuron with the largest discriminant will be the winner.

   Let input vector be n dimensional, so an input vector is $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$.

Let the map space be one dimensional (for simplicity, we can also consider two-dimensional map space). Let there are k neurons in the map space.

So for the j^{th} neuron the weight vector will be, $\mathbf{w}_j = (w_{j1}, w_{j2}, \cdots, w_{jn})^T$ where $j = 1, 2, \cdots, k$.

Now we want to find the best match between $\mathbf{x}$ and $\mathbf{w}_j$. The best matching node with $\mathbf{x}$ that j will emerge as the winner and the corresponding weight will be the winning weight vector. Let,

$$i(\mathbf{x}) = arg \min_j ||\mathbf{x} - \mathbf{w}_j||^2 \tag{2.13}$$

Thus for input $\mathbf{x}$, the i^{th} neuron in the competitive layer is the winner. Note that, in reality, for a 2-D latice each neuron will be identified by two indices. For simplicity, we used one index. This does not cause any problem as we can have a unique mapping between 2-D and 1-D indexes.

2. **Co-operative :** We want to find a neighborhood around the winning neuron (i.e. the i^{th} neuron). For a 1-D lattice defining topological neighborhood is straight forward. For a 2-D lattice we can define different kinds of neighborhood: square, hexagonal, circular and so on. Instead of defining explicit neighborhood we use an implicit neighborhood function. For this, we define topological neighborhood centered around i. We define h_{ji} which is the topological neighborhood centered around i, encompassing neuron j where we are measuring it. This h_{ji} should decrease with the lateral distance (on the lattice), that is the distance between the winning neuron i and neuron j. The lateral distance between i and j is denoted by d_{ji}, where i is the winning neuron and j is the excited neuron which is excited as an effect of the winning neuron. Here a typical choice is to use Gaussian type neighborhood function i.e.

$$h_{ji} = exp^{-\frac{d_{ji}^2}{2\sigma^2}} \tag{2.14}$$

This does not depend on the position of the winning neuron, so it is translation invariant.

In the case of 1-D lattice the $d_{ji} = |j - i|$. In case of 2-D lattice $d_{ji}^2 = ||r_j - r_i||^2$; where, $\mathbf{r}_j$ = coordinate of the excited neuron j and $\mathbf{r}_i$ = coordinate of the winning neuron i.

Now as the iteration progresses σ decreases as:

$$\sigma(t) = \sigma_0 - t\frac{\sigma_0 - \sigma_f}{maxstep} \tag{2.15}$$

So now (2.14) become

$$h_{ji}(t) = exp^{-\frac{d_{ji}^2}{2\sigma^2(t)}}; t = 0, 1, 2, \cdots, maxstep. \tag{2.16}$$

Here, maxstep is the number of times we want to iterate with the dataset in the network.

3. **Synaptic Weight Adaptation :** It enables the excited neurons to increase their discriminant function in response to the input example which caused the winning of the neurons. Here we generally use Hebbian learning. It means that when the pre-synaptic and post-synaptic are correlated then the synaptic connection is strengthened, if they are not correlated then the synaptic connection is weekend.

   We can take $g(\cdot)$ to be a linear function of y_j (for simplification), i.e., $g(y_j) = y_j$. Now we can take $y_j = h_{ji}$. Normal way of writing the Hebbian hypothesis is

$$\Delta\mathbf{w}_j = \eta y_j \mathbf{x} \tag{2.17}$$

   Now introducing forgetting term,

$$\Delta\mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j \tag{2.18}$$

   Now if we take $g(y_j) = \eta y_j$, therefore, $\Delta\mathbf{w}_j = \eta y_j \mathbf{x} - \eta y_j \mathbf{w}_j$. In case

of a 2-D lattice $g(y_j) = \eta h_{ji}$. Therefore,

$$\Delta \mathbf{w}_j = \eta h_{ji} \mathbf{x} - \eta h_{ji} \mathbf{w}_j \tag{2.19}$$

$$= \eta h_{ji} (\mathbf{x} - \mathbf{w}_j) \tag{2.20}$$

Using discrete time formulation we can write it as,

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta(t) h_{ji}(t) (\mathbf{x} - \mathbf{w}_j(t)) \tag{2.21}$$

Here we take η as a function of t. This is the updation rule. Here we have two time varying factors $\eta(t)$ and $h_{ji}(t)$.

Now we can update $\eta(t)$ by using the formula, $\eta(t) = \eta_0 \left(1 - \frac{t}{maxstep}\right)$.

Now, using eq. (2.21) we update the weights. Note that using this update rule, the winning node's weight vector gets the maximum amount of update and it move closer to the input $\mathbf{x}$. All other weight vector also move closer to the input, but for them the strength of update reduces as the distance of the node from the winner on the lattice increases.

Typically SOM learning is done in two phases. In the first phase the weights of the winner and its neighbors are updated as explained above. Then in the second phase only the winner is updated for some iterations.

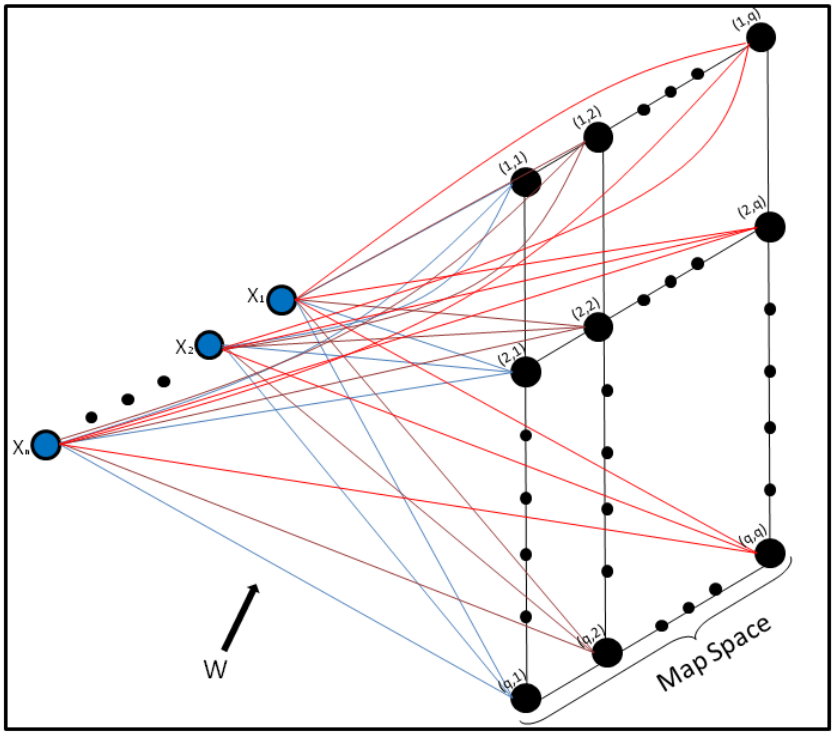

Figure 2.2: Self Organizing Maps (SOM)

Chapter 3

Related Works

Here we concentrate mostly on some existing works that are related to our work.

In [5] Laurent et al. presented a visual attention system which is inspired by the behavior and the neuronal architecture of the early primate visual system. Into a single topographical saliency map, multiscale image features are combined. In order to decrease saliency, a dynamical neural network then selects the attended locations. In a computationally efficient manner, the system breaks down the complex problem of scene understanding by rapidly selecting conspicuous locations.

Ciresan et al. in [6] proposed a biologically plausible deep artificial neural network architecture. Where they show that their model can match human performance on tasks such as handwritten digits recognition or traffic signs recognition unlike traditional methods of computer vision and machine learning which cannot. Small receptive fields of convolutional winner-take-all neurons yield large network depth which results in roughly as many sparsely connected neural layers as found in mammals between the retina and visual cortex. There they trained only winner neurons. Here the inputs are pre-processed in different ways and then given to several deep neural columns which become experts and then their predictions are averaged. This is the first time on a widely used computer vision benchmarks human-competitive results are reported.

In [7] Glorot et al. show that rectifying neuron is a better model of biological neurons and it gives equal or better performance than hyperbolic tangent neurons. Their results can be seen as a milestone in the attempts to understand the difficulty in training deep but purely supervised neural networks and also closing the performance gap between the neural networks learned with and without unsupervised pre-training.

Now we discuss some of the relevant experiments done on animals and humans.

Kreiman et al. in [8] mentioned that the entorhinal cortex, amygdala, and hippocampus obtain inputs from the temporal neocortical regions. These inputs are convergent and specialized for processing complex visual stimuli. Therefore, they are critical in the recognition and representation of visual information. In this experiment 427, single neurons are examined which are in human hippocampus, entorhinal cortex, and amygdale. As a result, they found a remarkable degree of category-specific firing of individual neurons on a trial-by-trial basis.

In [9] Quiroga et al. mentioned that when we see a person or an object it takes a fraction of second to recognize even when it has been seen under strikingly different conditions. Still, it is unclear how such robust high-level representation is achieved by neurons in human brain. It has been shown that neurons in the human medial temporal lobe (MTL) fire on category-specific inputs. Here they report on a remarkable subset of MTL neurons which are selectively activated when different pictures of some given landmarks, individuals or objects are presented. This experiment is done on 5 participants. They recorded the activation pulse of 993 neurons from each participant.

Chapter 4

Proposed Scheme

In this section, we propose a model to meet our objective that has been already discussed. Here we first describe the architecture of our model. Figure 4.1 shows the architecture. Figure 4.1 shows four layers including the input and output layers. This is the minimum number of layers the network can have. Between the second and the output layers there can be more than one hidden layer. In this model, we integrate concepts of multilayered perceptron and self-organizing maps. It should be noted that we could use multiple hidden layers but for simplicity, we use one hidden layer. In this model we have four layers these are input layer, spatial activation and intermediate feature extraction layer (SAIFEL), discriminatory feature extraction layer (DFEL) and output layer. Like any other neural network model here we first train the model then we test it. We train the network in two different phases. First, we train the full network and then we train a part (the discriminatory part) of the network to improve the classifier performance. Let the dimension of the input vector be n. So in the network, the number of nodes in the input layer is n. The second layer (SAIFEL) has two component: spatial activation layer (SAL) and intermediate feature extraction layer (IFEL). Now let in the SAL there be $k = q \times q = q^2$ hidden nodes arranged on a 2-D lattice like a SOM. Then in IFEL also there are k nodes and there is a one-to-one correspondance between the two sets of nodes. The output of the i^{th} SAL node will be denoted by s_i and the output of i^{th} IEFL node will be denoted by p_i. So on the

second layer, we have total $2k$ nodes. In fig. 4.1 the $2k$ nodes are shown on a 2-D lattice. Now after that, we have DFEL layer with k nodes. We will discuss the usage of this layer later. The last layer in our model is output layer. If we have m classes in the dataset, then in the output layer we have m nodes. Now we will discuss the training in details.

In the first part of the training we train the model in online learning mode, that means that after feeding each input we modify the network weights. Let an input vector be $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$. So after feeding the input $\mathbf{x}$ into the network first, we need to find the output of the SAL nodes and the IFEL nodes. The output for the i^{th} SAL node is computed as:

$$s_i = \exp^{-\frac{\sum_{j=1}^{n}(w_{ij} - x_j)^2}{c}}; i = 1, 2, \cdots, k. \tag{4.1}$$

Where, $c > 0$ is a constant. Now the output of the h^{th} IFEL node where $h = 1, 2, \cdots, k$ is computed as follows:

$$p_0 = 1;$$
$$p_h = \frac{1}{1 + e^{-u_h}}; h = 1, 2, \cdots, k$$

where

$$u_h = \sum_{i=0}^{n} w_{ih}^{IO} \times \mathcal{S}(x_i); h = 1, 2, \cdots, k. \tag{4.2}$$

Here $\mathcal{S}(\cdot)$ is as in (2.3). As mentioned earlier the third layer, i.e., the DFEL contains k nodes. The i^{th} node in the 3^{rd} layer is connected to the i^{th} node of the SAL and IFEL. The activation function of a neuron computes the product of its inputs. All connection weights between second layer and third layer are 1. Thus the output of the i^{th} node of the DFEL is computed as:

$$v_i = p_i \times s_i; i = 1, 2, \cdots, k \tag{4.3}$$

Every neuron of third layer is connected to every neuron of the fourth layer (output layer), i.e., it is fully connected. Suppose at the output layer we get

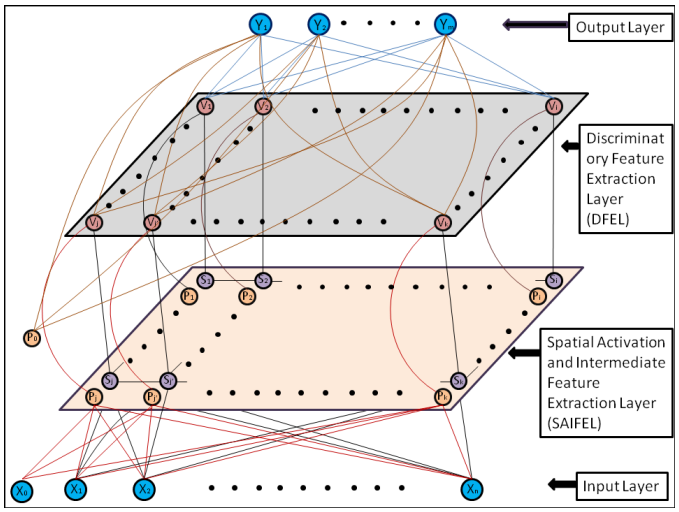

Figure 4.1: The proposed model

$\mathbf{y} = (y_1, y_2, \cdots, y_m)^T$, and our desired output is $\mathbf{o} = (o_1, o_2, \cdots, o_m)^T$. So the square error is $||\mathbf{y} - \mathbf{o}||^2$. We update the weights in the same way as we described in Section 2.1 to minimize the square error here.

In the second phase of the training it is same like the first phase except in the first phase we update all weights between input layer and the second layer (SAL and IFEL), but here we do not update the weights between input and SAL. So if $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$ is the input vector then we find the output from a SAL node using eq. (4.1). Then we compute the output of an IFEL node using eq. (4.2). After that we compute the DFEL output using eq. (4.3). Finally, we compute outputs in the output layer and depending on the error between the actual output and desired output we update the weights. The algorithm is summarised in Algorithm 1.

4.1 Results

Here we use MNIST dataset to do our experiment. We took only characters 0, 1, 7, and 8. For each character in the MNIST dataset, there are 6000 examples in training data and 1000 examples for testing. So total we have 7000 examples for each class. Now we use four classes (i.e. for digit 0, 1, 7, and 8). So in total, we have 28000 data points. We use five-fold cross-validation

mechanism. Here we randomly divide the dataset into five mutually exclusive subsets, where the number of data points in each subset is the same. We use four subsets (folds) for training and the remaining subset (fold) for testing. We use all unique combination 4 folds for training and remaining one for testing. So in total, we have five test accuracy results. We repeat the 5-fold experiment 5 times and report the average result. We have also done another experiments using an ordinary MLP for comparison.

In the first experiment, we use an MLP with one hidden layer and took $225 = (15 \times 15)$ nodes in the SAL and 225 nodes in the IFEL. So in this network, the number of nodes in the input layer is 784. The number of nodes in the hidden layer is 550 divided into two sets each of 225. The number of nodes in the output layer is 4. Then we train the network using the dataset for 2000 iterations.

First we want to investigate if for different kinds of stimuli (characters), different spatial regions of the SAL are activated or not. Note that here in every step the SAL influences the classification part of the network,so it is no more remains an MLP because its inputs are perturbed by the SAL output and this step may not be consistent with minimization of the square error.

For this network, we shall analyze two kinds of outputs: the responses in the DFEL nodes and the classification accuracy. To visualize the response behavior of the DFEL, we proceed as follows.

The DFEL has k nodes, the response from the i^{th} node for an input $\mathbf{x}$ be $v_i, i = 1, 2, \cdots, k$. Since the i^{th} node is connected with the $(l, u)^{th}$ node of the SAL, i.e., there is a one-to-one correspondence between the one-dimensional index of any node in DFEL and a node in SAL, we can use the two dimensional index (l, u) to indicate the i^{th} node. Thus,

$$v_{l,u} = v_i; \text{ where,} \quad (4.4)$$
$$l = 1, 2, \cdots, q$$
$$u = 1, 2, \cdots, q$$
$$i = (l-1)q + u$$

Now we take m arrays, $A_j; j = 1, 2, \cdots, m$ each of size $q \times q$. Each cell of every array is initialized to zero. To visualize the response surface of DFEL we compute as shown in algorithm 2.

In the first experiment, we use our model. Here in the input layer, we have 784 nodes ($n = 784$). In the SAL we have 15×15 neurons ($k = 225$), consequently IFEL also has $k = 225$ nodes. So in the hidden layer DEFL, we also have 225 neurons and at the output, we have 4 neurons ($m = 4$). We train the model as discussed earlier. We train the network in two different phase as explained earlier. In the first phase we train the whole network for 2000 iterations and in the second phase we train the discriminatory part of the network for 20000 iterations. Now for each of the testing data, we find the output of DEFL using Algo. 2. We show this outputs as a 15×15 image in Fig. 4.3 and in Fig. 4.4 for typical runs (for two different fold). Figure 4.3 and Fig. 4.4 both have four panels, one for each of the four classes. Here we can see that for each class there is almost a separate set of neurons that is activated. And in the region where the neurons are activated for a specific class, all the neurons are not activated with full power all the time. The activation strength of a neuron gives the extent it contributes towards making the decision. We find that there are some neurons which are always fully activated (i.e. $v_i = 1$) but there others which are partially activated like $v_j = 0.160$.

Next we have reported the same experiment using a conventional neural network with 225 (15×15) hidden nodes. Without any loss, we can also view these 225 neurons to be arranged in 15×15 grid.

For this MLP we show the hidden layer responses as a 15×15 image Fig. 4.2 (the image is formed in row major order). The top-left image is for

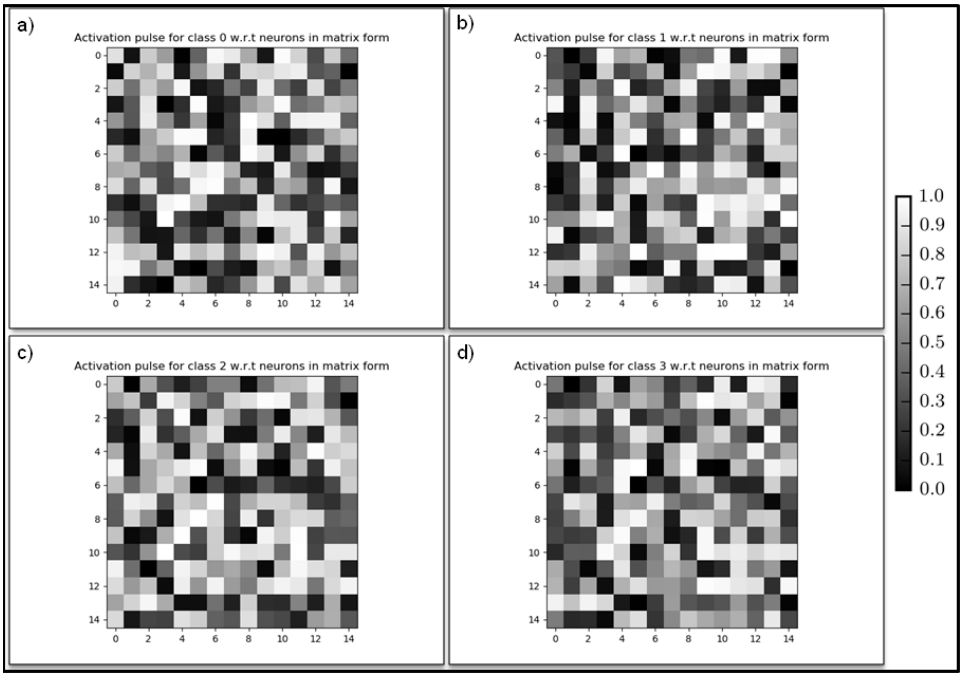

Figure 4.2: Each image is the average output of the hidden nodes for each class. a) is for digit 0. b) is for digit 1. c) is for digit 7. d) is for digit 8 for the MNIST 0 1 7 8 dataset.

digit 0, top-right image is for digit 1, bottom-left image is for digit 7 and bottom-right image is for digit 8. It is very clear that the active neurons for each class is distributed randomly without revealing any local structure (i.e., without forming a spatial cluster of neurons for a particular class).

The accuracy of this two experiment is shown in Table 4.1, which is calculated using the mechanism discussed previously.

We have also run this proposed model on iris dataset. The experimental setup is same as we explained previously. Here in the input layer, we have 4 nodes ($n = 4$). In the SAL we have 5×5 neurons ($k = 25$), consequently IFEL also has $k = 25$ nodes. So in DFEL, we have 25 neurons and at the output, we have 3 neurons ($m = 3$). We train the network in two different phase as explained earlier. In the first phase we train the whole network for 100 iterations and in the second phase, we train the discriminatory part of the network for 1000 iterations. Now for each of testing data, we find the output of DEFL using Algo. 2. We show the output in the same

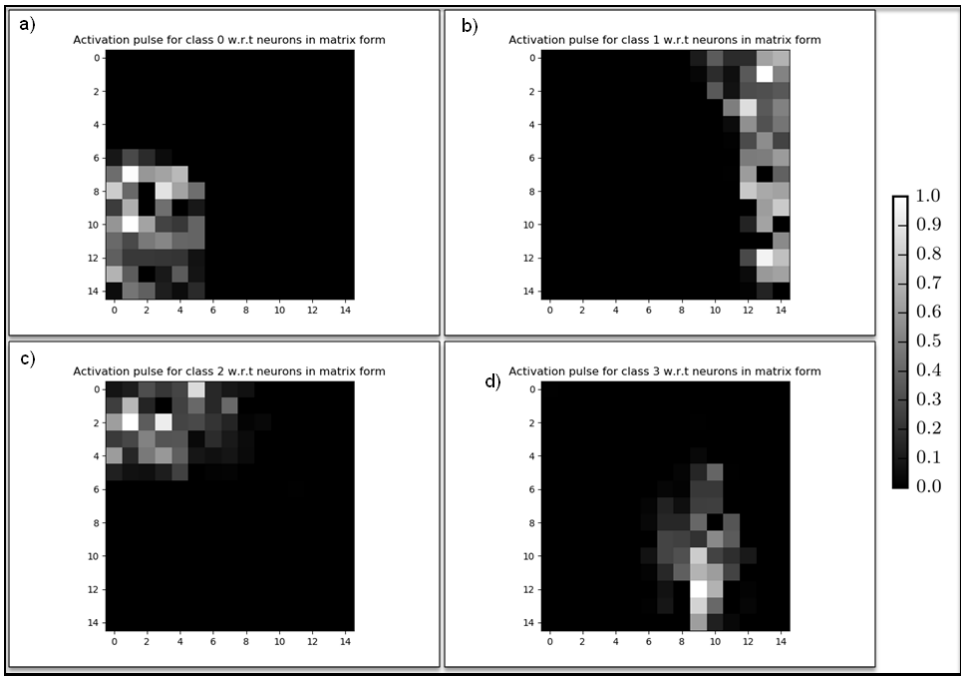

Figure 4.3: Each image is the average output of the DFEL nodes for each class for MNIST 0 1 7 8 dataset. a) is for digit 0. b) is for digit 1. c) is for digit 7. d) is for digit 8. for a run

manner as we discussed previously, it is shown in Fig. 4.6 for a typical run (for a fold). Now we repeated the same experiment using a conventional neural network with 25 (5×5) hidden nodes. Without any loss, we can also view these 25 neurons to be arranged in 5×5 grid. It is shown in Fig. 4.5. The accuracy of this two methods is shown in Table 4.2. In Table 4.2 we also show the accuracy on wdbc, glass datasets.

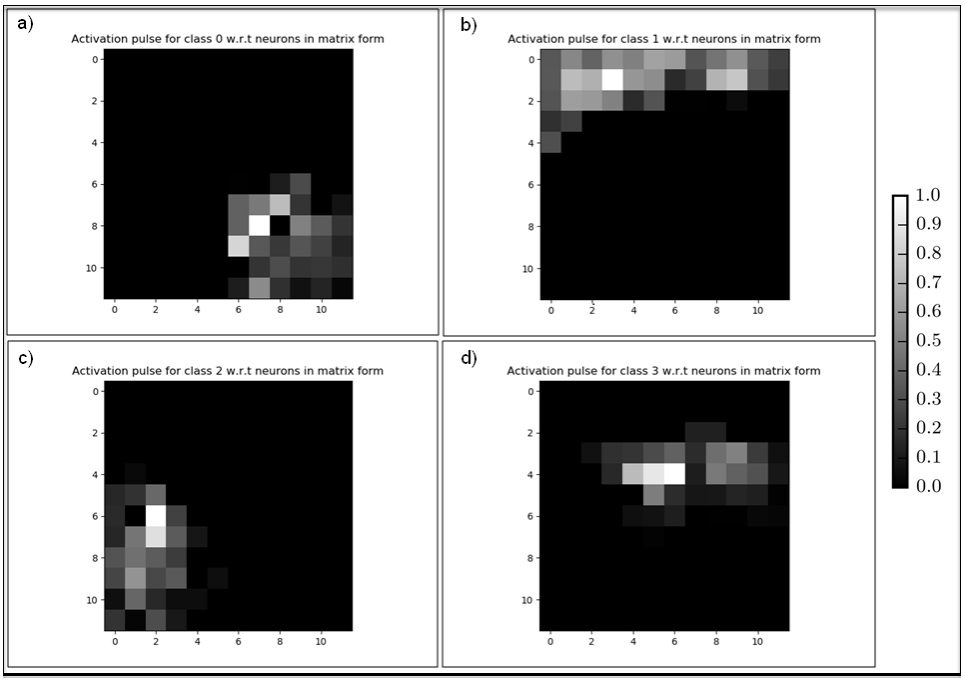

Figure 4.4: Each image is the average output of the DFEL nodes for each class for MNIST 0 1 7 8 dataset. a) is for digit 0. b) is for digit 1. c) is for digit 7. d) is for digit 8. for a run

Algorithm 1: The proposed model

Data : Dataset D
Result : A trained network
Assumption : We have N data points in the dataset D
Assumption : We train the whole network for "MAXITER" number of times, and only the subpart for "MAXITERSUB" number of times
Assumption : The dimension of each of the input in the dataset is n.
Initialize : Randomly initialize all the weights of the network , i = 0, iteration = 0
while *iteration* $<$ *MAXITER* **do**
    while $i < N$ **do**
        Take i^{th} input vector form D. Let it is $\mathbf{x} = (x_1, x_2, \cdots, x_n)$
        Feed $\mathbf{x}$ to the Network
        At the SAL we get s_i's and p_i's
        Update the weights between input and SAL using eq. (2.21)
        Now at the integration layer we compute v_i's by doing $v_i = s_i \times p_i$
        Now compute the outputs in the output layer and find the error
        Then back-propagate and update the weights between the output layer and SAL and between the SAL and input layer using eq. (2.11) and eq. (2.12) respectively;
    end
end
iteration = 0
while *iteration* $<$ *MAXITERSUB* **do**
    while $i < N$ **do**
        Take i^{th} input vector form D. Let it is $\mathbf{x} = (x_1, x_2, \cdots, x_n)$
        Feed $\mathbf{x}$ to the Network
        At the SAL we get s_i's and p_i's
        Now at the integration layer we compute v_i's by doing $v_i = s_i \times p_i$
        Now compute the outputs in the output layer and find the error
        Then back-propagate and update the weights
    end
end
return the trained network

Algorithm 2: Visualization of responses of DFEL

Assumption : We have N data points in the dataset D
Assumption : In the DFEL layer we have $q \times q$ nodes
Assumption : We have m classes. Among N data points, N_i is from class i; $\sum_{i=1}^{m} N_i = N$
for $i = 1, 2, \cdots, N$ **do**
    Apply $\mathbf{x}_i$ to the network
    Compute the response $v_{l,u}$ for $l = 1, 2, \cdots, q$; $u = 1, 2, \cdots, q$
    if $\mathbf{x}_i \in$ *class* j **then**
        for $l = 1, 2, \cdots, q$ **do**
            for $u = 1, 2, \cdots, q$ **do**
                $A_j(l, u) = A_j(l, u) + v_{l,u}$
            end
        end
    end
end
for $j = 1, 2, \cdots, m$ **do**
    for $l = 1, 2, \cdots, q$ **do**
        for $u = 1, 2, \cdots, q$ **do**
            $A_j(l, u) = A_j(l, u)/N_j$
        end
    end
end

Table 4.1: Accuracy obtained on MNIST 0 1 7 8 dataset

Dataset	Multi-layer perceptron with one hidden layer	Our model
MNIST 0 1 7 8	97.7418	97.2124
MNIST 0 1 7 8	98.1431	97.8981
MNIST 0 1 7 8	97.8469	97.8413
MNIST 0 1 7 8	97.8817	97.6714
MNIST 0 1 7 8	98.0124	97.8751
Average	97.9251	97.6996

Table 4.2: Accuracy obtained on other datasets

Dataset	Multi-layer perceptron with one hidden layer	Our model
IRIS	99.124	99.086
WDBC	96.421	96.121
GLASS	62.7901	62.261

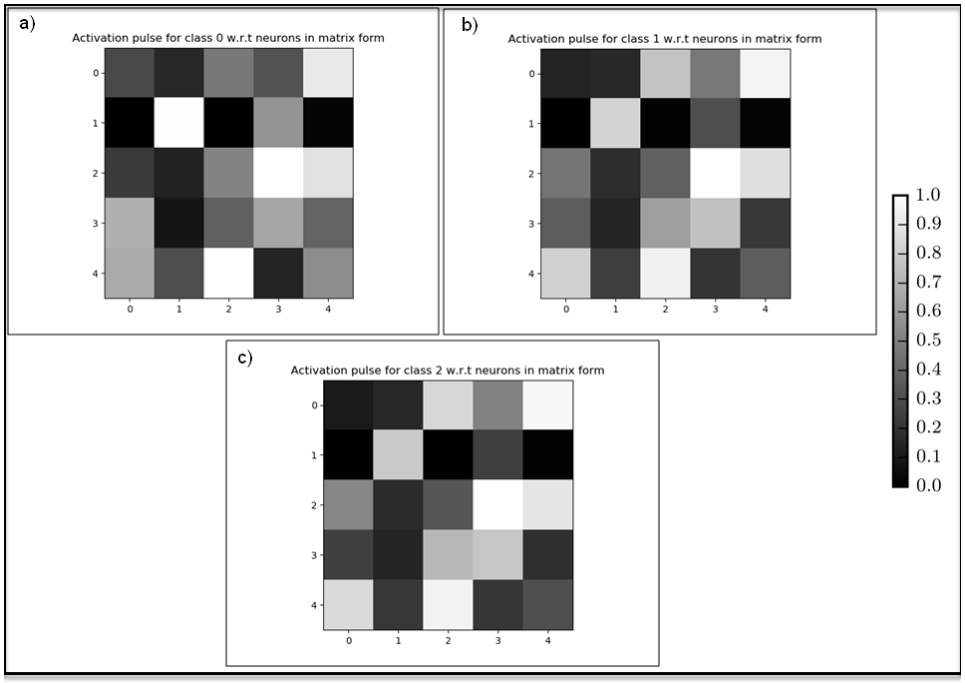

Figure 4.5: Each image is the average output of the DFEL nodes for each class of Iris. a) is for Iris-setosa. b) is for Iris-versicolor. c) is for Iris-verginica.

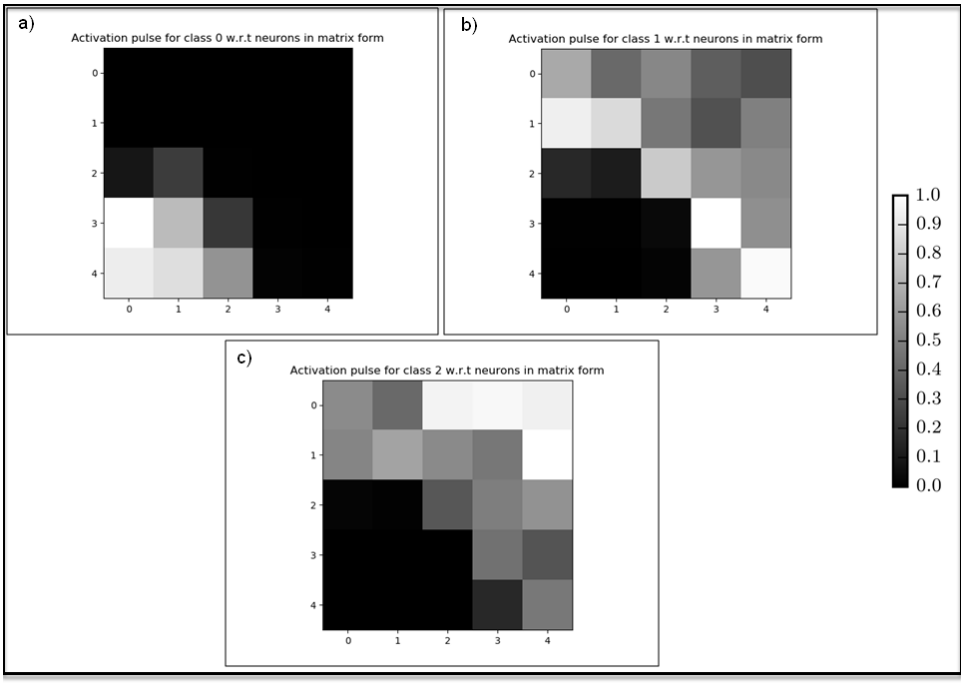

Figure 4.6: Each image is the average output of the DFEL nodes for each class of Iris. a) is for Iris-setosa. b) is for Iris-versicolor. c) is for Iris-verginica.

Chapter 5

Conclusion

Though artificial neural networks (ANNs) are inspired by human brains, there are notable differences between how present day ANNs and human brains work. For example, researchers [8, 9] have shown that for different visual stimuli clusters of neurons from different parts of the brain get activated. Being inspired by this observation, here, we have proposed a model integrating a multilayer perceptron and a self-organizing map. In this context, we have introduced two new layers: a spatial activation layer (SAL) and an intermediate feature extraction layer (IFEL). Together these two layers comprise a new layer, called spatial activation and intermediate feature extraction layer (SAIFEL). After that, we have a discriminatory feature extraction layer (DFEL). Basically, at DFEL pairwise multiplication of the corresponding outputs from the SAL and the IFEL are computed. Then form DFEL we have full connectivity to the output layer where we predict the classes. Moreover, we have trained the proposed model in two different phases. In the first phase, we have trained all the parameters of the model, and then, in the second phase, we have trained only the discriminatory part of the proposed model. From the experimental results, we have observed that for inputs corresponding to different classes, different sets of neurons get excited. We have, nonetheless, observed that the same set of neurons get activated for different inputs that correspond to the same class. Note that, this phenomenon is not observed in MLP. We found that the accuracy of the proposed model and that of MLPs are similar. The MLP, however,

needs a smaller number of iterations compared to our model.

Similar to most of the works, this work too has some limitations. First, we have investigated four datasets. The above-mentioned observation needs to be validated for a larger number of data sets. Second, for a given task, we need a higher number of iterations to train the proposed model compared to the number of iterations that we need to train an MLP. In other words, the proposed model is computationally more costly compared to an MLP. Third, we have not investigated the parameter sensitivities of the proposed model. Fourth, though the intension of this work is not to design better classifier but to realize a network which are a bit closer to the biological neural network, the performance of the proposed model was comparable to that of MLPs and was not better than the same. In future, we plan to address these limitations.

Bibliography

[1] "https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106."

[2] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, p. 13276, 2016.

[3] S. Kumar, "Neural networks: A classroom approach," 2004.

[4] K. Ohki, S. Chung, Y. H. Ch'ng, P. Kara, and R. C. Reid, "Functional imaging with cellular resolution reveals precise micro-architecture in visual cortex," *Nature*, vol. 433, no. 7026, p. 597, 2005.

[5] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.

[6] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*. IEEE, 2012, pp. 3642–3649.

[7] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[8] G. Kreiman, C. Koch, and I. Fried, "Category-specific visual responses of single neurons in the human medial temporal lobe," *Nature neuroscience*, vol. 3, no. 9, p. 946, 2000.

[9] R. Q. Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried, "Invariant visual representation by single neurons in the human brain," *Nature*, vol. 435, no. 7045, p. 1102, 2005.