CSL603 - Lab 3
Multi Layer Perceptron

Aditya Gupta
2015CSB1003
Soumyadeep Roy
2015CSB1035

November 5, 2017

1 2-Dimensional 3-Class Classification problem

1.1 Goal

Study the changes to the

- decision boundary and
- the training error

with respect to parameters such as

- number of training iterations,
- number of hidden layer neurons and
- finally the learning rate

1.2 MLP Training

During forward pass, we made $z_h = \sigma(w_h^T x)$, $y'_k = \exp(v_k^T z)/\sum_{k'=1}^{K}\exp(v_{k'}^T z)$ (The weights and points had different orientation to what discussed in class, therefore an appropriate formula equivalent to these was used.) During back-propagation, for a particular point we used

$$\Delta v_{hk} = \underbrace{\eta(y'_k - y_k)}_{\text{Common for a particular } k} \qquad z_h$$

,

$$\Delta w_{jh} = \underbrace{\eta\left(\sum_k (y'_k - y_k)v_{hk}\right) z_h(1-z_h)}_{\text{Common for a particular } h} x_j$$

(Various values were calculated only once which were common for a particular k for Δv_{hk} and these were further used in case of Δw_{jh}, moreover the common values in case of Δw_{jh} for a particular h were also calculated only once.)

1.3 Training Error

The training error was calculated as $E(w, v) = -\sum_{k=1}^{K} y_k \log y'_k$ for a particular point that was trained in this epoch.

1.4 MLP Testing

The similar formulas as for MLP training were used but here, instead we had multiple points, so an appropriate form was used (The bias term was introduced differently which in this case would be a column vector, rest was same).

1.5 Observations

1.5.1 Varying the number of hidden layer nodes

The following figures depict the change in the sum of squared error vs. the number of training iterations for a particular η value and varying the number of hidden layer nodes in powers of 2:

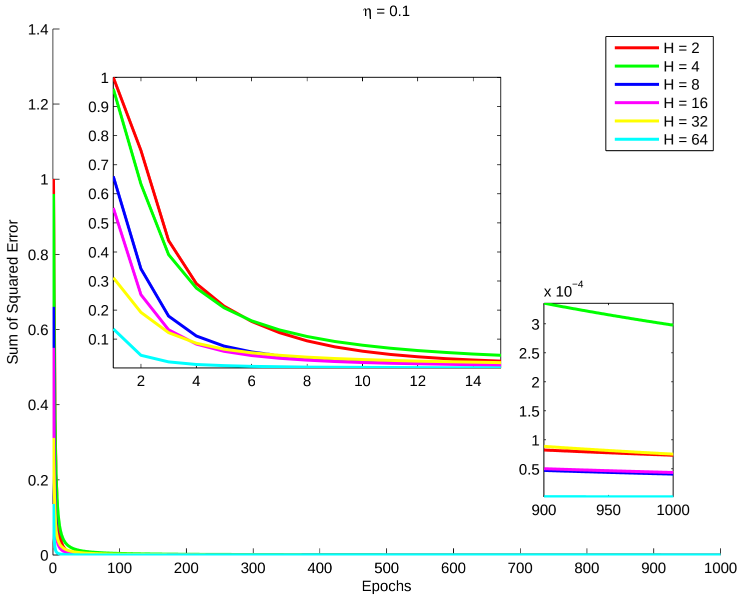


Figure 1: Change in Sum of Squared Error vs the Training Iterations for different values of H. $\eta = 0.1$

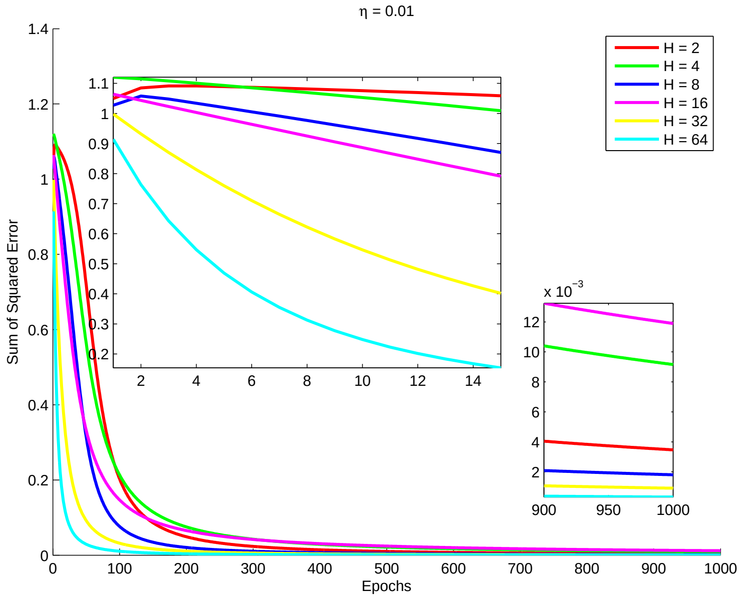


Figure 2: Change in Sum of Squared Error vs the Training Iterations for different values of H. $\eta = 0.01$

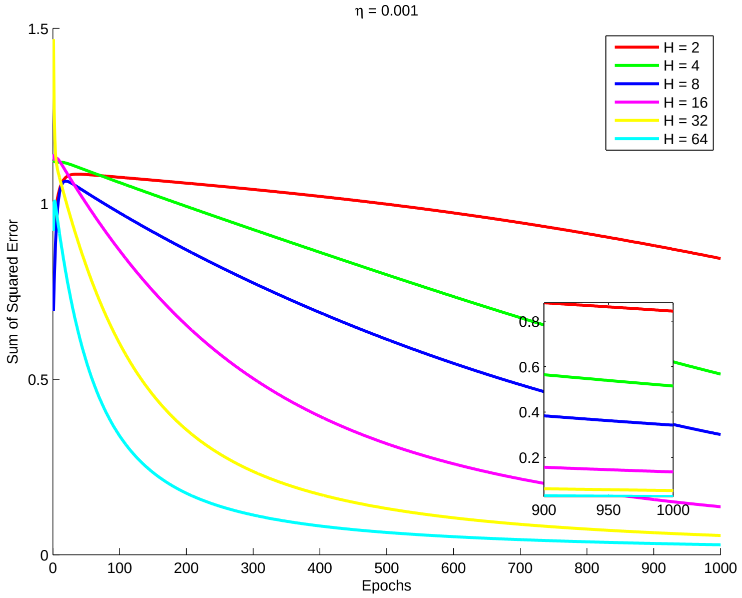


Figure 3: Change in Sum of Squared Error vs the Training Iterations for different values of H. $\eta = 0.001$

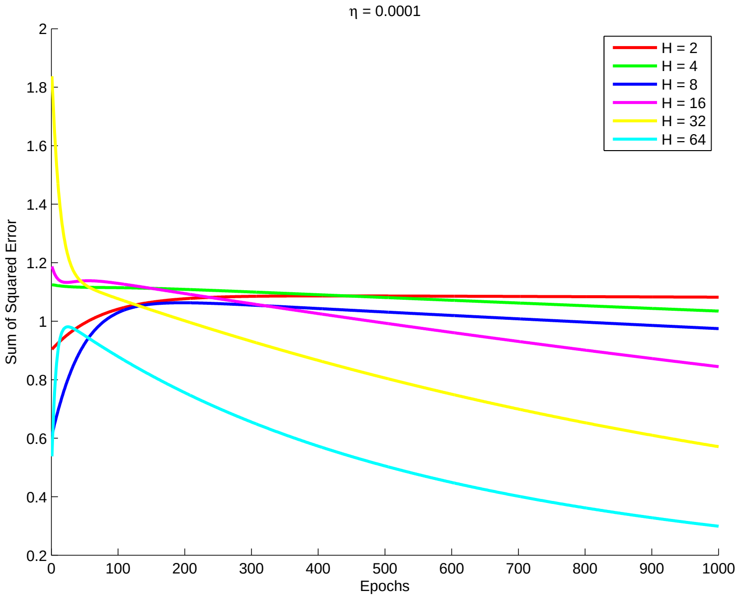


Figure 4: Change in Sum of Squared Error vs the Training Iterations for different values of H. $\eta = 0.0001$

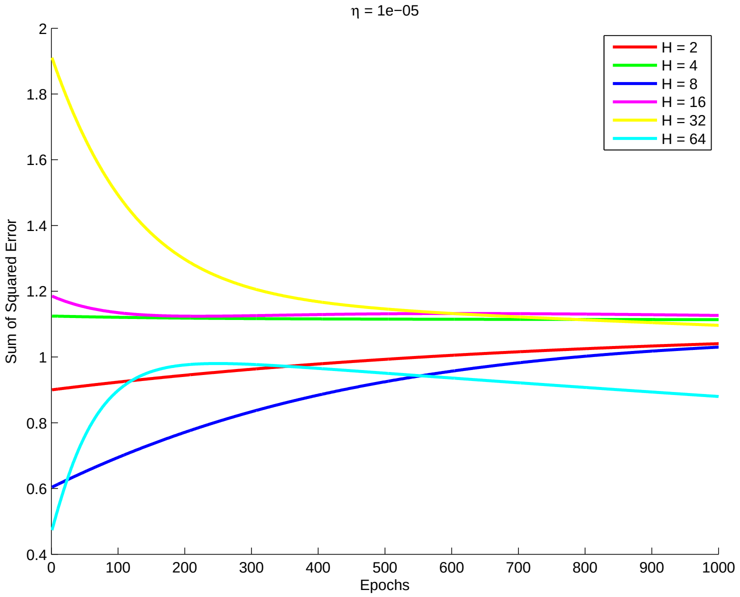


Figure 5: Change in Sum of Squared Error vs the Training Iterations for different values of H. $\eta = 0.00001$

1.5.2 Varying the learning rate

The following figures depict the change in the sum of squared error vs. the number of training iterations for a particular value of H and varying the η:

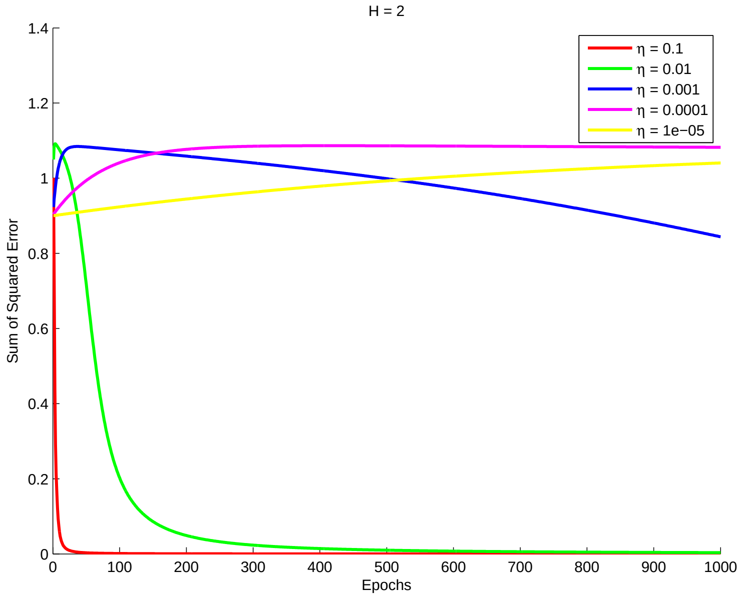


Figure 6: Change in Sum of Squared Error vs the Training Iterations for different values of η. $H = 2$

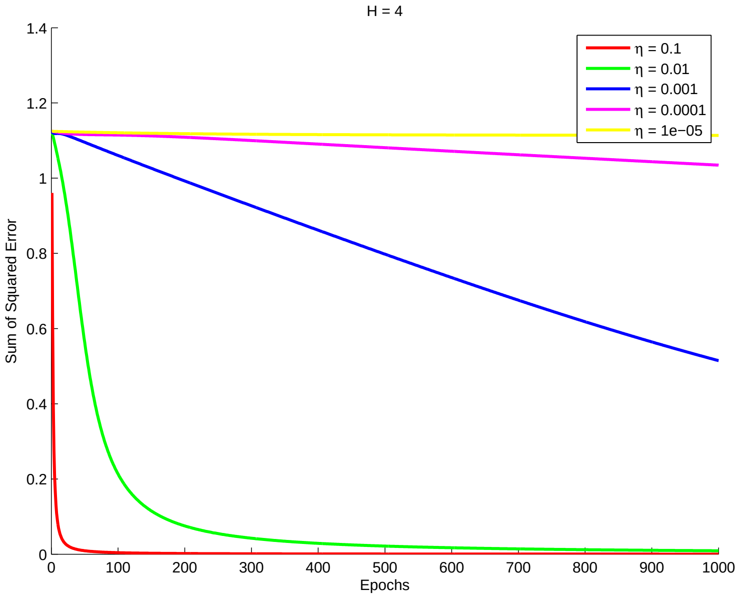


Figure 7: Change in Sum of Squared Error vs the Training Iterations for different values of η. $H = 4$

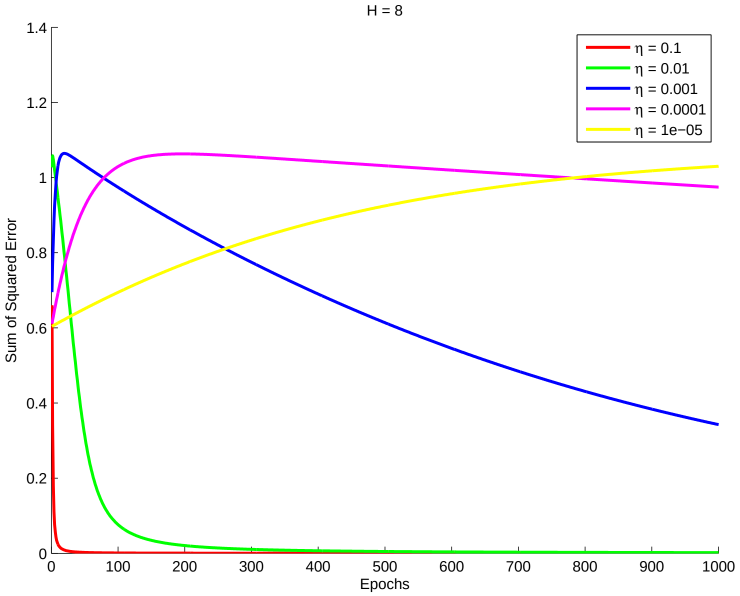


Figure 8: Change in Sum of Squared Error vs the Training Iterations for different values of η. $H = 8$

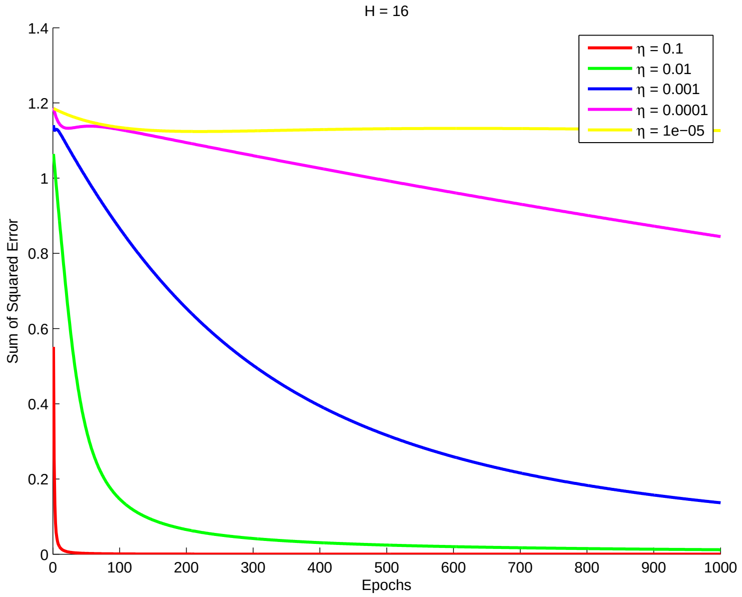


Figure 9: Change in Sum of Squared Error vs the Training Iterations for different values of η. $H = 16$

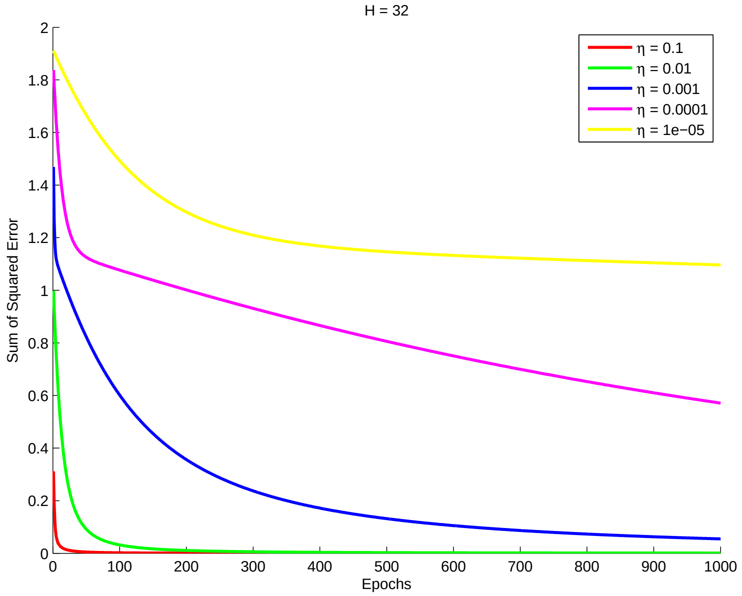


Figure 10: Change in Sum of Squared Error vs the Training Iterations for different values of η. $H = 32$

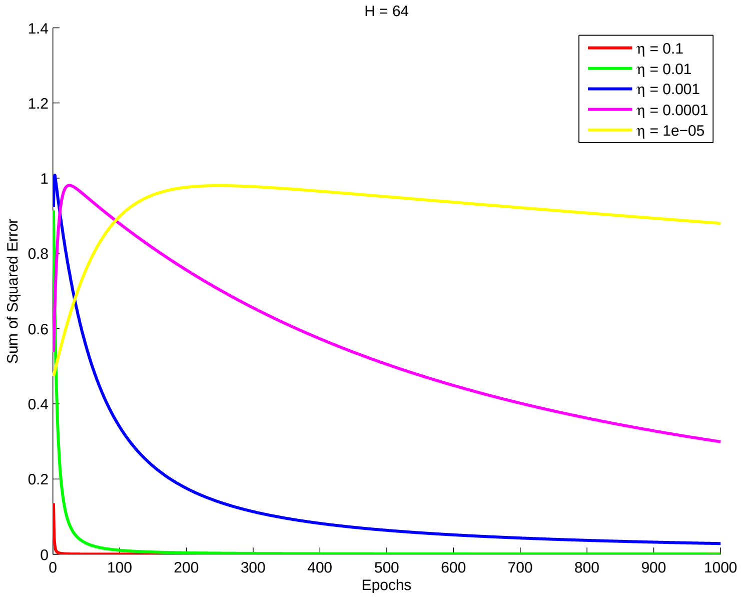


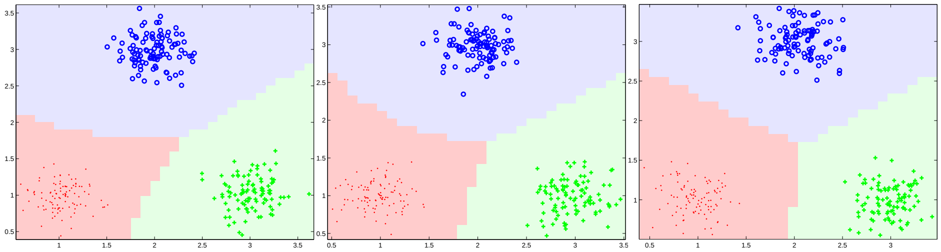
Figure 11: Change in Sum of Squared Error vs the Training Iterations for different values of η. $H = 64$

1.5.3 Varying the training iterations

The observations for the change in the sum of squared error by varying the training iterations can be made through the previous graphs.

1.5.4 Change in Decision Boundary

The change in decision boundaries for various values of H and η can be seen in the following graphs. Here red dots represent Class 1, green pluses represent Class 2 and blue circles represent Class 3.



(a) $H = 2$ (b) $H = 4$ (c) $H = 64$

Figure 12: Deicision boundary for $\eta = 0.01$

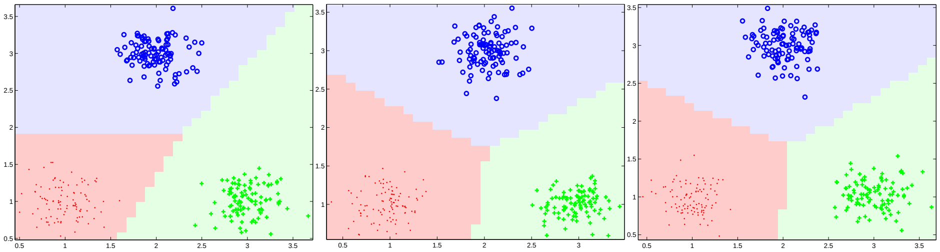


Figure 13: $H = 2$ Figure 14: $H = 4$ Figure 15: $H = 64$

Figure 16: Deicision boundary for $\eta = 0.001$

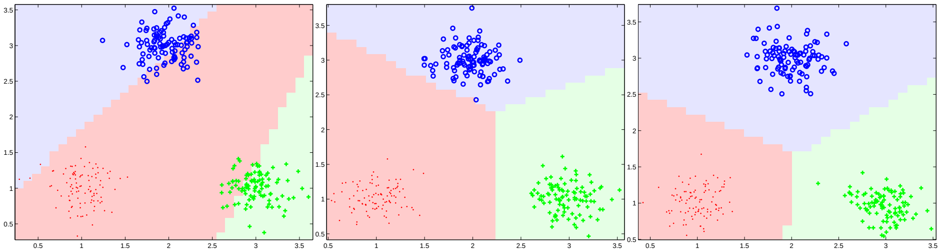


Figure 17: $H = 2$ Figure 18: $H = 4$ Figure 19: $H = 64$

Figure 20: Deicision boundary for $\eta = 0.0001$

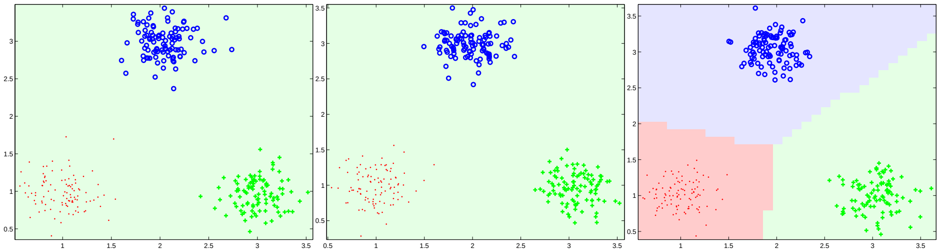


Figure 21: $H = 2$ Figure 22: $H = 4$ Figure 23: $H = 64$

Figure 24: Deicision boundary for $\eta = 0.00001$

1.6 Discussion and Conclusions

1.6.1 Hidden Layer Nodes

As we can see from Figure 1 and 2 increasing the number of hidden layer nodes increases the convergence rate for the neural network but in the end, all the networks become almost the same given sufficient iterations, although $H = 64$ still dominates in all cases. However, in case of lower learning rate, networks with more number of hidden layer nodes are able to converge quickly in comparison to others as can be seen in Figures 3, 4 and 5. This is because the network will be able to represent more complex data with an increase in the number of hidden layer nodes which will represent various intermediate functions and representation of-of data (i.e. classification into classes) can be done quickly with more hidden layer nodes rather than taking more iterations with less number of hidden layer nodes. Also, this process will saturate and beyond a particular limit increasing the hidden layer nodes will not make any significant difference.

1.6.2 Learning Rate

As we can see from Figure 6, 7, 8, 9, 10 and 11; increasing the learning rate leads to faster convergence. This is because the magnitude of the learning rate hasn't been too high to cause oscillations in the error space and hence increasing the learning rate within particular limits will only fasten the process of convergence of the network to a better solution.

1.6.3 Training Iterations

We are performing a stochastic gradient descent on the data which is based on finding the gradient with respect to a particular point and changing weights accordingly, this method will give better and better results as network is trained with more and more number of points, i.e. with increase in total iterations the error in the output of the network decreases as can be seen in all the Figures.

1.6.4 Decision Boundary

As we can see for appropriate learning rate all networks (i.e. varying them respect to the number of hidden layer nodes) give a very accurate description of decision boundary though they differ slightly in term of the structure and also the sum of squared errors. The reason for the latter is of the ability of networks with more complexity (more hidden layer nodes) give very strong probabilities in terms of output in case of the three classes whereas weaker networks give only mild probabilities everywhere, though the separating boundary can still remain same. We can also that though correct, weaker networks are "unsure" about their decision or are "indecisive" in cases. Also, here network with higher hidden layer nodes were able to get a nice decision boundary for even low learning rate as can be seen from Figures 12, 16, 20 and 24.

2 Neural Network to predict the steering angle from the road image for a self-driving car

2.1 Network Structure

If we consider the network to be composed of $n_1, n_2, ...n_k$ nodes in each layer with the corresponding vectors to be $\vec{v}_1, \vec{v}_2, ...\vec{v}_k$ and the weights for each layer be $\vec{w}_1, \vec{w}_2, ...\vec{w}_{k-1}$, we can then say that $v_{ij} = \sigma(\vec{w}^T_{i-1,j}\vec{v}_{i-1}), i \in \{2, 3, ..k\}$. In other words:

$$\vec{v}_i = \begin{pmatrix} v_{i,1} \\ v_{i,2} \\ ... \\ v_{i,n_i} \end{pmatrix}_{n_i \times 1} , \vec{w}_{i,j} = \begin{pmatrix} w_{i,1,j} \\ w_{i,2,j} \\ ... \\ w_{i,n_i,j} \end{pmatrix}_{n_i \times 1}$$

$$\vec{w}_i = \begin{pmatrix} \vec{w}^T_{i,1} \\ \vec{w}^T_{i,2} \\ ... \\ \vec{w}^T_{i,n_{i+1}} \end{pmatrix}_{n_{i+1} \times n_i} = \begin{pmatrix} w_{i,1,1} & w_{i,2,1} & ... & w_{i,n_i,1} \\ w_{i,1,2} & w_{i,2,2} & ... & w_{i,n_i,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i,1,n_{i+1}} & w_{i,2,n_{i+1}} & ... & w_{i,n_i,n_{i+1}} \end{pmatrix}$$

where

$$\begin{aligned} \vec{v}_i &\text{ is Layer } i \\ v_{i,j} &\text{ is } j^{\text{th}} \text{ node in Layer } i \\ \vec{w}_i &\text{ is weight from Layer } i \text{ to } i+1 \\ \vec{w}_{i,j} &\text{ is weight from Layer } i \text{ to } j^{\text{th}} \text{ node in Layer } i+1 \\ w_{i,a,b} &\text{ is weight from } a^{\text{th}} \text{ node in Layer } i \text{ to } b^{\text{th}} \text{ node in Layer } i+1 \end{aligned}$$

and thus

$$v_{i,j} = \begin{cases} \sigma(\vec{w}^T_{i-1,j}\vec{v}_{i-1}) & i \neq k \\ \vec{w}^T_{k-1,j}\vec{v}_{k-1} & i = k \end{cases}$$

and therefore

$$\vec{v}_i = \begin{cases} \vec{\sigma}(\vec{w}_{i-1}\vec{v}_{i-1}) & i \neq k-1 \\ \vec{w}_{k-1}\vec{v}_{k-1} & i = k \end{cases}$$

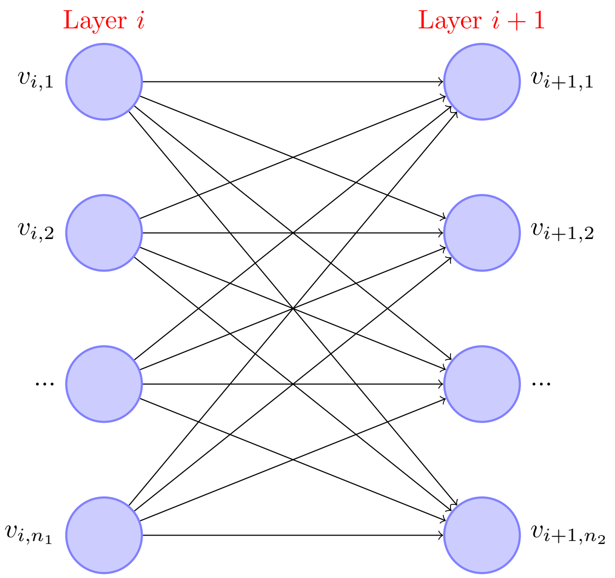


Figure 25: Portion of a neural network from layer i to layer $i+1$. The weight used here will be $\vec{w}_i$.

2.2 Gradient Descent

The error term for a minibatch of size n (i.e. from j to $j+n-1$) would be:

$$E = \sum_{i=j}^{j+n-1} \frac{1}{2}||\vec{o}^i - \vec{v}_k^i||^2$$

where superscript i denotes the i-th point. For a particular point the error would be:

$$\frac{1}{2}||\vec{o} - \vec{v}_k||^2 = \frac{1}{2}\sum_{l=1}^{n_k}(o_l - v_{k,l})^2$$

The gradient wrt weights w_{k-1,a_{k-1},a_k} can be obtained using chain rule:

$$\frac{\partial E}{\partial w_{k-1,a_{k-1},a_k}} = \underbrace{(v_{k,a_k} - o_{a_k})}_{\delta_{k,a_k}} v_{k-1,a_{k-1}}$$

Similarly for any previous layer we can obtain:

$$\frac{\partial E}{\partial w_{i,a_i,a_{i+1}}} = \underbrace{v_{i+1,a_{i+1}}(1 - v_{i+1,a_{i+1}}) \sum_{a_{i+2}=1}^{n_{i+2}} \delta_{i+2,a_{i+2}} w_{i+1,a_{i+1},a_{i+2}}}_{\delta_{i+1,a_{i+1}}} v_{i,a_i}$$

and wrt a particular point

$$\Delta w_{i,a_i,a_{i+1}} = \eta \delta_{i+1,a_{i+1}} v_{i,a_i}$$

For minibatch gradient descent we would add up these $\Delta w_{i,a_i,a_{i+1}}$'s. Also note that we also have some biases that would not propagate the gradient backwards so in case of calculation of δ_{i,a_i} we would neglect the term coming from $v_{a_{i+1}} = 1$ i.e. the bias term of the $i+1$-th layer.

Now we find a relation between $\delta_{i,j}$'s. We define $\vec{\delta_i}$ as follows:

$$\vec{\delta_i} = \begin{pmatrix} \delta_{i,1} \\ \delta_{i,2} \\ \dots \\ \delta_{i,n_i} \end{pmatrix}_{n_i \times 1}$$

And we define $\delta_{i,j}$ as follows:

$$\delta_{i,j} = \begin{cases} v_{k,j} - o_j & i = k \\ v_{i,j}(1 - v_{i,j}) \displaystyle\sum_{x=1}^{n_{i+1}} \delta_{i+1,x} w_{i,j,x} & \text{otherwise} \end{cases}$$

We can observe that:

$$\Delta \vec{w}_i = \eta \vec{\delta}_{i+1} \vec{v}_i^T$$

$$\vec{\delta}_i = \begin{cases} \vec{v}_i \circ (\vec{1} - \vec{v}_i) \circ \vec{w}_i^T \vec{\delta}_{i+1} & i \neq k \\ \vec{v}_k - \vec{o} & i = k \end{cases}$$

where $(\circ)$ is the elementwise product.

2.3 Minibatch gradient Descent

We define few more quantities (here superscript i denotes i-th point, n denotes total points in a minibatch):

$$\vec{V}_i = \begin{pmatrix} \vec{v}_i^1 & \vec{v}_i^2 & \dots & \vec{v}_i^n \end{pmatrix}$$

$$\vec{O} = \begin{pmatrix} \vec{o}^1 & \vec{o}^2 & \dots & \vec{o}^n \end{pmatrix}$$

$$\vec{\xi}_i = \begin{pmatrix} \vec{\delta}_i^1 & \vec{\delta}_i^2 & \dots & \vec{\delta}_i^n \end{pmatrix}$$

We note that:

$$\vec{V}_i = \begin{cases} \vec{\sigma}(\vec{w}_{i-1} \vec{V}_{i-1}) & i \neq k-1 \\ \vec{w}_{k-1} \vec{V}_{k-1} & i = k \end{cases}$$

$$\vec{\xi}_i = \begin{cases} \vec{V}_i \circ (\vec{1} - \vec{V}_i) \circ \vec{w}_i^T \vec{\xi}_{i+1} & i \neq k \\ \vec{V}_k - \vec{O} & i = k \end{cases}$$

$$\Delta \vec{w}_i = \eta \vec{\xi}_{i+1} \vec{V}_i^T = \eta \sum_{j=1}^{n} \vec{\delta}_{i+1}^j \vec{v}_i^{j,T}$$

2.4 Dropout

For implementation of dropout, some of the nodes in the layers V_1 to V_{k-1} are disabled with probability p (say 0.5), thus the modified layer vectors would become:

$$V_i' = V_i \circ D_i \qquad (i < k)$$

where D_i is a matrix of independent Bernoulli random variables with probability p, i.e. $(D_i)_{\alpha,\beta} \sim \text{Bernoulli}(p), 1 \leq \alpha \leq n_i, 1 \leq \beta \leq n$ (recall n is minibatch size) and of the same size as of V_i. During backpropagation, the ξ values are modified with the same D_i as:

$$\begin{aligned}
\xi_i' &= \xi_i \circ D_i \\
&= V_i' \circ (1 - V_i') \circ \vec{w}_i^T \vec{\xi}_{i+1}\,' \circ D_i \\
&= (D_i \circ V_i') \circ (1 - V_i') \circ \vec{w}_i^T \vec{\xi}_{i+1}\,' \qquad (i < k) \\
&= V_i' \circ (1 - V_i') \circ \vec{w}_i^T \vec{\xi}_{i+1}\,' \\
&= \xi_i
\end{aligned}$$

We note that for $i < k$, the ξ_i terms already include D_i's whose inclusion further doesn't make any change so we don't need to change anything in the backpropagation. We further note that:

- The bias term is not multiplied with D_i.
- At testing we multiply the weights by p and the network is used without dropout.

2.5 Observations

The following graphs were observed

2.5.1 Training for 5000 epochs at $\eta = 0.01$ without dropout

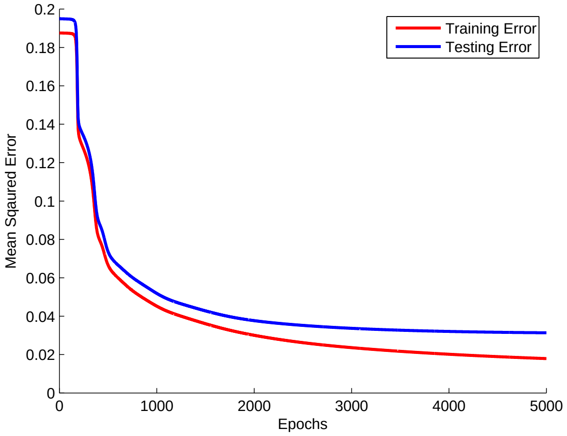


Figure 26: Variation of Training and Testing error as a function of number of epochs with parameters $\eta = 0.01$, architecture $[1024, 512, 64, 1]$, split ratio 0.8, epochs 5000, minibatch size 64, dropout rate 0.

2.5.2 Varying minibatch size

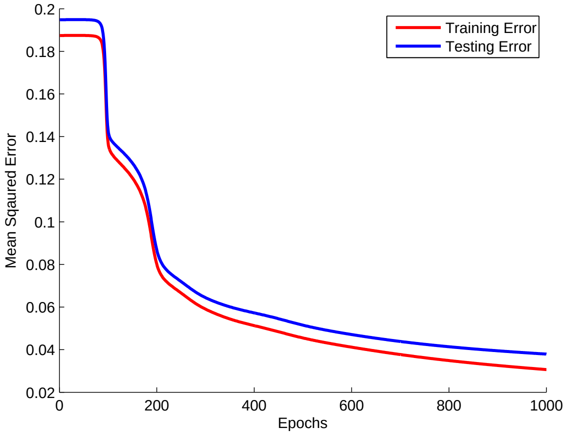


Figure 27: Variation of Training and Testing error as a function of number of epochs with parameters $\eta = 0.01$, architecture $[1024, 512, 64, 1]$, split ratio 0.8, epochs 1000, **minibatch size** 32, dropout rate 0.

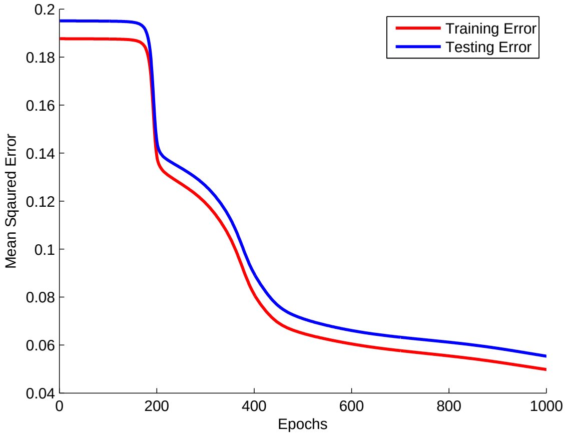


Figure 28: Variation of Training and Testing error as a function of number of epochs with parameters $\eta = 0.01$, architecture $[1024, 512, 64, 1]$, split ratio 0.8, epochs 1000, **minibatch size** 64, dropout rate 0.

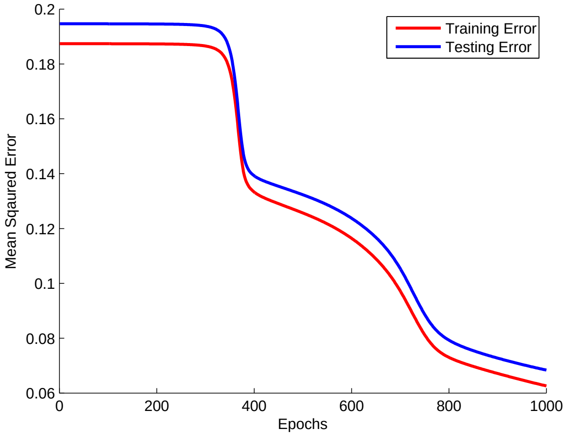


Figure 29: Variation of Training and Testing error as a function of number of epochs with parameters $\eta = 0.01$, architecture $[1024, 512, 64, 1]$, split ratio 0.8, epochs 1000, **minibatch size** 128, dropout rate 0.

2.5.3 Dropout

2.5.4 Varying Learning Rate

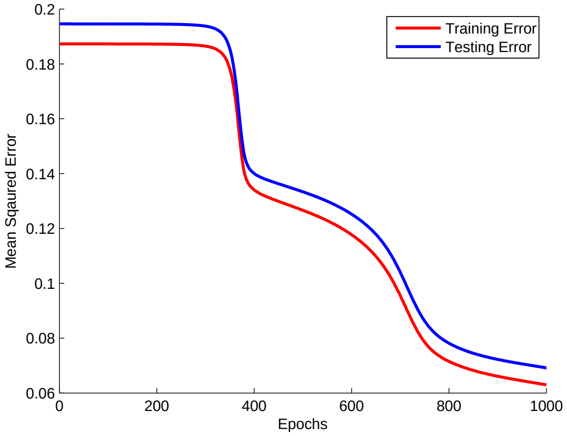


Figure 30: Variation of Training and Testing error as a function of number of epochs with parameters **learning rate** $\eta = 0.005$, architecture $[1024, 512, 64, 1]$, split ratio 0.8, epochs 1000, minibatch size 128, dropout rate 0.

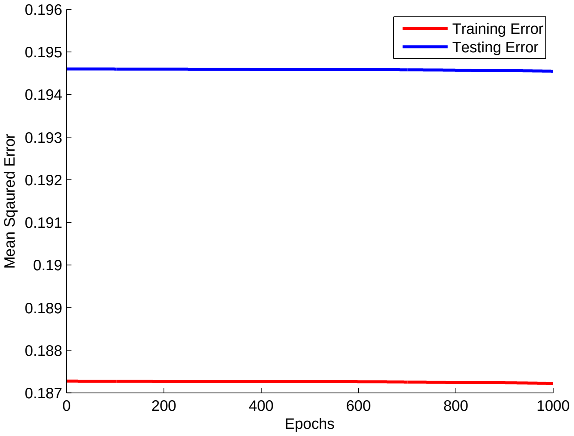


Figure 31: Variation of Training and Testing error as a function of number of epochs with parameters **learning rate** $\eta = 0.001$, architecture $[1024, 512, 64, 1]$, split ratio 0.8, epochs 1000, minibatch size 128, dropout rate 0.

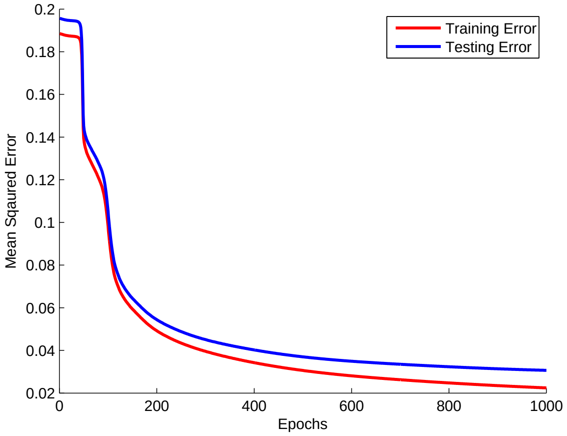


Figure 32: Variation of Training and Testing error as a function of number of epochs with parameters **learning rate** $\eta = 0.05$, architecture $[1024, 512, 64, 1]$, split ratio 0.8, epochs 1000, minibatch size 128, dropout rate 0.

2.6 Conclusions and Discussions

2.6.1 Learning Rate

2.6.2 Minibatch Size

2.6.3 Dropout

3 Improved network for steering angle prediction

3.1 Network Structure

The network structure was `[Conv-RELU-Conv-Relu-MaxPool]`×3`-Dense-Relu-Dense` where each Convolutional layer had 32 filters of size 3×3 (and a bias) and max pooling was done of size 2×2, the dense layers had 64 and 1 nodes respectively.

3.1.1 Convolution Layer

Suppose we have an image$I$ of size $H \times W$ with pixels $x_{i,j}$ which is cross-correlated with a kernel K with weights $w_{i,j}$ of size $k_1 \times k_2$ with bias b and an activation f, then the resultant image has the pixels:

$$(I * K)_{i,j} = x'_{i,j} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} x_{i+m,j+n} w_{m,n} + b$$

and after activation:

$$o'_{i,j} = f(x'_{i,j})$$

Let us find the weight update equations with t as target value and o^k as the predicted value (output at k-th layer), the given error is:

$$E = \frac{1}{2}\sum ||t - o^k||^2$$

and therefore for a resultant image of size $(H - k_1 + 1) \times (W - k_2 + 1)$ (here superscripts denote the layer index):

$$\begin{aligned}
\frac{\partial E}{\partial w^l_{m',n'}} &= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \underbrace{\frac{\partial E}{\partial x^l_{i,j}}}_{\delta^l_{i,j}} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \\
&= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial}{\partial w^l_{m',n'}} \left(\sum_{m=0}^{k_1}\sum_{n=0}^{k_2} w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l \right) \\
\frac{\partial x^{l+1}_{i,j}}{\partial w^l_{m',n'}} &= \frac{\partial}{\partial w^l_{m',n'}} \left(w^l_{0,0} o^{l-1}_{i,j} + ... + w^l_{m',n'} o^{l-1}_{i+m',j+n'} + ... + b^l \right) \\
&= o^{l-1}_{i+m',j+n'} \\
\frac{\partial E}{\partial w^l_{m',n'}} &= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta^l_{i,j} o^{l-1}_{i+m',j+n'}
\end{aligned}$$

Hence we can see that the gradients are also a cross-correlation operation as follows:

$$\left(\overrightarrow{\frac{\partial E}{\partial w^l}} \right)_{i,j} = (I' * \vec{\delta})_{i,j}$$

Now let us find a relation between $\delta^k_{i,j}$'s:

$$\begin{aligned}
\delta^l_{i,j} = \frac{\partial E}{\partial x^l_{i,j}} &= \sum_{m,n} \frac{\partial E}{\partial x^{l+1}_{m,n}} \frac{\partial x^{l+1}_{m,n}}{\partial x^l_{i,j}} \\
&= \sum_{m,n} \delta^{l+1}_{m,n} \frac{\partial x^{l+1}_{m,n}}{\partial x^l_{i,j}}
\end{aligned}$$

and hence:

$$\begin{aligned}
\frac{\partial x^{l+1}_{m,n}}{\partial x^l_{i,j}} &= \frac{\partial}{\partial x^l_{i,j}} \left(\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} o^l_{m+m',n+n'} + b^{l+1} \right) \\
&= \frac{\partial}{\partial x^l_{i,j}} \left(\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} f(x^l_{m+m',n+n'}) + b^{l+1} \right) \\
&= \frac{\partial}{\partial x^l_{i,j}} \left(w^{l+1}_{i-m,j-n} f(x^l_{i,j}) \right) \\
&= w^{l+1}_{i-m,j-n} f'(x^l_{i,j}) \\
\delta^l_{i,j} &= \frac{\partial E}{\partial x^l_{i,j}} = \sum_{m,n} \delta^{l+1}_{m,n} w^{l+1}_{i-m,j-n} f'(x^l_{i,j})
\end{aligned}$$

We can see that this is also a cross correlation operation with weights transposed:

$$\vec{\delta}^{\,l} = (\vec{\delta}^{\,l+1} * \vec{w}^{\,T,l+1}) \circ \vec{f'}$$

References

[1] Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning

[2] Backpropagation In Convolutional Neural Networks

[3] Convolutional Neural Networks: Architectures, Convolution / Pooling Layers

[4] Neural Networks Part 3: Learning and Evaluation

[5] Convnet: Implementing Convolution Layer with Numpy

[6] Convnet: Implementing Maxpool Layer with Numpy

[7] Implement MATLAB's im2col “sliding” in Python and Implement MATLAB's col2im 'sliding' in Python