NNLS Mid-term Exam Solutions

Hariharan Ravishankar

Problem1: A student claims shuffling of data will always help improve online and batch mode perceptron learning. Justify if the claim is true or false.

I will provide my arguments for batch and online mode learning separately.

Batch-Mode: Let θ is the set of parameters to learnt in the neural network and if $E_X(\theta)$ is the loss function to be optimized and X is the set of inputs. Since the loss function is non-convex in θ, there will be numerous local minima, some of them might be more preferrable over the others. If X, is presented without mini-batching or shuffling, it is quite likely that gradient descent algorithm can get stuck in one of these local minima. By shuffling the data and presenting them in mini-batches $\{X_i\}$, the gradient descent algorithm gets to explore loss surfaces corresponding to $E_{X_i}(\theta)$. Even if this is a local minimum, the learning will proceed with next mini-batch and so on. Thus, shuffling, and mini-batch presentations help gradient descent-based algorithm explore diverse trajectories of loss surfaces, thereby avoiding getting stuck in local minima of the non-convex loss function.

It should be noted that, if mini batch is full-sized that is, if $X_i = X$, shuffling will not have any impact. Since at every iteration, gradient descent is seeing all the samples, shuffling will not have any impact. In my experiments, I have observed that in full-batch experiments, the results did not change with shuffling, if I froze other levers like random weight initializations.

Online-mode: Since, online learning algorithm attends to the most recent sample, presenting data without shuffling could add bias in learning over the recent samples. For example, if data consists of M classes, if the samples are provided class-wise in sequential manner, it is possible that neural network could "forget" the earlier classes. In such scenarios, random shuffling will ensure that samples belonging to earlier classes are also replayed to the neural network periodically so that network can learn/remember these classes. In my HW1, I observed that providing 3-cluster data in sequential fashion in online-mode, did not lead to convergence. However, random shuffling of 3 classes of data, allowed the network to classify all the classes better.

The biggest advantage of online learning is adaptivity (i.e), it can respond to short-term non-stationary changes. If our data consists of such "drifts", that is the distribution changes over time, shuffling should not be pursued. For example, a hospital ward occupancy prediction problem will depend on the statistics related to recent outbreak of pandemic compared to a normal year. In such cases, we want the network to learn or be biased to the recent samples. In short, in non-stationary changing environments and problems, data shuffling may not always produce better results because of inability to model the "drift" the distributions.

Summary:

- Shuffling and mini-batch presentations will generally help in gradient-descent based weight optimization algorithms to navigate through the non-convex cost surface.
- Shuffling will not make any difference in full-batch gradient descent.
- In online learning, non-sequential presentations of different class data will help network to learn all classes sufficiently, as against possible forgetting in sequential presentation.
- However, if there is "drift" in data (i.e) non-stationary inputs, shuffling the data will not allow the network to learn the recent samples better. Hence, should be avoided.

Problem 2: S. Haykin 4.14

Temporal Processing

Output of neuron j is defined by

$y_j(n) = \phi\left(\sum_{i=1}^{m_o} w_{ji}\, u_i(n)\right)$, where ϕ is activation function.

Here $u_i(n)$ is the convolution of $x_i(n)$ and time window $\theta(n, \tau_{ji}, \sigma_{ji})$, which is defined as

$$\theta(n, \tau_{ji}, \sigma_{ji}) = \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}} \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right),\ i = 1, 2 \ldots m_o$$

$w_{ji}, t_{ji}, \sigma_{ji}$ are the parameters from to synapse i to neuron j, to be learnt using back propagation.

Let us define variables $v_j(n)$ which is the local receptive field at neuron j such that

$$y_j(n) = \phi(v_j(n))$$

$v_j(n) = \sum_{i=1}^{m_o} w_{ji}\, u_i(n)$, where m_o is the number of hidden units in previous layer

Convolution with Gaussian time-window

Noting that $u_i(n) = x_i(n) * \theta(n, \tau_{ji}, \sigma_{ji})$, where $*$ is the convolution operator.

$\boxed{u_i(n) = \sum_{n_o=0}^{n} x_i(n_0)\theta(n - n_o, \tau_{ji}, \sigma_{ji})}$, where n_o is the convolution index.

Let E be the loss function associated with the network and we will assume backpropagated error gradients are available till $y_j(n)$ (i.e) $\frac{\delta E}{\delta y_j(n)}$ is known. Based on this assumption, we can proceed backwards to derive backprop equations for $w_{ji}, t_{ji}, \sigma_{ji}$

Derivatives

Using chain rule of derivatives,

$\frac{\delta E}{\delta v_j(n)} = \frac{\delta E}{\delta y_j(n)} . \phi'(v_j(n))$, where $\phi'(x)$ is derivative of activation function

The rest of the derivatives will be expressed in terms of $\frac{\delta E}{\delta v_j(n)}$

$$\boxed{\begin{aligned} \frac{\delta E}{\delta w_{ji}} &= \frac{\delta E}{\delta v_j(n)} . \frac{\delta v_j(n)}{\delta w_{ji}} \\ \frac{\delta v_j(n)}{\delta w_{ji}} &= u_i(n) = \sum_{n_o}^{n} x_i(n_0)\theta(n - n_o, \tau_{ji}, \sigma_{ji}) \end{aligned}} \quad (1)$$

Similarly, $\frac{\delta E}{\delta \tau_{ji}} = \frac{\delta E}{\delta v_j(n)} . \frac{\delta v_j(n)}{\delta \tau_{ji}}$

$$\boxed{\frac{\delta v_j(n)}{\delta \tau_{ji}} = \sum_{n_o}^{n} w_{ji} .\ x_i(n_0) .\ \frac{\delta\theta(n - n_o, \tau_{ji}, \sigma_{ji})}{\delta \tau_{ji}}} \quad (2)$$

Proceeding similarly for σ_{ji},

$$\frac{\delta E}{\delta \sigma_{ji}} = \frac{\delta E}{\delta v_j(n)} \cdot \frac{\delta v_j(n)}{\delta \sigma_{ji}}$$

$$\frac{\delta v_j(n)}{\delta \sigma_{ji}} = \sum_{n_o}^{n} w_{ji} \,.\, x_i(n_0) \,.\, \frac{\delta \theta(n - n_o, \tau_{ji}, \sigma_{ji})}{\delta \sigma_{ji}} \qquad (3)$$

Derivatives of $\theta(n, \tau_{ji}, \sigma_{ji})$ w.r.t σ_{ji} and τ_{ji}

Given definition of $\theta(n, \tau_{ji}, \sigma_{ji})$, we proceed to find $\frac{\delta \theta(n, \tau_{ji}, \sigma_{ji})}{\delta \sigma_{ji}}$ and $\frac{\delta \theta(n, \tau_{ji}, \sigma_{ji})}{\delta \tau_{ji}}$

$$\frac{\delta \theta(n, \tau_{ji}, \sigma_{ji})}{\delta \sigma_{ji}} = \frac{\delta\left(\frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}} \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)\right)}{\delta \sigma_{ji}}$$

$$= \frac{1}{\sqrt{(2\pi)}} \cdot \left(\frac{\delta\left(\frac{\exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)}{\sigma_{ji}}\right)}{\delta \sigma_{ji}} \right), \text{applying quotient role}$$

$$= \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}^2} \cdot \left(\frac{\delta\left(\exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)\right)}{\delta \sigma_{ji}} \,.\, \sigma_{ji} \ - \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right) . \frac{\delta(\sigma_{ji})}{\delta \sigma_{ji}} \right)$$

$$= \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}^2} \cdot \left(\frac{(n - \tau_{ji})^2 \,.\, \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)}{\sigma_{ji}^2} \ - \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right) \right)$$

$$= \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}^4} \left((n - \tau_{ji})^2 - \sigma_{ji}^2\right) . \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)$$

$$\frac{\delta \theta(n, \tau_{ji}, \sigma_{ji})}{\delta \sigma_{ji}} = \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}^4} \left((n - \tau_{ji})^2 - \sigma_{ji}^2\right) . \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right) \qquad (4)$$

Similarly,

$$\frac{\delta \theta(n, \tau_{ji}, \sigma_{ji})}{\delta \tau_{ji}} = \frac{\delta\left(\frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}} \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)\right)}{\delta \tau_{ji}}$$

$$= \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}} \cdot \frac{\delta\left(\exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)\right)}{\delta \tau_{ji}}$$

$$= \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}} \cdot \left(\frac{1}{2\ \sigma_{ji}^2} * 2 * (n - \tau_{ji})\right) . \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right)$$

$$\frac{\delta \theta(n, \tau_{ji}, \sigma_{ji})}{\delta \tau_{ji}} = \frac{1}{\sqrt{(2\pi)}\ \sigma_{ji}^3} . (n - \tau_{ji}) . \exp\left(-\frac{1}{2\sigma_{ji}^2}\ (n - \tau_{ji})^2\right) \qquad (5)$$

Back-propagation equations Combining the derivatives developed earlier, we are ready to write backpropagation update equations using chain rule of derivatives.

$$w_{ji}(t+1) = w_{ji}(t) - \eta * \frac{\delta E}{\delta w_{ji}}$$
$$\sigma_{ji}(t+1) = \sigma_{ji}(t) - \eta * \frac{\delta E}{\delta \sigma_{ji}} \qquad (6)$$
$$\tau_{ji}(t+1) = \tau_{ji}(t) - \eta * \frac{\delta E}{\delta \tau_{ji}}$$

All the derivatives needed for update equations are available in earlier equations.

Demonstration on single hidden layer network

Consider 1 hidden layer network with temporal processing units.

$$y_j^1(n) = \phi\left(\sum_{i=1}^{n_1} w^1{}_{ji}\, u^1{}_i(n)\right), \text{ where } u^1{}_i(n) = x_i(n) * \theta^1{}_{ji}(n)$$

$$y_j^2(n) = \phi\left(\sum_{i=1}^{n_2} w^2{}_{ji}\, u^2{}_i(n)\right), \text{ and } u^2{}_i(n) = y^1{}_i(n) * \theta^2{}_{ji}(n)$$

Let $E = \sum \frac{1}{2} |\boldsymbol{d}(\boldsymbol{n}) - \boldsymbol{y}^2(\boldsymbol{n})|^2$ and gradient with each output neuron be $\frac{\delta E}{\delta y^2{}_j(n)}$

Let $v_j^1(n)$ and $v_j^2(n)$ be the local receptive fields of first and second layer neurons.

Let there be n_I, n_1, n_2 in input, hidden and output units respectively.

Based on the equations derived, the update equations are as follows.

$$\frac{\delta v^1{}_j(n)}{\delta w^1{}_{ji}} = u^1{}_i(n) = \sum^{n_i} x_i(n_0)\theta^1\left(n - n_o, \tau^1{}_{ji}, \sigma^1_{ji}\right)$$

$$\frac{\delta E}{\delta w^1{}_{ji}} = \frac{\delta E}{\delta v_j^1(n)} \cdot \sum^{n_i} x_i(n_0)\theta^1\left(n - n_o, \tau^1{}_{ji}, \sigma^1_{ji}\right)$$

$$\frac{\delta v^2{}_j(n)}{\delta w^2{}_{ji}} = u^2{}_i(n) = \sum^{n_1} y^1{}_i(n_0)\theta^2\left(n - n_o, \tau^2{}_{ji}, \sigma^2_{ji}\right)$$

$$\frac{\delta E}{\delta w^2{}_{ji}} = \frac{\delta E}{\delta v_j^2(n)} \cdot \sum^{n_1} y^1{}_i(n_0)\theta^2\left(n - n_o, \tau^2{}_{ji}, \sigma^2_{ji}\right)$$

$$w^1{}_{ji}(t+1) = w^1{}_{ji}(t) - \eta * \frac{\delta E}{\delta w^1{}_{ji}}$$

$$w^2{}_{ji}(t+1) = w^2{}_{ji}(t) - \eta * \frac{\delta E}{\delta w^2{}_{ji}}$$

For sigmas, $$\frac{\delta E}{\delta \sigma^1{}_{ji}} = \frac{\delta E}{\delta v^1{}_j(n)} \cdot \frac{\delta v^1{}_j(n)}{\delta \sigma^1{}_{ji}}$$

$$\frac{\delta v^1{}_j(n)}{\delta \sigma^1{}_{ji}} = \sum_{n_o}^{n} w^1{}_{ji} \,.\, x_i(n_0) \,.\, \frac{\delta \theta^1\left(n - n_o, \tau^1{}_{ji}, \sigma 1_{\ ji}\right)}{\delta \sigma^1{}_{ji}}$$

$$\frac{\delta E}{\delta \sigma^2{}_{ji}} = \frac{\delta E}{\delta v^2{}_j(n)} \cdot \frac{\delta v^2{}_j(n)}{\delta \sigma^2{}_{ji}}$$

$$\frac{\delta v^2{}_j(n)}{\delta \sigma^2{}_{ji}} = \sum_{n_o}^{n_1} w^2{}_{ji} \,.\, y^1{}_i(n_0) \,.\, \frac{\delta \theta^2\left(n - n_o, 2_{ji}, \sigma^2{}_{ji}\right)}{\delta \sigma^2{}_{ji}}$$

$$\sigma^1{}_{ji}(t+1) = \sigma^1{}_{ji}(t) - \eta * \frac{\delta E}{\delta \sigma^1{}_{ji}}$$

$$\sigma^2{}_{ji}(t+1) = \sigma^2{}_{ji}(t) - \eta * \frac{\delta E}{\delta \sigma^2{}_{ji}}$$

For taus, $$\frac{\delta E}{\delta \tau^1{}_{ji}} = \frac{\delta E}{\delta v^1{}_j(n)} \cdot \frac{\delta v^1{}_j(n)}{\delta \tau^1{}_{ji}}$$

$$\frac{\delta v^1{}_j(n)}{\delta \tau^1{}_{ji}} = \sum_{n_o}^{n_I} w^1{}_{ji} \,.\, x_i(n_0) \,.\, \frac{\delta \theta^1\left(n - n_o, \tau^1{}_{ji}, \sigma^1{}_{ji}\right)}{\delta \tau^1{}_{ji}}$$

$$\frac{\delta E}{\delta \tau^2{}_{ji}} = \frac{\delta E}{\delta v^2{}_j(n)} \cdot \frac{\delta v^2{}_j(n)}{\delta \tau^2{}_{ji}}$$

$$\frac{\delta v^2{}_j(n)}{\delta \tau^2{}_{ji}} = \sum_{n_o}^{n_1} w^2{}_{ji} \,.\, y^1{}_i(n_0) \,.\, \frac{\delta \theta^2\left(n - n_o, \tau^2{}_{ji}, \sigma^2{}_{ji}\right)}{\delta \tau^2{}_{ji}}$$

$$\tau^1{}_{ji}(t+1) = \tau^1{}_{ji}(t) - \eta * \frac{\delta E}{\delta \tau^1{}_{ji}}$$

$$\tau^2{}_{ji}(t+1) = \tau^2{}_{ji}(t) - \eta * \frac{\delta E}{\delta \tau^2{}_{ji}}$$

This completes the derivation for generic and single hidden layer network for temporal processing network.

Problem 3: Multi-class logistic regression - Iris Dataset

Update Equations summary:

Let $\{\boldsymbol{x_i}, d_i\}$ be the N sample training pairs belonging to M=3 classes in IRIS Dataset. For every class, we learn a weight vector $\boldsymbol{w_k}$ which is $d+1$ length vector accounting for bias term. The likelihood of a sample belonging to class C_k is given by softmax function.

$$p(y_k \mid \boldsymbol{x}_i) = y_{\mathrm{ik}}(\boldsymbol{x_i}) = \frac{\exp\left(w_k^T\, \tilde{\boldsymbol{x}}_i\right)}{\sum_l^M \exp\left(w_l^T\, \tilde{\boldsymbol{x}}_i\right)}$$, where $\tilde{x}_i$ is the augmented vector.

Encoding d_i as 1-of-M binary vector and denoting it by $\boldsymbol{t_i}$ with 1 in k^{th} location if $\boldsymbol{x_i}\, \epsilon\, C_k$. Let T be the NxM matrix composed of $\{t_i\}$.

- Data likelihood is given by

$$P(T \mid \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots \boldsymbol{w_M}\}) = \prod_{i=1}^{N}\prod_{k=1}^{M} (y_{\mathrm{ik}}(\boldsymbol{x_i}))^{(t_{\mathrm{ik}})}$$

- Taking Negative logarithm and denoting it by E,

$$E(\boldsymbol{w}_1, \boldsymbol{w}_2 .. \boldsymbol{w_M}) = -\sum_{i=1}^{N}\sum_{k=1}^{M} t_{\mathrm{ik}} \ln(y_{\mathrm{ik}}(\boldsymbol{x_i}))$$

- Gradient w.r.t w_j was derived as (in HW2),

$$\nabla^E{}_{\boldsymbol{w_j}} = \sum_{i=1}^{N} (y_{\mathrm{ij}} - t_{\mathrm{ij}}) * x_i$$

- Update equation for weights is given by

$$w_j = w_j + \eta * \Delta^{\mathrm{E}}_{\mathrm{w_j}} \qquad \text{- (1)}$$

The derived equations were implemented using matlab and were tested on IRIS dataset. Experiments, results summary, outcomes and observations from cross validation experiments are shared below.

Results and Discussion

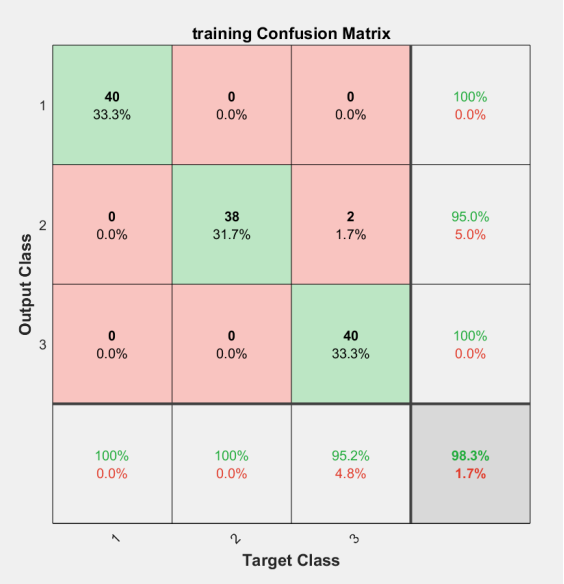


Figure 1 Training Confusion Matrix

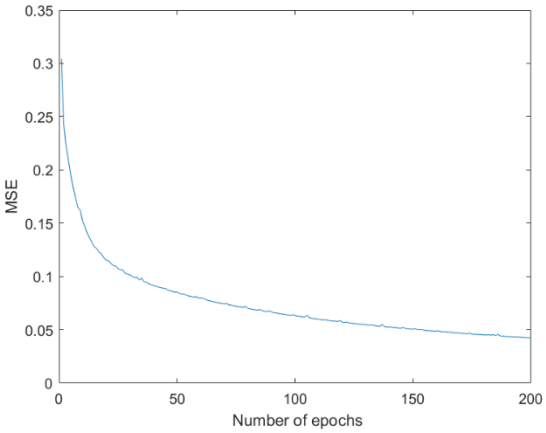


Figure 2 Error curve over epochs

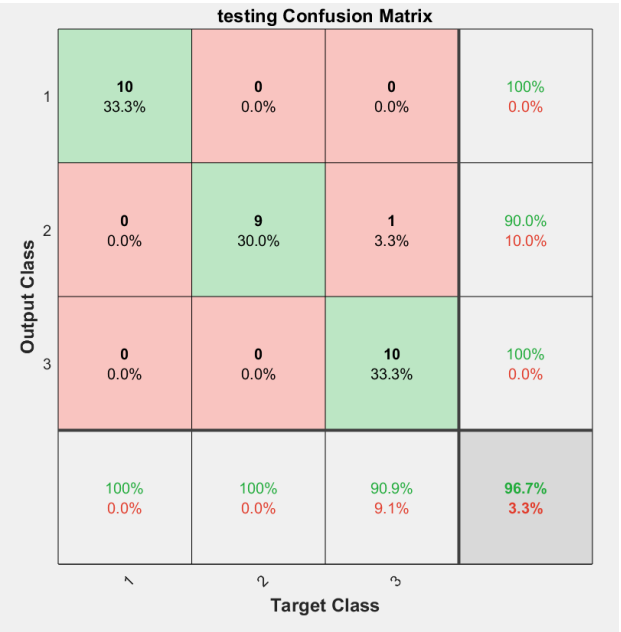


Figure 2 Testing Confusion Matrix

Observations

1. **Training accuracy of 98.3% and testing accuracy of 96.7 % was achieved with multi-class logistic regression model.**
2. It should be noted that similar performance was achieved with MLP in HW2 problem.
3. Samples from second class, Iris Versicolor were misclassified.
4. There were two misclassifications in training and one in testing, respectively.
5. For the chosen parameter of learning rate and mini-batch size, the error curve is shown in Fig 2, which depicts smooth convergence over epochs.

Cross-validation Experiments

I proceeded to evaluate the algorithm in cross-validation mode with 5-fold validation strategy.

Observations from cross-validation experiments

1. Average training and test set accuracies were **97.5% and 97.3%** respectively averaged over the 5-folds of cross-validation
2. This demonstrates that algorithm generalized well on this dataset
3. It was also observed that algorithm converged with similar results for other hyper-parameters like η. I did not see performance degradation or improvement with η varied between {0.01, 0.1, 0.5}.

Problem 4: Simon Haykin 4.17

Pattern classification of "equiprobable" overlapping two-dimensional Gaussian distributed patterns $\mathcal{C}_1$ and $\mathcal{C}_2$. The conditional probability density functions are given by

Class $\mathcal{C}_1$: $$p_{\boldsymbol{x}|\mathcal{C}_1}(\boldsymbol{x}|\mathcal{C}_1) \sim \mathcal{N}(\boldsymbol{\mu_1}, \sigma_1^2 \boldsymbol{I}), \quad \boldsymbol{\mu_1} = [0,\ 0]^T, \sigma_1^2 = 1$$

Class $\mathcal{C}_2$: $$p_{\boldsymbol{x}|\mathcal{C}_2}(\boldsymbol{x}|\mathcal{C}_2) \sim \mathcal{N}(\boldsymbol{\mu_2}, \sigma_2^2 \boldsymbol{I}), \quad \boldsymbol{\mu_2} = [0,\ 0]^T, \sigma_1^2 = 4$$

Derivation of optimal Bayesian decision boundary

Optimal decision boundary is defined by the likelihood test

$$\Lambda(\boldsymbol{x}) = \frac{p_{\boldsymbol{x}|\mathcal{C}_1}(\boldsymbol{x}|\mathcal{C}_1)}{p_{\boldsymbol{x}|\mathcal{C}_2}(\boldsymbol{x}|\mathcal{C}_2)} \qquad (1)$$

The threshold for classification is determined by ratio of priors and hence $\lambda = 1$. Equating (1) to 1, taking logarithm on both sides and utilizing conditional distributions defined earlier, we get

$$\log p_{\boldsymbol{x}|\mathcal{C}_1}(\boldsymbol{x}|\mathcal{C}_1) - \log p_{\boldsymbol{x}|\mathcal{C}_2}(\boldsymbol{x}|\mathcal{C}_2) = 0$$

$$-\frac{1}{2}(x_1^2 + x_2^2) - \log\left(\frac{1}{4}\right) + \frac{(x_1 - 2)^2 + x_2^2}{8} = 0$$

$$\left(x_1 + \frac{2}{3}\right)^2 + x_2^2 = (2.34)^2$$

Optimal decision boundary is a circle centered at $(-\frac{2}{3}, 0)$ with radius 2.34.

a) Calculating the theoretical probability of misclassification

Since the priors are equal, the theoretical probability of misclassification is given by,

$$P_e = 0.5 * \left(P_e^1 + P_e^2\right).$$

The decision boundary is a circle with centre at $\left[-\frac{2}{3}, 0\right]$ and radius 2.34,

$$\Lambda(\boldsymbol{x}) \Rightarrow \left(x_1 + \frac{2}{3}\right)^2 + (x_2)^2 - (2.34)^2 = 0$$

Regions of misclassification for Class 1 is therefore, $\Lambda(\boldsymbol{x})$ > 0 and for class 2 region of misclassification is $\Lambda(\boldsymbol{x})$ < 0.

$$P_e^1 = \int_{\Lambda(\boldsymbol{x})>0} p(\boldsymbol{x} \mid \mathrm{C1})$$

$$P_e^2 = \int_{\Lambda(\boldsymbol{x})<0} p(\boldsymbol{x} \mid \mathrm{C2})$$

Since we cannot compute these integrals analytically, I computed the misclassification errors by numerical evaluation. I considered two approaches a) Numerical integration evaluation in matlab b) Relative frequency approach (Monte Carlo estimates). Both methods gave same theoretical probability of misclassification error.

It was found that P_e^1 = 0.1050 and P_e^2 = 0.2648. Hence P_e = 0.1850

Neural Network Experiments

Gaussian Data generation:

I used matlab's mvrnd function to generate samples for two classes, with parameter "samples".

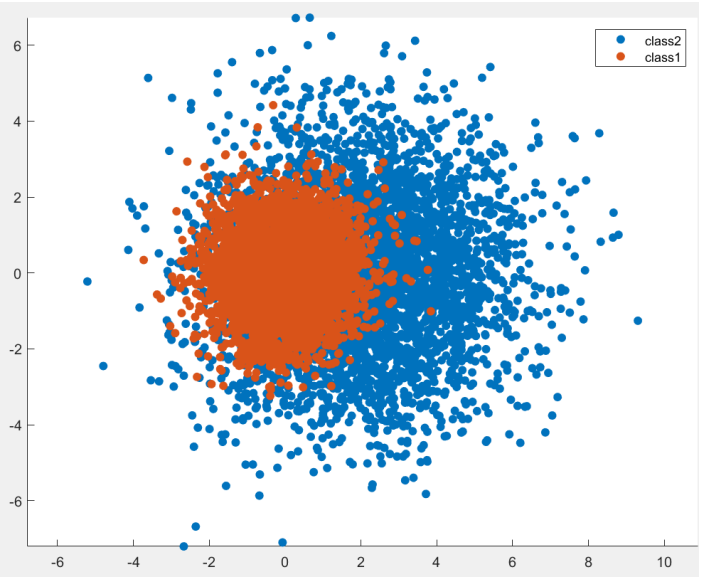


Figure 3 Visualizing generated samples

Observations

- The two classes overlap significantly.
- The visualization explains intuitively the misclassification probability of 0.1850 computed earlier.
- The visualization also gives clues on why optimal decision surface will be a circle, as derived earlier.

b) Identifying optimal number of hidden neurons

I configured the MLP network with one hidden layer, with sigmoid activations for this problem. For this experiment, I froze learning rate parameter $\eta = 0.1$ and momentum constant = 0. The first experiment involved experimenting with 2 or 4 hidden neurons.

Since, every experiment is one instantization of the learning process, I decided to average the results across 10 runs as shown above, with random initializations and random permutations of mini batches. The table below shows comparisons of results.

Training Set Size	Number of Epochs	Number of hidden neurons	Probability of misclassification
500	320	2	0.200
500	320	4	**0.192**
2000	80	2	0.194
2000	80	4	**0.191**
8000	20	2	0.198
8000	20	2	**0.190**

Table 1 Comparison of 2 hidden neurons versus 4 hidden neurons averaged over 10 runs

To evaluate probability of misclassification, I generated new samples and tested the learnt network's prediction on those samples, whose misclassification accuracies are shown in table above. It can be noted that for all the 3 experiments, 4 hidden neuron network provides slight improvement over the 2 hidden neuron configurations. Also, it should be noted that, performance of both the configurations is close to the theoretical misclassification error of 0.1850.

I also visualized the decision surface of the 2-hidden neuron network and the 4-hidden neuron network, which is shown below. It was observed that the 4-hidden neuron network provided a decision surface close to resembling the circle derived earlier.

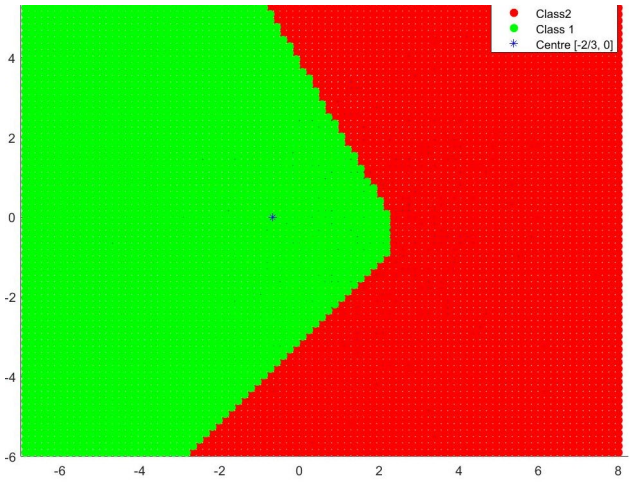
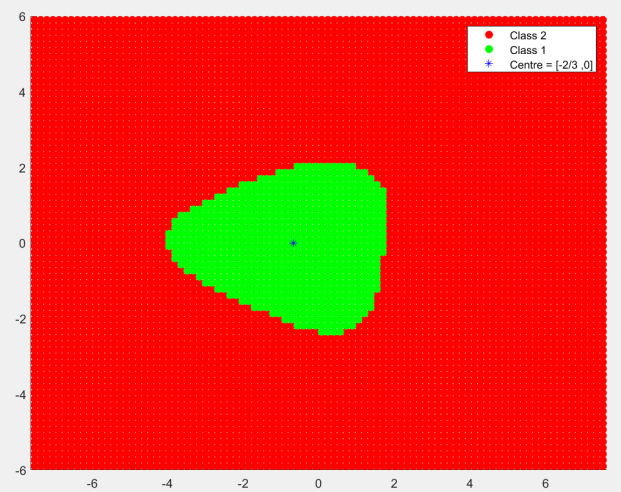


Figure 4 Decision surface of 2 hidden neurons

Observations

- Meshgrid was created and classified with learnt neural network.
- As can be seen, the decision surface is not "circle" as derived theoretically.
- However, it still is a reasonably good decision surface explaining acceptable probability of misclassification.



Observations

- 4 hidden neuron configuration produced a better decision surface than 2 hidden neuron network.
- As can be seen, decision surface is not a perfect circle, but this is a better decision surface than what was achieved with 2 hidden neuron network.

Figure 5 Decision surface for 4 hidden neuron configuration

Based on lower misclassification probabilities demonstrated in Table 1, I select 4 hidden neuron configurations as the optimal neural network architecture. Remainder of the report, I will be reporting experiments with 4 hidden neurons configuration. Also, rest experiments will be with {500 } samples and {320} epochs.

c) Identifying optimal learning rate and momentum constant

The DOE for this experiment involved varying learning rate $\eta \in \{0.01, 0.1, 0.5\}$ and momentum constant $\alpha \in \{0.0, 0.1, 0.5\}$

Learning Rate	Momentum constant	Probability of misclassification
0.01	0	0.194
0.01	0.1	0.189
0.01	0.5	0.190
0.1	0	0.192
0.1	**0.1**	**0.188**
0.1	0.5	0.191
0.5	0	0.199
0.5	0.1	0.197
0.5	0.5	0.209

Figure 6 Exploration of best learning rate and momentum constant for 4 hidden neuron networks averaged over 10 runs with random permutations and initializations

Observations

- Learning rate played a more crucial role than momentum constant.
- As can be seen in the table, higher learning rate of 0.5, lead to the 3 worst misclassification errors. This highlights problem of not capturing the local minima on the cost surface due to high learning rate.
- Best results were obtained for learning rate = 0.1 and momentum constant of 0.1.
- The second best Pe=0.189 was with learning rate =0.01 and momentum constant of 0.1.
- As can be noted in table, with slow learning rate (0.01) and zero momentum (first row), network could not produce good misclassification accuracy.
- However, with non-zero momentum of 0.1 and 0.5, network was able to learn and produce good results for both learning parameters 0.01 and 0.1.
- ***It can be concluded that higher learning rate will lead to instability in learning while lower learning rate will lead to inability to learn. However, a reasonable learning rate aided by non-zero momentum constant can produce satisfactory results, as demonstrated above.***

d) Evaluating optimal configuration

- From my experiments, the optimal configuration turned out to be 4 hidden neurons, learning rate = 0.1 and momentum constant 0.1.
- The misclassification error probability was **0.188** which is closer to theoretical value of **0.185**.
- The decision surface of this configuration is shown below.

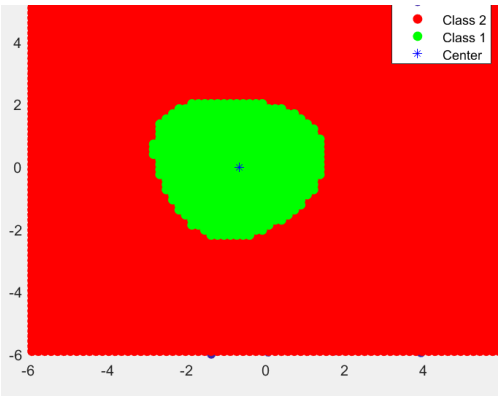


Figure 7 Decision surface for the best configuration

Observations

- The best configuration neural network produced the best decision surface of all other networks.
- As can be seen, decision surface is almost a circle, centered around [-2/3, 0] with radius = 2.34.

Conclusions: Multi-layer perceptron was able to generate non-linear decision boundaries for the two-class bivariate gaussian densities as demonstrated here. MLP came very close to matching the theoretical optimum misclassification probability, with the best configuration achieving probability of error of 0.188 against theoretical value of 0.185.