LUMS
A Not-for-Profit University

Department of Electrical Engineering
School of Science and Engineering

EE514/CS535 Machine Learning

ASSIGNMENT 3 – SOLUTIONS

Due Date: 5:00 pm, Thursday, May 03, 2023.
Format: 7 problems, for a total of 100 marks
Instructions:

- You are allowed to collaborate with your peers but copying your colleague's solution is strictly prohibited. This is not a group assignment. Each student must submit his/her own assignment.
- Solve the assignment on blank A4 sheets and staple them before submitting.
- Submit in the dropbox labeled EE-514 outside the instructor's office.
- Write your name and roll no. on the first page.
- Feel free to contact the instructor or the teaching assistants if you have any concerns.

- You represent the most competent individuals in the country, do not let plagiarism come in between your learning. In case any instance of plagiarism is detected, the disciplinary case will be dealt with according to the university's rules and regulations.

Problem 1 (10 marks)

Consider a 3-layer neural network with linear activation functions. The first layer has two input nodes, the second layer has three hidden nodes, and the third layer has one output node.

Let the activation functions in the hidden layer and the output layer be linear i.e., $f(x) = x$.

(a) (i) Write down the mathematical expression for the output of the network given input vector $\boldsymbol{x} = [x_1, x_2]$. Represent the weights and biases of the network using matrices $\boldsymbol{W}_1$, $\boldsymbol{W}_2$ and vectors $\boldsymbol{b}_1$, $\boldsymbol{b}_2$.

(ii) Calculate the output of the network for a given input vector $\boldsymbol{x} = [2, 3]$, assuming the following weights and biases:

$$\boldsymbol{W}_1 = \begin{bmatrix} 1 & 0 \\ 0 & -4 \\ 5 & 1 \end{bmatrix}, \boldsymbol{b}_1 = \begin{bmatrix} 3 \\ -5 \\ 10 \end{bmatrix}, \boldsymbol{W}_2 = \begin{bmatrix} 2 & 3 & 4 \end{bmatrix}, \boldsymbol{b}_2 = \begin{bmatrix} -1 \end{bmatrix}.$$

(b) (i) Show that the 3-layer neural network can be reduced to a single-layer linear network by combining the weight matrices and bias vectors. Derive the resulting weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{b}$ for the single-layer linear network.

Solution: Noting that the output is given by

$$\boldsymbol{y} = \boldsymbol{W}_2 \left(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1\right) + \boldsymbol{b}_2 = \boldsymbol{W}_2 \, \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{W}_2 \boldsymbol{b}_1 + \boldsymbol{b}_2,$$

we can relate output and input using a single matrix $\boldsymbol{W} = \boldsymbol{W}_2 \, \boldsymbol{W}_1 = [22 - 8]$ and a bias vector $\boldsymbol{b} = \boldsymbol{W}_2 \boldsymbol{b}_1 + \boldsymbol{b}_2 = 30$.

(ii) Calculate the output of the reduced single-layer linear network for the same input vector $\boldsymbol{x} = [2, 3]$, using the derived weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{b}$. Verify that the output is the same as the output obtained in Part a (ii). Briefly explain why this is so.

Solution: y = 50

Problem 2 (10 marks)

(a) If you apply a filter of size $k \times k$ and stride s to an input of size $n \times n$ with padding p, what will be the dimensions of the resulting feature map?

Solution: The dimension of the output feature map will be $\frac{n-k+2p}{s} + 1 \times \frac{n-k+2p}{s} + 1$.

(b) Consider a 3×3 matrix representing a patch of an image I, where each entry corresponds to the gray scale color of a pixel. Additionally, you are given a 2×2 convolutional kernel K as follows:

$$I = \begin{bmatrix} 3 & 1 & 1 \\ 3 & 0 & 2 \\ 4 & 4 & 0 \end{bmatrix}$$

and

$$K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Assuming a stride of 1 and no padding, what is the output of applying the filter to the input?

Solution: The output is

$$\begin{bmatrix} 3 & 3 \\ 7 & 0 \end{bmatrix}$$

Problem 3 (15 marks)

Suppose you have a simple neural network with one input layer, one hidden layer, and one output layer. The input layer has 2 neurons, the hidden layer has 3 neurons, and the output layer has 1 neuron. The activation function for all neurons is the sigmoid function. The network has already been initialized with the following weights and biases:

$$W_{\text{hidden}} = \begin{bmatrix} 0.3 & 0.8 \\ 0.5 & 0.1 \\ 0.9 & 0.7 \end{bmatrix}, \quad b_{\text{hidden}} = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.5 \end{bmatrix}, \quad W_{\text{output}} = \begin{bmatrix} 0.3 & 0.5 & 0.9 \end{bmatrix}, \quad b_{\text{output}} = \begin{bmatrix} 0.2 \end{bmatrix}$$

(a) Given the input vector (0.5, 0.8), perform a forward pass through the network to compute the output.

Solution: To perform a forward pass, we need to compute the activations of each layer. Using the given weights and biases, we have:

$$\sigma\left(\begin{bmatrix} 0.3 & 0.8 \\ 0.5 & 0.1 \\ 0.9 & 0.7 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.8 \end{bmatrix} + \begin{bmatrix} 0.2 \\ 0.4 \\ 0.5 \end{bmatrix}\right) = \sigma\left(\begin{bmatrix} 0.99 \\ 0.73 \\ 1.51 \end{bmatrix}\right) \approx \begin{bmatrix} 0.729 \\ 0.675 \\ 0.819 \end{bmatrix}$$

Next, we compute the activation of the output layer:

$$a_{output} = \sigma(W_{output} a_{hidden} + b_{output})$$

Using the given weights and biases, we have:

$$a_{output} = \sigma\left(\begin{bmatrix} 0.3 & 0.5 & 0.9 \end{bmatrix} \begin{bmatrix} 0.729 \\ 0.675 \\ 0.819 \end{bmatrix} + 0.2\right) \approx 0.82$$

(b) Suppose the true output for the given input is 0.6. Compute the error between the true output and the computed output.

Solution: The error between the true output and the computed output is given by:

$$E = \frac{1}{2}(y - a_{output})^2$$

where y is the true output (which is 0.6 in this case). Substituting the values, we get:

$$E = \frac{1}{2}(0.6 - 0.82)^2 \approx 0.02$$

(c) Perform a backward pass through the network using backpropagation to update the weights and biases. Use a learning rate of 0.1, to determine the updated weights and biases after one iteration of backpropagation.

Solution: To perform backpropagation, we first compute the error at the output layer using the mean squared error loss function. Let the true output be y and the computed output be $\hat{y}$, then the error is given by:

$E = \frac{1}{2}(y - \hat{y})^2$

Substituting the values, we get:

$E = \frac{1}{2}(0.6 - 0.82)^2 = 0.02$

To update the weights and biases, we need to compute the gradients of the error with respect to each weight and bias. Let σ be the sigmoid function, then the gradient of the error with respect to the output layer weights is:

$\frac{\partial E}{\partial W_{output}} = (\hat{y} - y) \cdot \sigma(z_{output}) \cdot \sigma(z_{hidden})$

where $z_{output} = W_{output} \cdot a_{hidden} + b_{output}$ is the input to the output layer, and $a_{hidden} = \sigma(z_{hidden})$ is the output of the hidden layer. Substituting the values, we get:

$\frac{\partial E}{\partial W_{output}} = (0.866 - 0.6) \cdot \sigma(1.41) \cdot \begin{bmatrix} 0.731 & 0.524 & 0.622 \end{bmatrix} = \begin{bmatrix} 0.035 & 0.025 & 0.030 \end{bmatrix}$

Similarly, the gradient of the error with respect to the output layer bias is:

$\frac{\partial E}{\partial b_{output}} = (\hat{y} - y) \cdot \sigma(z_{output})$

To update the hidden layer weights and biases, we need to propagate the error back from the output layer to the hidden layer. The gradient of the error with respect to the hidden layer weights is:

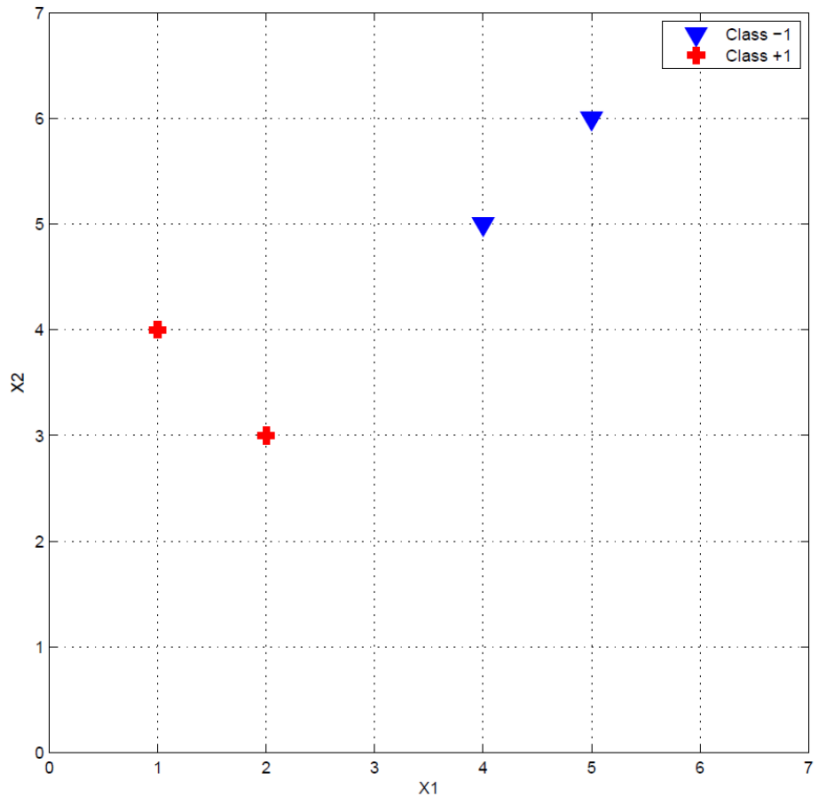
$\frac{\partial E}{\partial W_{hidden}} = \frac{\partial E}{\partial a_{hidden}} \cdot \frac{\partial a_{hidden}}{\partial z_{hidden}} \cdot a_{input}$

where a_{input} is the input to the hidden layer, and $\frac{\partial E}{\partial a_{hidden}}$ is the gradient of the error with respect to the output of the hidden layer. Using the chain rule, we can express this as:

$\frac{\partial E}{\partial a_{hidden}} = \frac{\partial E}{\partial z_{output}} \cdot \frac{\partial z_{output}}{\partial a_{hidden}}$

Problem 4 (15 marks)

For the training data plotted below, find the weight vector and bias for the decision boundary $\mathbf{w}^T\mathbf{x} - \theta = 0$ maximizing the classification margin. Also, indicate the support vectors and compute the classification margin.



Solution: The support vectors are $\mathbf{x}^b = (4, 5)$ and $\mathbf{x}^r = (2, 3)$; we have used b and r to denote blue and red support vectors respectivley. The decision boundary must be passing through (3,4) and perpendicular to the line connecting the support vectors to maximize the classification margin. This yields the decision boundary

$$x_1 + x_2 - 7 = 0.$$

If we compare this with the notation we adopt $w_1x_1 + w_2x_2 - \theta = 0$, we obtain $w_1 = w_2$

In SVM formulation, we also require the following equations to hold

$$\mathbf{w}^T\mathbf{x}^b - \theta = 1, \quad 2w_1 + 3w_2 - \theta = 1,$$
$$\mathbf{w}^T\mathbf{x}^r - \theta = -1, \quad 4w_1 + 5w_2 - \theta = -1,$$

which yields $w_1 = w_2 = \frac{-1}{2}$, and $\theta = -7/2$.

Classification margin is given by $\frac{2}{\|\mathbf{w}\|} = 2\sqrt{2}$.

Problem 5 (15 marks)

Given a dataset,

#	Data Point
x_1	$(2, 2)$
x_2	$(2, 3)$
x_3	$(3, 2)$
x_4	$(8, 7)$
x_5	$(7, 8)$
x_6	$(5, 8)$
x_7	$(4, 7)$
x_8	$(5, 3)$
x_9	$(11, 2)$
x_{10}	$(11, 3)$
x_{11}	$(10, 3)$

we wish to partition this dataset into 3 clusters using the K-Means algorithm.

(a) Show that the K-Means algorithm always converges to a local minimum in a finite number of steps. (Hint: Use the fact that the algorithm decreases the objective function in each iteration.)

Solution: Let $C_1, C_2, ..., C_k$ be the k clusters generated by the K-Means algorithm. Let $\text{SSE}(C_i)$ denote the sum of squared distances of all points in cluster C_i from the cluster centroid. That is, $\text{SSE}(C_i) = \sum_{x \in C_i} ||x - \mu_i||^2$, where μ_i is the centroid of cluster C_i.

At each iteration, the K-Means algorithm assigns each data point to the cluster with the closest centroid and then updates the centroids of each cluster. Let S_t denote the sum of squared distances of all points in their respective clusters at iteration t. That is, $\text{SSE}_t = \sum_{i=1}^{k} S(C_i)_t$. We claim that SSE_t is non-increasing and converges to a limit SSE^* as $t \to \infty$.

To prove this, note that at each iteration, the algorithm updates the centroids to minimize S_t. Thus, we have $\text{SSE}_{t+1} \leq \text{SSE}_t$ for all t. Furthermore, since S_t is non-negative, it follows that SSE_t is a decreasing sequence bounded below by 0, and therefore converges to some limit SSE^*.

Since the number of ways to partition the data into k clusters is finite, it follows that there exists some partition of the data that minimizes $\text{SSE}(C_1)+\text{SSE}(C_2)+\ldots+\text{SSE}(C_k) = \text{SSE}^*$. Therefore, the K-Means algorithm must converge to a local minimum of SSE^* in a finite number of steps.

(b) Even though it does converge every time, it is highly sensitive to the initialization of centroids. Poor initialization can result in poor clustering.
To overcome this we decide to use K-means++. K-Means++ is a variant of the K-Means algorithm that uses an improved initialization scheme. The algorithm first selects one centroid uniformly at random from the data points and then selects subsequent centroids from the remaining data points with probability proportional to the square of their distance from the nearest already chosen centroid.
Run the K-Means++ Algorithm on this dataset until convergence.

Solution: As indicated in the problem statement, the following steps in K-means++ algorithms:

1. Initialize the first cluster center by randomly selecting one data point from the data set.

2. For each remaining data point, calculate its distance to the nearest cluster center that has already been chosen. This can be done using the Euclidean distance formula.
3. Select the next cluster center by randomly choosing a data point from the remaining data points, with the probability of a point being selected proportional to the square of its distance to the nearest cluster center.
4. Repeat steps 2 and 3 until k cluster centers have been chosen.
5. Assign each data point to the nearest cluster center based on its distance.
6. Recalculate the cluster centers as the mean of the data points assigned to each cluster.
7. Repeat steps 5 and 6 until convergence (when the cluster assignments no longer change).

For the given dataset, if we choose x_1 as the first cluster center and calculate the distances from each data point to x_1, we select x_4 as the next cluster center and then select x_9 as the final cluster center. Consequently, we have the the following clusters:

Cluster 1: x_1, x_2, x_3, x_8

Cluster 2: x_4, x_5, x_6, x_7

Cluster 3: x_9, x_{10}, x_{11}

(c) Although K-Means++ requires more computations than K-Means for initialization, it often results in quicker convergence to a better clustering solution. Why is this the case?

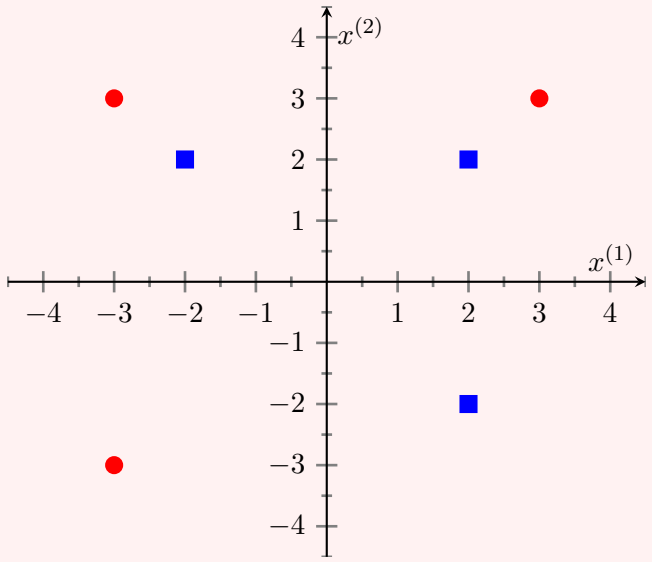
Problem 6 (15 marks)

Consider a binary classification problem with two inputs and the following labeled data-set for training.

Label y	Data Point $(x^{(1)}, x^{(2)})$
1	$(-3, -3)$
1	$(-3, 3)$
1	$(3, 3)$
-1	$(2, 2)$
-1	$(2, -2)$
-1	$(-2, 2)$

(a) Plot the points on a 2D plane. Can we use hard SVM for this problem? Provide a brief justification to support your answer.

Solution:



We cannot use hard SVM since the points are not linearly separable.

(b) Since the data is not linearly separable, we map the 2D feature space to 3D feature space using the mapping function $\phi(x)$ to make it linearly separable. Determine the mapping function that can enable us to use hard SVM in 3D feature space.

Solution: We can use a mapping function to transform the data to 3D space, where it becomes linearly separable. One such function is the polynomial kernel function given by

$$\phi(x^{(1)}, x^{(2)}) = (x^{(1)}, x^{(2)}, (x^{(1)})^2 + (x^{(2)})^2)$$

This function maps each 2D data point to a 3D space, where the third dimension is the square of the Euclidean distance of the data point from the origin. We can see that the data points in the first two dimensions are the same as the original data, and the third dimension provides additional information that makes the data linearly separable.

Using this mapping function, the transformed data becomes:

Label y	Data Point $\phi(x^{(1)}, x^{(2)})$
1	$(-3, -3, 18)$
1	$(-3, 3, 18)$
1	$(3, 3, 18)$
-1	$(2, 2, 8)$
-1	$(2, -2, 8)$
-1	$(-2, 2, 8)$

In the transformed 3D space, the data points with label 1 are clustered in a region separated from the data points with label -1, and a linear separator can easily separate them.

(c) We have a linear decision boundary (hard SVM) in 3D space to separate the transformed data in 3D (new feature space). Indicate this boundary as a (non-linear) decision boundary on the plot obtained in part (a).

(d) Instead of mapping the data into 3D space and using hard SVM to learn the decision boundary in 3D, we can use the kernel trick to learn a non-linear boundary you have plotted in part (c) in the original 2D feature space. Formulate a kernel function associated with the mapping function you used in part (b).

Solution: The mapping function we proposed earlier is: $\phi(\mathbf{x}) = \phi(x^{(1)}, x^{(2)}) = (x^{(1)}, x^{(2)}, (x^{(1)})^2 + (x^{(2)})^2)$. To construct a kernel function associated with this mapping, we can use the following:

$$K(\mathbf{x}, \mathbf{z} = \phi(\mathbf{x})^T \phi(\mathbf{z}) = x^{(1)} z^{(1)} + x^{(2)} z^{(2)} + \left((x^{(1)})^2 + (x^{(2)})^2\right)\left((z^{(1)})^2 + (z^{(2)})^2\right)$$

Problem 7 (20 marks)

You are given the following dataset of emails with 4 features, that is, the presence of the key word in the email: account, money, links, and password. The output variable is a binary label indicating whether the email is spam (Yes).

account	money	links	password	spam
No	No	Yes	No	No
No	No	No	No	No
Yes	Yes	Yes	Yes	No
Yes	No	Yes	Yes	Yes
No	Yes	No	No	Yes
No	No	No	Yes	Yes
No	No	Yes	Yes	Yes
Yes	Yes	Yes	No	Yes

(a) What is the entropy of the target value 'spam' in the data?

Solution: Since we can model the target value as a random variable 'spam' with values Yes and No. In decision trees, we estimate the probabilities of these from the data by relative frequency. This gives us

$$P(\text{spam} = \text{Yes}) = \frac{3}{8}$$
$$P(\text{spam} = \text{No}) = \frac{5}{8}$$

and entropy is given by

$$H(\text{'spam'}) = -\frac{5}{8}\log_2\frac{5}{8} \; - \; \frac{3}{8}\log_2\frac{3}{8} \approx 0.9544$$

(b) Which attribute would the decision tree algorithm that minimizes entropy choose to use for the root of the tree?

Solution: We're considering the four features: **account**, **money**, **links** and **password** as the first split in the tree. Since the entropy before the split is the same for all four, we can ignore that and only look at which value gives us the lowest average entropy post-split.

The split after password=No is uniform, so that's the highest in all 8 possible splits, but the split password=Yes is the most uneven in all possible splits. Here, we have to calculate to be sure, that is,

$$\frac{n_1}{n}H(R_1) + \frac{n_2}{n}H(R_2),$$

where R_1 and R_2 correspond to the data points we obtain after splitting (Yes for R_1) with respect to 'password' and $n_1 = n_2 = 4$, that is,

$$\frac{4}{8}\left(\frac{3}{4}\log_2\frac{3}{4} \; - \; \frac{1}{4}\log_2\frac{1}{4}\right) + \frac{4}{8}\left(\frac{2}{4}\log_2\frac{2}{4} \; - \; \frac{2}{4}\log_2\frac{2}{4}\right) \approx 0.9056$$

You can check that for all other features, we get

$$\frac{n_1}{n}H(R_1) + \frac{n_2}{n}H(R_2) \approx 0.9512,$$

and we therefore chose password as a feature for first split.

(c) Determine the information gain due to the split you chose in the previous question?

Solution: Information gain is given by

$$H(\text{'spam'}) - \left(\frac{n_1}{n}H(R_1) + \frac{n_2}{n}H(R_2)\right) \approx 0.0487$$

(d) Draw the full decision tree that would be learned for this data.

Solution: After splitting with respect to 'password', we obtain the following tables.

'password' = Yes

account	money	links	password	spam
Yes	Yes	Yes	Yes	No
Yes	No	Yes	Yes	Yes
No	No	No	Yes	Yes
No	No	Yes	Yes	Yes

'password' = No

account	money	links	password	spam
No	No	Yes	No	No
No	No	No	No	No
No	Yes	No	No	Yes
Yes	Yes	Yes	No	Yes

For splitting at depth level 1, we can take a shortcut. In both tables, the values of 'money' correspond exactly to one of the classes. After splitting on 'money', in all cases the resulting subset of the data contains only one class. This means that the distribution on the classes is 0/1 and the entropy is 0. This isn't true for any of the other features, so 'money' must be the best feature for both branches. Consequently, we will have a tree with four (completely pure) leaves.

— End of Assignment —