1. (18 points) **Answer the following questions regarding Fisher's Linear Discriminant Analysis (FLDA) and Principal Component Analysis (PCA).**

   (a) (3 points) **Consider two classes represented by two Gaussian distributions: $G_1$, with mean $\mu_1$ and variance $\sigma_1^2$; and $G_2$, with mean $\mu_2$ and variance $\sigma_2^2$. Using equations, define the within-class and between-class separation of $G_1$ and $G_2$.**

   - Within-Class Separation: $S_W = \sigma_1^2 + \sigma_2^2$
   - Between-Class Separation: $S_B = (\mu_1 - \mu_2)^2$

   (b) (3 points) **Provide an equation for the objective function used by FLDA on $G_1$ and $G_2$. What is the objective function trying to optimize?**

   The objective function for FLDA is defined as $J = \frac{(\mu_1-\mu_2)^2}{\sigma_1^2+\sigma_2^2}$. The objective function is trying to maximize the separation of the means, and minimize the overlap between the classes.

   (c) (5 points) **What are the differences between FLDA and PCA? List at least 3 challenges of PCA and at least 3 challenges of FLDA. Justify your answers.**

   PCA is an unsupervised algorithm for finding a basis (set of vectors) to (1) uncorrelate the data, by preserving the dimensionality, and (2) dimensionality reduction. As an unsupervised algorithm, PCA does not use class labels to find such basis but rather it finds the data-centric vectors that explain the most variance in the data. (A similar derivation of PCA also shows that the PCA basis can be found by minimizing the projection error.) If the data variance direction of projection is not the one that maximizes class separability than PCA will not be able to preserve class separability. An alternative dimensionality reduction technique is FLDA, a supervised algorithm to find a basis whose projection maximize class separability. FLDA assumes class Gaussianity and can only project up to $C - 1$ dimensions, where $C$ is the number of classes in the data set (assuming $C << N$, where $N$ is the number of samples).

   The 3 challenges of PCA include:

   - Linear dimensionality reduction. It does not preserve the information of the embedded manifold.
   - Unsupervised, which means that PCA is not necessarily helpful to a subsequent classification task.
   - PCA assumes data is Gaussian-distributed.

   The 3 challenges of FLDA include:

   - Linear dimensionality reduction. It does not preserve the information of the embedded manifold.

- FLDA will fail if discriminatory information is not in the mean but in the variance of the data.
- FLDA assumes data is Gaussian-distributed.

(d) (7 points) **Suppose you want to project a D-dimensional data space to a 1-dimensional space. Show that PCA's direction of projection corresponds to the eigenvector (associated with largest eigenvalue) of the covariance matrix of the scaled feature matrix X. Show all your work.**

The PCA linear transformation $A$ that transforms $X$ to a lower-dimensional space $Y$ is defined as: $Y = AX$, where $A$ is a $M \times D$ matrix, $X$ is a $D \times N$ data matrix and therefore $Y$ is also a $M \times N$ transformed data matrix ($M \leq D$).

The variance of the transformed data $Y$ is given by:

$$\begin{aligned} R_y &= E[YY^T] \\ &= E[AX(AX)^T] \\ &= E[AXX^TA^T] \\ &= AE[XX^T]A^T \\ &= AR_xA^T \end{aligned}$$

If we write $A$ in terms of its vector elements, $a_i$, of size $D \times 1$:

$$A = \begin{bmatrix} \overrightarrow{a_1}^T \\ \overrightarrow{a_2}^T \\ \vdots \\ \overrightarrow{a_M}^T \end{bmatrix}$$

Then,

$$\begin{aligned} R_y &= \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_M^T \end{bmatrix} R_X \begin{bmatrix} a_1 & a_2 & \dots & a_M \end{bmatrix} \\ &= \begin{bmatrix} a_1^TR_Xa_1 & a_1^TR_Xa_2 & \dots & a_1^TR_Xa_M \\ a_2^TR_Xa_1 & a_2^TR_Xa_2 & \dots & a_2^TR_Xa_M \\ \vdots & \vdots & \ddots & \vdots \\ a_M^TR_Xa_1 & a_M^TR_Xa_2 & \dots & a_M^TR_Xa_M \end{bmatrix} \end{aligned}$$

Consider the case where we are trying to project the data $X$ into a 1-dimensional space, so we are trying to find the direction $a_1$ where maximal data variance is preserved:

$$\arg_{a_1} \max a_1^T R_X a_1$$

We need to constraint the vector to have norm 1 because we are only interested in the direction of the vector not its magnitude,

$$\|a_1\|_2^2 = 1 \iff a_1^T a_1 = 1 \iff a_1^T a_1 - 1 = 0$$

Then, using Lagrange optimization:

$$\mathcal{L}(a_1, \lambda_1) = a_1^T R_X a_1 - \lambda_1 (a_1^T a_1 - 1)$$

Solving for $a_1$:

$$\frac{\partial \mathcal{L}}{\partial a_1} = 0 \iff 2R_X a_1 - 2\lambda_1 a_1 = 0 \iff R_X a_1 = \lambda_1 a_1$$

We find that $a_1$ is the engenvector of $R_X$ associated with the largest eigenvalue $\lambda_1$.

2. (7 points) **Write down the pseudo-code for training the Perceptron algorithm.**

   The pseudo-code of the Perceptron algorithm is as follows:

   1. Initialize the weight vector, $\mathbf{w}$, and the bias term, $b$.
   2. For every point in the training data, compute the output prediction.

      $$y_n = \phi(\mathbf{w}^T \mathbf{x_n} + b) \text{ where } \phi(x) = \begin{cases} 1, & x > 0 \\ -1, & x \leq 0 \end{cases}$$

   3. If the target $t_n = y_n$, then the point $x_n$ is correctly classified. No update should be made to the weights and bias term.
   4. If the target $t_n \neq y_n$, then the point $x_n$ is misclassified. We should
      1. Update the weights vector: $\mathbf{w} = \mathbf{w} + \eta y_n \mathbf{x_n}$
      2. Update the bias term: $b = b + \eta y_n$
   5. Continue iterating through the training data until no misclassified points are observed. The algorithm will only converge if the classes are linearly separable.

3. (8 points) **Answer the following questions regarding the soft-margin Support Vector Machine (SVM) classifier.**

(a) (4 points) **Define the slack variable, $\xi_n$, in the soft-margin SVM. What is its role in the final solution?**

A slack variable is defined as $\xi_n = 0$ for data points that are on or inside the correct margin boundary and $\xi_n = |t_n - y(x_n)|$ for other points. Thus a data point that is on the decision boundary $y(x_n) = 0$ will have $\xi_n = 1$, and points with $\xi_n > 1$ will be misclassified. The exact classification constraints are then replaced with

$$t_n y(x_n) \geq 1 - \xi_n, n = 1, \ldots, N$$

in which the slack variables are constrained to satisfy $\xi_n \geq 0$.

- Data points for which $\xi_n = 0$ are correctly classified and are either on the margin or on the correct side of the margin.
- Points for which $0 < \xi_n \leq 1$ lie inside the margin, but on the correct side of the decision boundary.
- And those data points for which $\xi_n > 1$ lie on the wrong side of the decision boundary and are misclassified.

(b) (4 points) **Suppose that you only want to penalize samples that are misclassified, propose a new slack variable and objective function to optimize this SVM.**

All correctly classified samples on or outside the margin will satisfy $t_n y(x_n) \geq 1$ (this includes the support vectors).

All correctly classified samples that lay inside the margin will satisfy $t_n y(x_n) \geq 0$ (these includes all samples on the discriminant function, $y(x_n) = 0$).

Thus, all samples that are misclassified, will have $t_n y(x_n) < 0$.

Let $\xi_n = 0$ for all correctly classified samples $n$ such that $t_n y(x_n) > 0$, and $\xi_n = |t_n - y(x_n)|$ for all other samples. The remaining optimization is unchanged. In this scenario, samples laying on the margin will not be penalized only those that are misclassified.

4. (6 points) **Suppose you have an MLP composed of one input layer with 10 neurons, followed by one hidden layer with 50 neurons, and finally one output layer with 3 neurons. All artificial neurons use the ReLU activation function, $\phi(x)$.**

(a) (1 point) **What is the shape of the input matrix X?**

The shape of the input matrix $\mathbf{X}$ is $m \times 10$, where $m$ represents the training batch size.

(b) (1 point) **What are the shapes of the hidden layer's weight vector $W_h$ and its bias vector $b_h$?**

The shape of the hidden layer's weight vector $W_h$ is $10 \times 50$, and the length of its bias vector $b_h$ is 50.

(c) (1 point) **What are the shapes of the output layer's weight vector $W_o$ and its bias vector $b_o$?**

The shape of the output layer's weight vector $W_o$ is $50 \times 3$, and the length of its bias vector $b_o$ is 3.

(d) (1 point) **What is the shape of the network's output matrix Y?**

The shape of the networks output matrix Y is $m \times 3$.

(e) (2 points) **Write the equation that computes the network's output matrix Y as a function of X, $W_h$, $b_h$, $W_o$, and $b_o$.**

$\mathbf{Y} = \phi(\phi(\mathbf{X}W_h + b_h)W_o + b_o)$, where $\phi(x)$ is the ReLU activation function. Recall that the ReLU function just sets every negative number in the matrix to zero. Also note that when you are adding a bias vector to a matrix, it is added to every single row in the matrix, which is called broadcasting.

5. (9 points) **Answer the following questions regarding an ANN architecture and justify your answers:**

(a) (3 points) **How many neurons do you need in the output layer if you want to classify email into spam or ham? What activation function should you use in the output layer?**

To classify email into spam or ham, you just need one neuron in the output layer of a neural network – for example, indicating the probability that the email is spam. You would typically use the logistic activation function in the output layer when estimating a probability.

(b) (3 points) **If instead you want to tackle MNIST, how many neurons do you need in the output layer, and which activation function should you use?**

If instead you want to tackle MNIST, you need 10 neurons in the output layer, and you must replace the logistic function with the softmax activation function, which can handle multiple classes, outputting one probability per class.

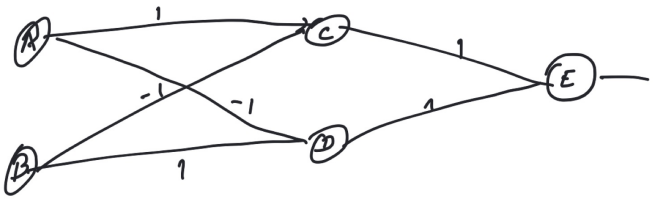
(c) (3 points) **What about for getting your network to predict housing prices? How many neurons do you need in the output layer, and which activation function should you use?**

If you want your neural network to predict housing prices, then you need one output neuron, using no activation function at all in the output layer.

6. (7 points) **Draw an Artificial Neural Network (ANN) that computes $A \oplus B$ (where $\oplus$ represents the XOR operation). Let all the bias terms be 0 and use the threshold activation function $\phi(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$. *Hint:* $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$.**

$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$

$\phi(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$




| A | B | t | C | D | E |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |

7. (15 points) **Answer the following questions regarding training ANNs. Justify your answers.**

   (a) (2 points) **What is momentum optimization? Why is it useful? How is it integrated in backpropagation?**

   Momentum assists with speed of learning. In a neural network, the learning rate parameter needs to be crucially defined to ensure stability and convergence. Too large of learning rate will cause instability and sometimes diverge. Too small learning rate will cause stability but it will take too long to converge. A momentum term can be added to delta correction rule of each weight connection without the need of increasing the learning rate.

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \text{ where}$$
$$\Delta w^{(t)} = -\eta \nabla J(w^{(t)}) + \alpha \Delta w^{(t-1)}$$

   where $\alpha$ is typically set to 0.9. At the current location for the coefficients $w^{(t)}$, we first compute the gradient and then add a scaled version of the previous delta correction rule (which will increase speed if they have the same sign, or *cancel* with current gradient if they have a different sign, i.e., the gradients are pointing in different directions).

   In the Nesterov's momentum formula, the delta correction rule is defined as:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \text{ where}$$
$$\Delta w^{(t)} = -\eta \nabla J(m^{(t)}) \text{ and}$$
$$m^{(t)} = w^{(t)} + \alpha \Delta w^{(t-1)}$$

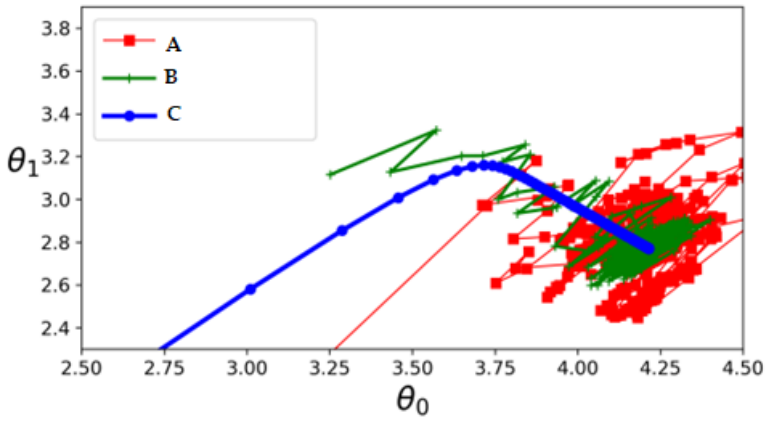
where $\alpha$ is typically set to 0.9. With the Nesterov's formula, we first add the scaled version of the previous correction rule, and then at that location ($m^{(t)} = w^{(t)} + \alpha \Delta w^{(t-1)}$), we compute the gradient.

(b) (2 points) **What strategies can you use to avoid overfitting when training ANNs?**

As usual, applying the Occam's Razor principle of selecting the simplest model should be applied. Moreover, other deep learning strategies to combat overfitting include:

- L1, L2 and Elastic Net regularization
- Dropout
- Early stopping criteria

(c) (3 points) **In the picture below, which curve (A, B or C) corresponds to mini-batch, online or batch learning?**




- A: Online learning. We see that the updates are too erratic. In online learning, we use a single sample at a time to make updates to weights. This update is not necessarily helpful for the average samples, thus the erratic behavior.
- B: Mini-batch learning. Intermediate behavior between batch and online learning. Faster than batch learning but less erratic and smoother convergence than online learning.
- C: Batch Learning. The converge is much smoother but it may take much longer than online learning.

(d) (2 points) **In a Convolutional Neural Network (CNN), why would you want to add a max pooling layer rather than a convolutional layer with the same stride? Justify your answer.**

Convolutional layers convolve the image with a small filter locally, thus capturing local features. If we use a larger stride, we are innately skipping some parts of the image, which could lead to lose features such as edges.

Instead, it is preferred to use a pooling layer as we are reducing the size of the feature map, which decreases the dimensionality of the data quite a lot, but we are not skipping information in strides. We want to stack several convolutional layers on top of each other to build up a really large/solid set of features, and then using a pooling layer, we reduce the dimensionality while preserving the spatial information that would have been lost if we used a large stride (again using strides can skip over features such as edges, etc.).

Moreover, a pooling layer has no parameters at all, whereas a convolutional layer has quite a few.

(e) (2 points) **What strategies can you use to mitigate the vanishing/exploding gradient effects when training deep ANNs?**

We can use:

- Weight initialization such as Glorot and He initialization
- Using non-saturating activation functions such as ReLU or its variants
- Batch normalization
- Gradient clipping

(f) (2 points) **In a CNN, which filter size should you use if you want to capture color depth information only?**

We should use a $1 \times 1$ filter which will not capture spatial information, but only capture depth information.

(g) (2 points) **In practice, how would you determine whether to gather more data to train your classifier?**

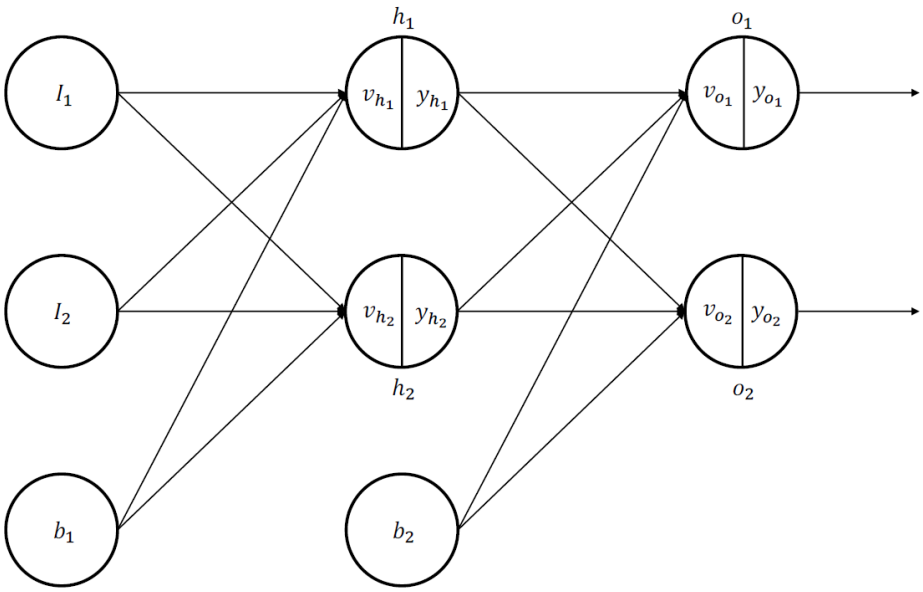
1. Determine whether the performance on the training set is acceptable. If performance on the training set is poor, the learning algorithm is not using the training data that is already available, so there is no reason to gather more data.
   - Instead, try adding complexity to the model by adding more layers or adding more hidden units to each layer.
   - Also, try improving the optimization algorithm, for example by tuning the learning rate.
   - If more complex models and carefully tuned optimization algorithms do not work well, then the problem might be the *quality* of the training data. The data may be too noisy or may not include the right inputs needed to predict the desired outputs. This suggests starting over, collecting cleaner data, or collecting a richer set of features.
2. If the performance on the training set is acceptable, then measure the performance on a test set. If the performance on the test set is also acceptable, then there is nothing left to be done. If test set performance is much worse than

training set performance, then gathering more data is one of the most effective solutions. In some applications, gathering more data is simply infeasible or impossible.

- A simple alternative to gathering more data is to reduce the size of the model or improve regularization, by adjusting hyperparameters such as weight decay coefficients,or by adding regularization strategies such as dropout.
- If you find that the gap between train and test performance is still unacceptable even after tuning the regularization hyperparameters, then gathering more data is advisable.

3. When deciding whether to gather more data, it is also necessary to decide how much to gather. It is helpful to plot curves showing the relationship between training set size and generalization error.
   - You can experiment with training set sizes on a logarithmic scale, for example, doubling the number of examples between consecutive experiments.

8. (20 points) **Consider the following neural network:**




**with the initial weights and biases listed in the table below**

**with all activation functions equal to the sigmoid function,** $\phi(x) = \frac{1}{1+e^{-x}}$**, and its derivative is,** $\phi'(x) = \phi(x)(1 - \phi(x))$**.**

**Consider the data point** $x = [0.05, 0.10]^T$ **with desired output vector** $d = [0.01, 0.99]^T$**. The objective function to be used to train this network is the squared error loss:**

$$J = \frac{1}{2}\sum_{i=1}^{N}(t_i - y_i)^2$$

| weights/bias | connection | values |
|---|---|---|
| $w_1$ | $I_1 \rightarrow h_1$ | 0.15 |
| $w_2$ | $I_2 \rightarrow h_1$ | 0.20 |
| $w_3$ | $I_1 \rightarrow h_2$ | 0.25 |
| $w_4$ | $I_2 \rightarrow h_2$ | 0.30 |
| $w_5$ | $h_1 \rightarrow o_1$ | 0.40 |
| $w_6$ | $h_2 \rightarrow o_1$ | 0.45 |
| $w_7$ | $h_1 \rightarrow o_2$ | 0.50 |
| $w_8$ | $h_2 \rightarrow o_2$ | 0.60 |
| $b_1$ | | 0.35 |
| $b_2$ | | 0.60 |

**where $t_i$ is the desired output value and $y_i$ is the network output value. Answer the following questions:**

(a) (5 points) **Apply one forward pass using the point $x = [0.05, 0.10]^T$. What is the estimated output using this network?**

For the first neuron in the hidden layer:

$$v_{h_1} = w_1 \times I_1 + w_2 \times I_2 + b_1 = 0.3775$$
$$y_{h_1} = \phi(v_{h_1}) = 0.5932$$

For the second neuron in the hidden layer:

$$v_{h_2} = w_3 \times I_1 + w_4 \times I_2 + b_1 = 0.3925$$
$$y_{h_2} = \phi(v_{h_2}) = 0.5968$$

For the first neuron in the output layer:

$$v_{o_1} = w_5 \times y_{h_1} + w_6 \times y_{h_2} + b_2 = 1.1058$$
$$y_{o_1} = \phi(v_{o_1}) = 0.7514$$

For the second neuron in the output layer:

$$v_{o_2} = w_7 \times y_{h_1} + w_8 \times y_{h_2} + b_2 = 1.2248$$
$$y_{o_2} = \phi(v_{o_2}) = 0.7729$$

Computing the error:

$$J = \frac{1}{2}\left(E_{o_1}^2 + E_{o_2}^2\right) = \frac{1}{2} * [(0.01 - 0.7514)^2 + (0.99 - 0.7729)^2] = 0.2984$$

(b) (7 points) **Using the backpropagation algorithm, find the updated value for weight $w_5$ with a learning rate of $\eta = 0.1$.**

For $w_5$, need to know how much a change in $w_5$ would affect the total error, thus we need to compute $\frac{\partial J}{\partial w_5}$.

$$\frac{\partial J}{\partial w_5} = \frac{\partial J}{\partial E_{o_1}} \times \frac{\partial E_{o_1}}{\partial y_{o_1}} \times \frac{\partial y_{o_1}}{\partial v_{o_1}} \times \frac{\partial v_{o_1}}{\partial w_5}$$

where

$$\begin{aligned}
\frac{\partial J}{\partial E_{o_1}} &= E_{o_1} = 0.01 - 0.7514 = -0.7414 \\
\frac{\partial E_{o_1}}{\partial y_{o_1}} &= -1 \\
\frac{\partial y_{o_1}}{\partial v_{o_1}} &= \phi'(v_{o_1}) = y_{o_1} \times (1 - y_{o_1}) = 0.1868 \\
\frac{\partial v_{o_1}}{\partial w_5} &= y_{h_1} = 0.5932
\end{aligned}$$

Putting it all together, we find that $\frac{\partial J}{\partial w_5} = -0.7414 \times (-1) \times 0.1868 \times 0.5932 \approx 0.08215$.

Since the learning rate is given, $\eta = 0.1$, we find the updated $w_5$:

$$w_5 = w_5 - \eta \times \frac{\partial J}{\partial w_5} = 0.40 - 0.1 \times 0.08215 = 0.3918$$

(c) (8 points) **Using the backpropagation algorithm, find the updated value for weight $w_1$ with a learning rate of $\eta = 0.1$.**

Similar to previous step, we need to compute $\frac{\partial J}{\partial w_1}$ in order to find how much a change in $w_1$ would affect the total error.

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial y_{h_1}} \times \frac{\partial y_{h_1}}{\partial v_{h_1}} \times \frac{\partial v_{h_1}}{\partial w_1}$$

where

$$
\begin{aligned}
\frac{\partial J}{\partial y_{h_1}} &= \frac{\partial E_{o_1}}{\partial y_{h_1}} + \frac{\partial E_{o_2}}{\partial y_{h_1}} \\
\frac{\partial E_{o_1}}{\partial y_{h_1}} &= \frac{\partial E_{o_1}}{\partial v_{o_1}} \times \frac{\partial v_{o_1}}{\partial y_{h_1}} \\
&= \frac{\partial E_{o_1}}{\partial y_{o_1}} \times \frac{\partial y_{o_1}}{\partial v_{o_1}} \times \frac{\partial v_{o_1}}{\partial y_{h_1}} \\
&= 0.7414 \times 0.1868 \times 0.40 = 0.0553974 \\
\frac{\partial E_{o_2}}{\partial y_{h_1}} &= \frac{\partial E_{o_2}}{\partial y_{o_2}} \times \frac{\partial y_{o_2}}{\partial v_{o_2}} \times \frac{\partial v_{o_2}}{\partial y_{h_1}} \\
&= -0.2171 \times 0.1755 \times 0.45 = -0.017145
\end{aligned}
$$

Finally, we find that $\frac{\partial J}{\partial y_{h_1}} = \frac{\partial E_{o_1}}{\partial y_{h_1}} + \frac{\partial E_{o_2}}{\partial y_{h_1}} = 0.0553974 - 0.017145 = 0.0382524$

Next, compute $\frac{\partial y_{h_1}}{\partial v_{h_1}}$:

$$\frac{\partial y_{h_1}}{\partial v_{h_1}} = y_{h_1} \times (1 - y_{h_1}) = 0.5932 \times (1 - 0.5932) = 0.2413$$

Finally, compute $\frac{\partial v_{h_1}}{\partial w_1}$:

$$\frac{\partial v_{h_1}}{\partial w_1} = I_1 = 0.05$$

Then, we can compute $\frac{\partial J}{\partial w_1}$:

$$\frac{\partial J}{\partial w_1} = 0.0382524 \times 0.2413 \times 0.05 = 0.000461515$$

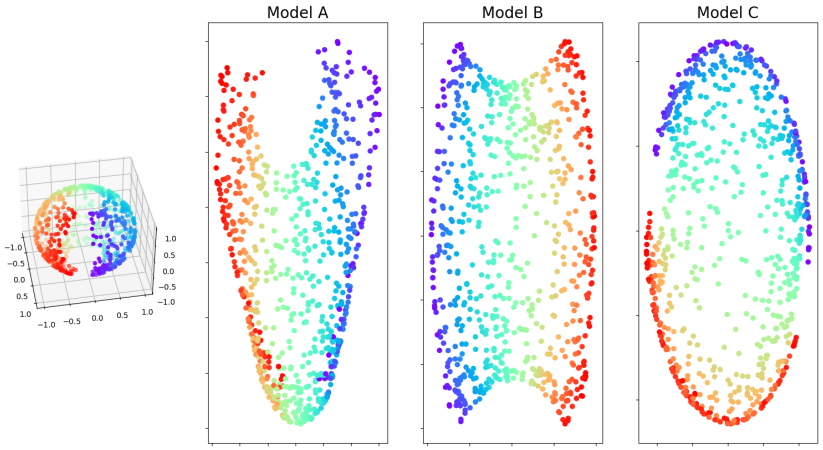
Therefore, the updated $w_1$ would be:

$$w_1 = w_1 - \eta \times \frac{\partial J}{\partial w_1} = 0.15 - 0.1 \times 0.000461515 = 0.1499538$$

9. (10 points) **Consider the three-dimensional "open sphere" dataset displayed in the leftmost plot below, as well as the performance of three different dimensionality reduction models (A, B and C):**

   **Answer the following questions:**

   (a) (5 points) **Which model performance (A, B or C) corresponds to Multi-Dimensional Scaling (MDS) with Euclidean distance, Locally Linear Embedding (LLE) and Isometric Mapping (ISOMAP)? Justify your answer.**




The classic MDS (uses Euclidean distance) will approximate the manifold close to a linear manifold (if the manifold is sufficiently represented, in terms of samples, then classic MDS will approximate PCA). For this reason, we expect MDS to simply take a projection of the data, in one of the hyperplane subspaces embedded in the 3-D space. This representation captures the global structure of the data, as MDS attempts to maximize the pairwise distance matrix in the lower dimensionalspace. So performance in C corresponds to the classic MDS.

ISOMAP is a manifold learning technique that is able to capture preserve the global structure by computing the geodesic distance between pairs of points. ISOMAP is well equipped to describe the intrinsic manifold shape provided that the manifold is fully describe without any under sampled region (i.e. absence of holes). For this reason, ISOMAP produced performance in B.

LLE is a manifold learning technique that is able to capture the local structure by reconstructing a high-dimensional manifold locally with *patches*. Each data sample is reconstructed as a linear combination of its neighbors, and this relationship is preserved in the lower-dimensional space. LLE is best at keeping the local structure but some of the global representation will be lost, which is better described in performance A.

(b) (5 points) **Between MDS, LLE and ISOMAP, which algorithm is better equipped at preserving local structure and global structure of the manifold? Justify your answer.**

See above.