1. (20 points) **Consider the dataset $\mathbf{X} = \{x_i\}_{i=1}^N$, where $x_i \geq 0, \forall i$. Suppose your goal is to perform density estimation using a Mixture Model, in particular, a Rayleigh Mixture Model. Its data likelihood can be written as:**

$$f(x) = \sum_{k=1}^{K} \pi_k g_k\left(x|\sigma_k\right)$$

where

$$\sum_{k=1}^{K} \pi_k = 1$$

and

$$g_k\left(x|\sigma_k\right) = \frac{x}{\sigma_k^2} e^{-x^2/(2\sigma_k^2)}$$

with $\sigma_k > 0$ and $0 \leq \pi_k \leq 1, \forall k$. Answer the following questions:

   (a) (2 points) **Assuming your data is i.i.d., write down the observed data likelihood, $\mathcal{L}^0$.**

   The observed data likelihood is:

$$\mathcal{L}^0 = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \frac{x_i}{\sigma_k^2} e^{-x_i^2/(2\sigma_k^2)}$$

   (b) (2 points) **Use the Expectation-Maximization (EM) algorithm to find the maximum likelihood solutions. Start by introducing the hidden latent variables Z. Describe precisely what they are.**

   Yes, we can introduce the hidden latent variable Z, where z_i corresponds to the Rayleigh component from which sample x_i was drawn from. Moreover, since we have a total of K Rayleigh components, $z_i \in \{1, 2, \ldots, K\}$.

   (c) (3 points) **For the hidden variables you defined above, write down the complete data likelihood, $\mathcal{L}^c$.**

   The complete data likelihood is given by:

$$\mathcal{L}^c = \prod_{i=1}^{N} \pi_{z_i} \frac{x_i}{\sigma_{z_i}^2} e^{-x_i^2/(2\sigma_{z_i}^2)}$$

(d) (4 points) **Write down the EM optimization function, $Q(\Theta, \Theta^t)$, where $\Theta = \{\pi_k, \sigma_k\}_{k=1}^K$. Your final solution should contain the sum of simple (natural-)log-terms.**

The EM optimization function is given by:

$$\begin{aligned}
Q(\Theta, \Theta^t) &= \mathbb{E}_z[\ln(\mathbf{L}^c)|\mathbf{X}, \Theta^t] \\
&= \sum_{z_i=1}^{K} \ln(\mathbf{L}^c) P(z_i|x_i, \Theta^t) \\
&= \sum_{k=1}^{K} \left[\sum_{i=1}^{N} \left(\ln(\pi_k) + \ln(x_i) - 2\ln(\sigma_k) - \frac{x_i^2}{2\sigma_k^2} \right) \right] P(z_i = k|x_i, \Theta^t) \\
&= \sum_{k=1}^{K} \left[\sum_{i=1}^{N} \left(\ln(\pi_k) + \ln(x_i) - 2\ln(\sigma_k) - \frac{x_i^2}{2\sigma_k^2} \right) \right] C_{ik}
\end{aligned}$$

(e) (4 points) **Derive the update equations for the parameters σ_k.**

The solution for the parameter σ_k is:

$$\begin{aligned}
\frac{\partial Q(\Theta, \Theta^t)}{\partial \sigma_k} &= 0 \\
\sum_{i=1}^{N} \left(-\frac{2}{\sigma_k} + \frac{4\sigma_k x_i^2}{(2\sigma_k^2)^2} \right) C_{ik} &= 0 \\
\sum_{i=1}^{N} \left(-2\sigma_k^2 + x_i^2 \right) C_{ik} &= 0 \\
\sum_{i=1}^{N} 2\sigma_k^2 C_{ik} &= \sum_{i=1}^{N} x_i^2 C_{ik} \\
\sigma_k &= \sqrt{\frac{\sum_{i=1}^{N} x_i^2 C_{ik}}{\sum_{i=1}^{N} 2C_{ik}}}
\end{aligned}$$

(f) (5 points) **Derive the update equations for the parameters π_k.**

In order to find the solution for π_k, we must add the constraint $\sum_{k=1}^{K} \pi_k = 1$ to the optimization function:

$$Q_\pi(\Theta, \Theta^t) = Q(\Theta, \Theta^t) - \lambda \left(\sum_{k=1}^{N} \pi_k - 1 \right)$$

The solution for π_k is then:

$$\frac{\partial Q_\pi(\Theta, \Theta^t)}{\partial \pi_k} = 0$$

$$\sum_{i=1}^{N} \frac{1}{\pi_k} C_{ik} - \lambda = 0$$

$$\pi_k = \frac{1}{\lambda} \sum_{i=1}^{N} C_{ik}$$

Since $\sum_{k=1}^{K} \pi_k = 1$, we find that: $\sum_{k=1}^{K} \frac{1}{\lambda} \sum_{i=1}^{N} C_{ik} = 1 \iff \lambda = N$. Hence, the solution for π_k is:

$$\pi_k = \frac{\sum_{i=1}^{N} C_{ik}}{N}$$

2. (10 points) **Compare Gaussian Mixture Models and K-Means as clustering algorithms. List at least 1 similarity and at least 2 differences.**

   Gaussian Mixtuer Models (GMM) can be used as a probabilistic or soft-membership clustering. It determines cluster membership based on maximum a posteriori approach.

   K-Means is a deterministic or hard-membership clustering algorithm. It is a type of centroid-based clustering.

   Both GMM and the K-Means clustering algorithms require the user to define the number of clusters K to learn. Next, both start by initializing the cluster centroids. A centroid in K-Means corresponds to the cluster representative. A data point is assign to the cluster which cluster centroid is closest to. A centroid in GMM is assign to the mean of the Gaussian-ditribution clusters.

   K-Means seeks to find two parameters: the memberships $\pi_{ij} \in \{0, 1\}$, and the centroids locations μ_j. GMM seeks to find 3 parameters: the memberships π_{ij}, the mean centers μ_j, and the covariance matrices Σ_j.

   Since K-Means utilizes a distance metric to assign cluster membership, data scaling is crucial. Data scaling is not as an important factor for GMMs.

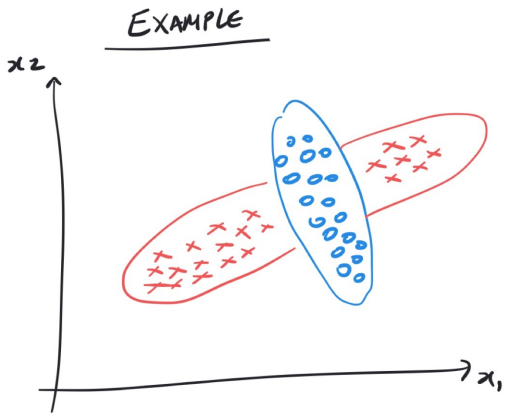
3. (10 points) **This problem is about cluster of validity measures.**

   (a) (4 points) **Between the Gaussian Mixture Models (GMM) and the K-Means with Euclidean distance, which is the more likely to produce a larger silhouette index? Justify your answer.**

   K-Means with Euclidean distance is more likely to produce a larger silhouette index because it clusters data as disjoint compact spherical clusters. Since silhouette looks for compactness and cluster separability, this result will maximize its score.

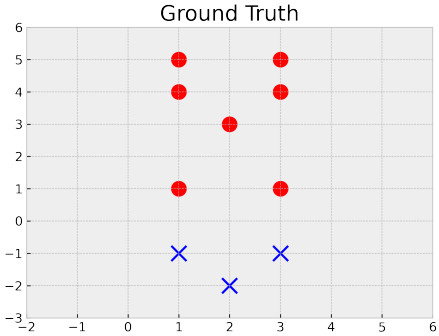
   (b) (2 points) **Draw a 2-cluster example that produces a negative silhouette index.**

   An example is depicted below:



INTRA – DISTANCES ARE LARGER THAN INTER – DISTANCES.

(c) (4 points) **Consider the following clustering result (left) and the ground truth labels (right). The rand index requires the calculation of the parameter a, which corresponds to the number of pairs assigned to the same cluster in both the clustering result and the ground truth.**



Compute the value of a for this example.

By visual inspecting the clustering result and the ground truth, we see that two points, $(1, 1)$ and $(3, 1)$, are incorrectly assigned to the blue/crosses cluster.

The number of pairs assigned to the same cluster in both the clustering result and the ground truth is:

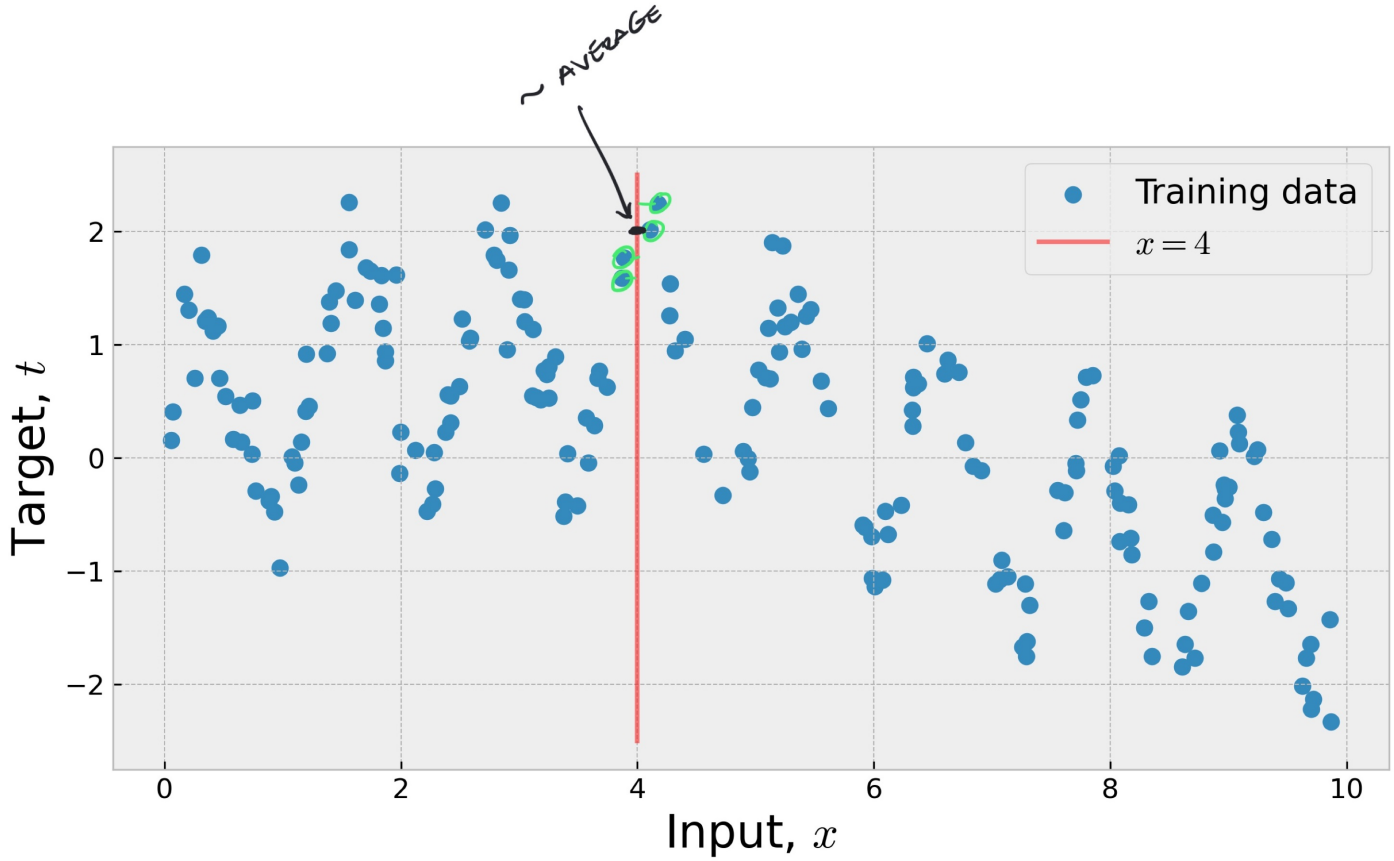
$$\begin{aligned} a &= \binom{5}{2} + \binom{2}{2} + \binom{3}{2} \\ &= \frac{5!}{2!(5-2)!} + 1 + \frac{3!}{2!(3-2)!} \\ &= 10 + 1 + 3 = 14 \end{aligned}$$

4. (10 points) **We saw examples in class of the uniform and weighted K-Nearest Neighbors (KNN) used in the context of classification tasks. Answer the following questions:**

   1. **Explain how we can use KNN for regression tasks.**

2. **Based on the dataset depicted below, what is the (approximate) uniform KNN prediction for $x = 4$ based on $k = 4$ neighbors and Euclidean distance?**

3. **Explain what are the effects on the *best fitting curve* when using uniform vs weighted KNN?**



KNN can be used for regression by predicting a continuous value based on the average of the k nearest neighbors. This is the uniform (or standard) KNN. For the dataset below, at $x = 4$, the $k = 4$ closest (x-value) neighbors are highlighted in the image below. The average target value is approximately $y = 2$.

Weighted KNN for regression tasks will have the same challenges, namely, it will become prone to overfitting.

5. (10 points) **For this problem, consider the Logistic Regression and the Fisher's Linear Discriminant Analysis (FLDA) for a 2-class classification task. Answer the following questions:**

   (a) (4 points) **Write down the objective function for each classifier and clearly specify all parameters.**

   FLDA finds the discriminant function that maximizes between-class separability and within-class compactness. Assuming a 2-class dataset in 2-dimensionality fea-

ture space, we can write the objective function as:

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + 2_2^2} = \frac{\overrightarrow{w}^T \mathbf{S}_B \overrightarrow{w}}{\overrightarrow{w}^T \mathbf{S}_W \overrightarrow{w}}$$

where

$$m_2 - m_1 = \overrightarrow{w}^T (\overrightarrow{m}_2 - \overrightarrow{m}_1)$$

and

$$s_k^2 = \overrightarrow{w}^T \sum_{n \in C_k} (\overrightarrow{x}_n - \overrightarrow{m}_{\mathbf{k}})(\overrightarrow{x}_n - \overrightarrow{m}_{\mathbf{k}})^T \overrightarrow{w}$$

The Logistic Regression algorithm is a probabilistic discriminant function. The objective function is:

$$J(\mathbf{w}, w_0) = \sum_{i=1}^{N} -t_i \log \phi(z_i) - (1 - t_i) \log(1 - \phi(z_i))$$

whwew $t_i \in \{0, 1\}$ is the target label for sample x_i, and $y_i = \phi(z_i) = \frac{1}{1+e^{-z_i}}$ is the mapper function.

(b) (6 points) **Describe, in words, how each classifier learns a decision surface. Compare the two classifiers with respect to the following 5 criteria: (1) whether they are probabilistic or deterministic, (2) sensitivity to outliers, (3) class imbalance, (4) assumptions they make, (5) whether categorical input data can be used.**

FLDA first finds the direction of projection that maximize class separability and minimize class compactness. The discriminant function is them place orthogonal to that direction. FLDA assumes class are Gaussian-distributed, are extremelly sensitive to outliers, require a *decent* sample size in order to obtain good mean and variances estimators, and can only handle continuous data (especifically Gaussian-distributed).

Logistic regression (LR) finds the discriminant function that maximizes the odds ratio yielding a probability of assignment. LR is not as sensitive to outliers, it can handle class imbalance, does not make any distribution assumptions and can utilize categorical data as independent variables.

6. (10 points) **Consider the support vector machine (SVM) classifier for a two-class problem, $t \in \{-1, 1\}$. Answer the following questions:**

(a) (3 points) **What is the distance of a point x_n to the decision surface?**

The distance of a point x_n to the decision surface, $y(x) = \mathbf{w}^T \phi(\mathbf{x}) + b$, is given by:

$$\frac{t_n y(x_n)}{\|w\|} = \frac{t_n \left(w^t \phi(x_n) + b\right)}{\|w\|} \geq 0$$

(b) (4 points) **We derived the following optimization function for the hard-margin SVM:**

$$\arg_{w,b} \max \left\{ \frac{1}{\|w\|} \min_n \left[t_n(w^T \phi(x_n) + b) \right] \right\} \tag{1}$$

We later rewrote it as

$$\begin{aligned} &\arg_{w,b} \min \frac{1}{2}\|w\|^2 \\ &\text{subject to } t_n(w^T \phi(x_n) + b) \geq 1 \end{aligned} \tag{2}$$

Explain how did we move from optimization (1) to (2). In particular, how can we guarantee that $t_n(w^T \phi(x_n) + b) \geq 1 \ \forall n$?

We first rescale $w \to \kappa w$ and $b \to \kappa b$ to set $t_n(w^T \phi(x_n) + b) = 1$, which means κ corresponds to the smallest distance to the decision surface, which, in turn, are the support vectors. In this case, all data points will satisfy the constraint $t_n(w^T \phi(x_n) + b) \geq 1 \ \forall n$.

(c) (3 points) **In the soft-margin SVM, we introduced slack variables, $\xi_n \geq 0$. The optimization function is:**

$$\begin{aligned} &\arg_{w,b} \min \frac{1}{2}\|w\|^2 + C \sum_{n=1}^{N} \xi_n \\ &\text{subject to } t_n y(x_n) \geq 1 - \xi_n, n = 1, \ldots, N \\ &\text{and } \xi_n \geq 0, n = 1, \ldots, N \end{aligned} \tag{3}$$

Each training sample x_n must satisfy two inequality constraints as shown above. The Lagrange optimization introduces two Lagrange multipliers, $a_n \geq 0$ and $\mu_n \geq 0$ for each constraint, respectively.

Describe what type of point x_n if $a_n = 0$ or $a_n > 0$ and $\mu_n = 0$ or $\mu_n > 0$. (The types of points are: support vector, correctly classified and outside margin, correctly classified but inside margin, misclassified).

A subset of the data points may have $a_n = 0$, in which case they do not contribute to the predictive model. The remaining data points constitute the support vectors. These have $a_n > 0$ and hence $t_n y(x_n) = 1 - \xi_n$.

Support vectors for which $0 < a_n < C$ have $\xi_n = 0$ so that $t_n y(x_n) = 1$.

If $a_n < C$, then $\mu_n > 0$, which requires $\xi_n = 0$ and hence such points lie on the margin.

Points with $a_n = C$ can lie inside the margin and can either be correctly classified if $\xi_n \leq 1$ or misclassified if $\xi_n > 1$.

7. (10 points) **This problem is about Principal Component Analysis (PCA). Answer the following questions:**

(a) (5 points) **Write down the pseudo-code for implementing PCA for dimensionality reduction. Make sure to include any preprocessing steps and the dimensions of any matrix/vector dimensions are clearly defined.**

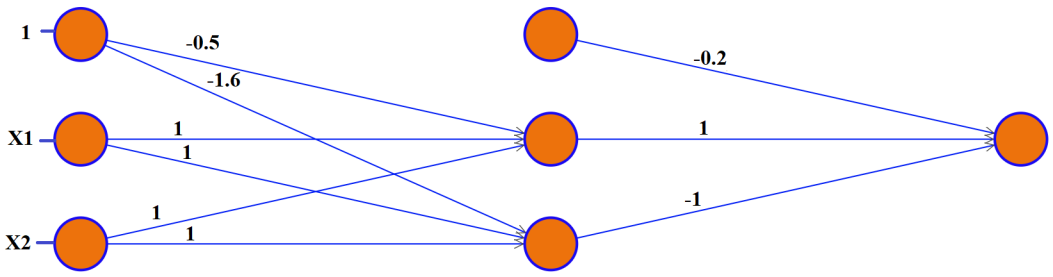
Consider the data X with N data points defined in a D-dimensional space, that is, X is a $D \times N$ matrix. The pseudo-code for PCA is as follows:

1. Subtract the mean, $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$.
2. Compute the covariance matrix R_X (by definition, the covariance already subtracts the mean μ_X). This matrix is of size $D \times D$.
3. Compute eigenvectors and eigenvalues of the matrix R_X, and store the sorted eigenvectors (e_i) in decreasing eigenvalue (λ_i) order.
4. Build the modal matrix $\mathbf{U} = \left[\mathbf{e_1} \mid \mathbf{e_2} \mid \ldots \mid \mathbf{e_D}\right]$, where all the (unit-length) eigenvectors are stacked in columns, sorted in descending order of their eigenvalues, i.e., $\lambda_1 > \lambda_2 > \cdots > \lambda_D$.
   Keep the top M eigenvectors with the largest eigenvalues. Hence $\mathbf{U}$ is a $D \times M$ matrix.
5. Apply the linear transformation: $\mathbf{y} = \mathbf{U}^T\mathbf{X}$. Here $\mathbf{y}$ is a matrix of size $M \times N$, where $M \leq D$.

(b) (5 points) **Suppose you want to use PCA to reduce the dimensionality of your dataset prior to training a classifier. Explain how many principal components you will keep in order to carry the classification task.**

Intead of keeping the number of components that explain 90% of the variance in the data, we can use cross-validation to determine how many number of components will maximize the subsequent classification score.

8. (10 points) **Consider the following neural network architecture: an input layer with 2 units, 1 hidden layer with 2 units and the output layer with 1 unit.**



The weight matrix/vector for the hidden and output layers are $W_H = \begin{bmatrix} -0.5 & -1.6 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$ **and** $W_O = \begin{bmatrix} w_C \\ w_A \\ w_B \end{bmatrix} = \begin{bmatrix} -0.2 \\ 1 \\ -1 \end{bmatrix}$**, respectively.**

Consider the threshold activation function for the hidden layer, $\phi_T(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$**, and the sigmoid activation function for the output layer,** $\phi_s(x) = \frac{1}{1+e^{-x}}$**. Recall that** $\phi_s'(x) = \phi_s(x)(1 - \phi_s(x))$**.**

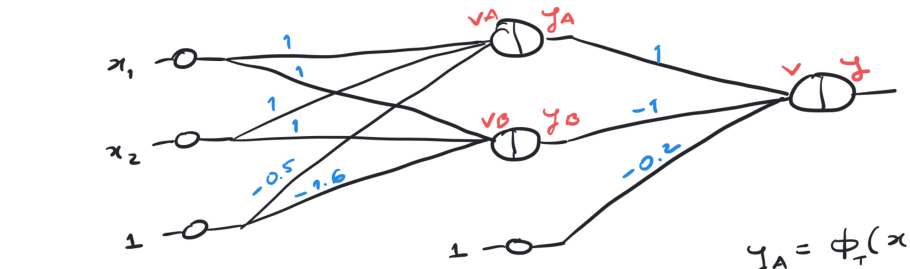
Answer the following questions:

(a) (4 points) **Draw the decision function this architecture is currently learning.**

Since the hidden layer uses a sigmoid activation function, the two units in the hidden layer are drawing boundaries. The first unit outputs $y_A = \phi_T(x_1 + x_2 - 0.5)$, the second unit outputs $y_B = \phi_T(x_1 + x_2 - 1.6)$.

The output layer finds $y = \phi_s(y_A - y_B - 0.2)$.

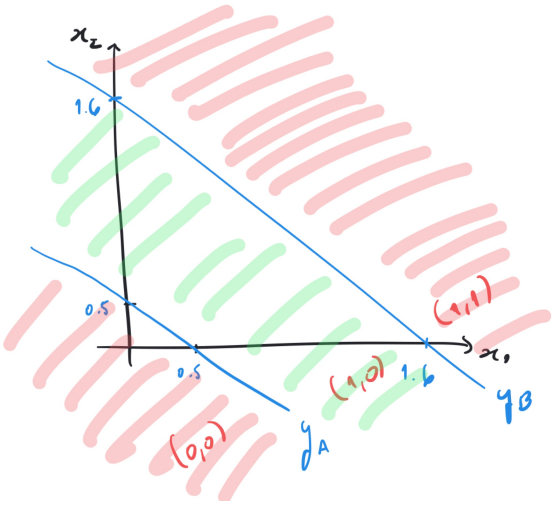
The decision surface learned by this MLP is depicted below:



$y_A = \phi_T(x_1 + x_2 - 0.5)$

$y_B = \phi_T(x_1 + x_2 - 1.6)$

$y = \phi_s(y_A - y_B - 0.2)$



■ – class 1 $(0.5 \leq y \leq 1)$

■ – class 0 $(0 \leq y < 0.5)$

(b) (6 points) **Consider the binary cross-entropy as the objective function**

$$H(y) = \sum_{i=1}^{N} -t_i \ln(y_i) - (1 - t_i) \ln(1 - y_i)$$

where t_i **is the target value of sample** $\mathbf{x}_i$ **and** y_i **is the output of the mapper function.**

Use backpropagation with online learning to update the weight w_A directly connected to the output layer. Consider the data point $\mathbf{x} = \begin{bmatrix}1\\0\end{bmatrix}$ with $t = 1$ and a learning rate of $\eta = 0.01$. Show all your work.

For the data point $\mathbf{x} = \begin{bmatrix}1\\0\end{bmatrix}$, we find: $y_A = 1$, $y_B = 0$, $v = 1 - 0 - 0.2 = 0.8$ and $y = \phi_s(0.8) \approx 0.68997$.

The update equation for w_A is: $w_A = w_A - \eta \frac{\partial H}{\partial w_A}$.

where

$$\begin{aligned}
\frac{\partial H}{\partial w_A} &= \frac{\partial H}{\partial y} \times \frac{\partial y}{\partial v} \times \frac{\partial v}{\partial w_A} \\
&= \left(-\frac{t_i}{y_i} + \frac{1 - t_i}{1 - y_i}\right) \times \phi'(v) \times y_A \\
&= \left(-\frac{1}{0.68997} + \frac{1 - 1}{1 - 0.68997}\right) \times \phi'(0.8) \times 1 \\
&= -\frac{0.2139}{0.68997} \\
&\approx -0.31
\end{aligned}$$

With this, we find the new update value for w_A: $w_A = 1 - 0.01 \times (-0.31) = 1.0031$.

9. (10 points) **This problem addresses the Multi-Layer Perceptron (MLP). Answer the following questions:**

   (a) (5 points) **Can you list all the hyperparameters you can tweak in a basic MLP? If the MLP overfits the training data, how could you tweak these hyperparameters to try to solve the problem?**

   The hyperparameters of a basic MLP are the number of hidden layers, number of units in each layer and each activation function used in the hidden layers and output layer. In addition to these, other techniques may introduce additional hyperparameters such as: type of weight initialization, activation function hyperparameters (e.g., the amount of leak in leaky ReLU), gradient clipping threshold, type of optimizer and its hyperparameters (e.g., the momentum hyperparameter), type of regularization for each layer and regularization hyperparameters (e.g., dropout rate when using dropout), and so on.

   If the MLP overfits the training data, you can try reducing the number of hidden layers and reducing the number of neurons per hidden layer.

   (b) (5 points) **How would you encode the output layer, and its activation function, to perform the following tasks: binary classification, multi-class classification, single-output regression and multiple-output regression.**

   In general, the ReLU activation function (or one of its variants) is a good default for the hidden layers. For the output layer, in general you will want the logistic activation function for binary classification, the softmax activation function for multi-class

classification, or the linear activation function for single-output or multiple-output regression. In regression tasks, if the target is bounded, we can use saturating activation functions such as logistic or tanh (provided the target is scaled to $[0, 1]$ or $[-1, 1]$, respectively). If the target is always positive, we can utilize the ReLU activation function in the output units.