Homework 4 Part 1

This is an individual assignment.

Write your answers using markdown cells or embed handwritten answers with `IPython.display.Image`.

Exercise 1 (8 points)

Consider the following application scenarios. Between the Linear Support Vector Machines (SVM) and Logistic Regression (LR), which one would you select for each scenario? Justify your answers.

1. (2 points) **Detecting fraudulent transactions in a credit card dataset, where the fraudulent transactions might be a minority class and are likely to be distinct from normal transactions.**

SVM will be the best, because SVM can effectively handle imbalanced datasets, prioritizing the minority class. In addition, SVM can capture complex relationships in the data, enhancing their ability to distinguish between normal and fraudulent transactions.

In [ ]:

1. (2 points) **Text classification tasks, where each word or n-gram can be considered as a feature, leading to high-dimensional feature spaces.**

SVM will be the best, because SVM performs well in high-dimensional spaces, making them suitable for text classification tasks where each word or n-gram represents a feature.

In [ ]:

1. (2 points) **Medical diagnosis, where the occurrence of a rare disease is much lower than that of the healthy population.**

LR will be the best. When the occurrence of a rare disease is much lower than that of the healthy population, LR can be adjusted to handle these imbalanced datasets.In addition, LR outputs probabilities, which can be useful in medical scenarios to assess the likelihood of a patient having the disease.

In [ ]:

2. (2 points) **Large-scale text categorization, where the focus is on assigning documents to predefined categories, rather than estimating the probability of belonging to a specific class.**

SVM will be the best. SVM is efficient in high-dimensional spaces, making them well-suited for Large-scale text categorization. What's more, SVM can find the hyperplane that maximizes the margin between different categories, leading to better generalization and robust performance in large-scale text categorization.

In [ ]:

Exercise 2 (10 points)

Consider a feed-forward multilayer perceptron with a D-M-D architecture ($M \leq D$) and linear activation functions. Show that such a neural network, when trained to predict the input at the output layer, performs principal component analysis by considering the minimization it solves. Indicate if the network will require bias terms.

The neural network, with a D-M-D architecture and linear activation functions, performs PCA when trained to predict the input at the output layer. The hidden layer acts as a lower-dimensional representation of the input space, and minimizing the mean squared error during training effectively achieves dimensionality reduction. In this case, bias terms is not necessary because the linear activation functions in each layer allow the network to capture the mean of the data without explicitly using bias terms.

In [ ]:

In [ ]:

Exercise 3 (6 points)

Propose an objective function that incorporates class information into PCA such that instances of the same class are mapped to nearby locations in the new space. You do not

have to solve for a solution, but must explain all terms and their function.

$$J = \text{Trace}\left((X - \bar{X})^T W (X - \bar{X})\right)$$

$(X - \bar{X})$: This part centers the data by subtracting the mean of each feature across instances, a standard PCA step.

$\bar{X}$: Represents the mean matrix, where each column corresponds to the mean value of the corresponding feature across all instances.

W: The weight matrix is a diagonal matrix where each diagonal element W_{ii} represents the weight assigned to the instance i, indicating the importance of preserving spatial relationships within the same class.

$(X - \bar{X})^T W (X - \bar{X})$: Quantifies the reconstruction error in the transformed space, weighted by W, similar to the covariance matrix in traditional PCA but with class-specific weights.

In [ ]:

In [ ]:

Exercise 4 (10 points)

Design the simplest feed-forward MLP, by specifying numerical values for its weights and biases, using the threshold activation function, $\phi(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$, to solve the following scenarios:

1. (2 points) **The NAND logic gate, $\overline{x_1 \cap x_2} = \overline{x_1} \cup \overline{x_2}$, where $x_i \in \{0, 1\}, i = 1, 2$.**

Number of input neurons: 2 (one for each input x_1 and x_2)
Number of hidden layers: 1
Neurons in the hidden layer: 2
Number of output neurons: 1

1. Input Layer -> Hidden Layer: $w_1 = 0.5$, $w_2 = 0.5$ $b_1 = -0.7$
2. Hidden Layer -> Output Layer: $w_3 = -1$, $w_4 = -1$ $b_2 = 1.5$

The threshold activation function $\phi(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$

The output y of the network is given by:

$y = \phi(w_1 x_1 + w_2 x_2 + b_1)$

$$z = \phi(w_3 h_1 + w_4 h_2 + b_2)$$

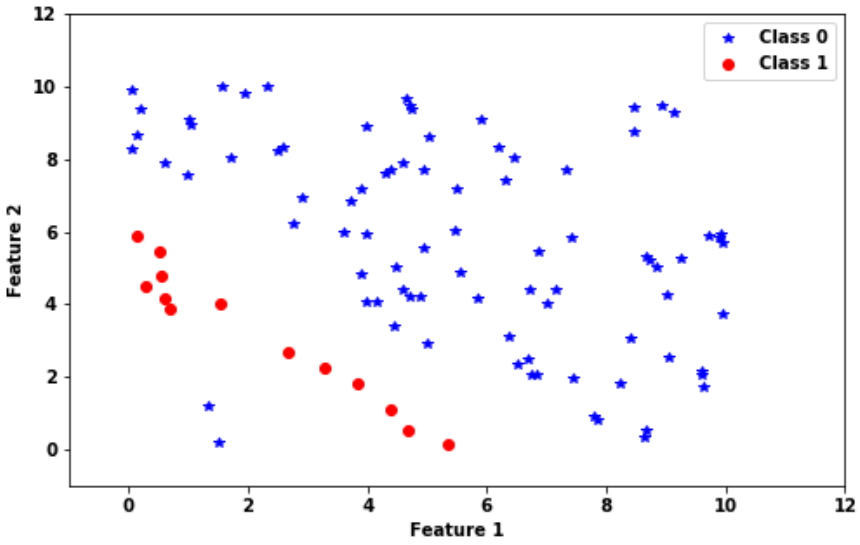
h_1 and h_2 are the hidden layer neurons.

In [ ]:

1. (4 points) **The following two-dimensional dataset,**

In [2]:
```
from IPython.display import Image
Image('figures/dataset.png',width=500)
```

Out[2]:



Number of input neurons: 2 (one for each dimension of the dataset)
Number of hidden layers: 1
Neurons in the hidden layer: 2
Number of output neurons: 1

1. Input Layer -> Hidden Layer: $w_1 = 0.5$, $w_2 = 0.5$ -$b_1 = -0.5$
2. Hidden Layer -> Output Layer: $w_3 = -1$, $w_4 = -1$ $b_2 = 1$

The output y of the network is given by:

$$h_1 = \text{Activation}(w_1 x_1 + w_2 x_2 + b_1)$$

$$h_2 = \text{Activation}(w_3 h_1 + w_4 h_2 + b_2)$$

x_1 and x_2 are the two dimensions of the dataset, and h_1 and h_2 are the hidden layer neurons.

1. (4 points) **The given ground truth table,**

x1	x2	x3	t
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Number of input neurons: 3 (one for each input x_1, x_2, x_3)
Number of hidden layers: 1
Neurons in the hidden layer: 2
Number of output neurons: 1

Weights and Biases:

1. Input Layer -> Hidden Layer: $w_1 = 0.5$, $w_2 = -0.5$, $w_3 = 0.5$ $b_1 = -0.5$
2. Hidden Layer -> Output Layer: $w_4 = 1$, $w_5 = -1$ $b_2 = 0.5$

The output y of the network is given by:

$h_1 = \text{Activation}(w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1)$

$h_2 = \text{Activation}(w_4 h_1 + w_5 h_2 + b_2)$

x_1, x_2, and x_3 are the inputs, and h_1 and h_2 are the hidden layer neurons.

In [ ]:

Exercise 5 (4 points)

Explain why the input-to-hidden weights must be different from each other (e.g., random) or else learning cannot proceed well. Specifically, what happens if the weights are initialized so as to have identical values?

When input-to-hidden weights in a neural network are identical, all neurons in the hidden layer learn the same thing, limiting the network's ability to capture complex patterns. Random initialization of weights ensures diverse learning across neurons, leading to more effective training.

In [ ]:

Exercise 5 (7 points)

Answer the following questions, and provide appropriate justifications, about Convolutional Neural Networks (CNNs):

1. (2 points) **Would you prefer to add more filters in the first convolutional layer or the second convolutional layer?**

Adding more filters in the second convolutional layer is often preferred. This is because the first layer typically captures basic features like edges and textures. The second layer combines these basic features to detect more complex patterns. So, more filters in later layers allow the network to learn more complex features.

In [ ]:

1. (2 points) **Why would you want to add a max pooling layer rather than a convolutional layer with the same stride?**

A max pooling layer is added instead of a convolutional layer with the same stride for two main reasons: to reduce computational load and to provide translation invariance. Max pooling reduces the spatial dimensions (width and height) of the input volume, thus reducing the number of parameters and computation in the network. It also helps make the representation more invariant to small translations of the input.

In [ ]:

1. (3 points) **Consider a CNN composed of three convolutional layers, each with 3×3 kernels, a stride of 2, and "same" padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200×300 pixels. What is the total number of parameters in the CNN?**

For each convolutional layer, the number of parameters is given by the formula:
$(\text{Filter size}^2 \times \text{Input channels} + 1) \times \text{Number of filters}$.
Layer 1: $(3^2 \times 3 + 1) \times 100 = 2,800$ parameters.
Layer 2: $(3^2 \times 100 + 1) \times 200 = 180,200$ parameters.
Layer 3: $(3^2 \times 200 + 1) \times 400 = 720,400$ parameters.

Total parameters = $2,800 + 180,200 + 720,400 = 903,400$ parameters.

In [ ]:

On-Time (5 points)

Submit your assignment before the deadline.

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.
