# ECE 479 - Homework 2

Ahmadreza Eslaminia

Ae15

## Question 1: k-Nearest Neighbor

1. KNN Concepts

   a. (a) False. kNN can be used for both regression and classification by appropriately choosing the distance metric and the number of neighbors to consider.
   b. (b) False. kNN is a non-parametric model because it does not make assumptions about the underlying distribution of the data.
   c. (c) False. The optimal value of k depends on the dataset and the problem at hand, and a larger k value may lead to over smoothing and decreased performance.
   d. (d) False. Training a kNN model involves storing the entire dataset in memory, which can be computationally expensive for large datasets.
   e. (e) False. kNN can suffer from the curse of dimensionality, where the distance metric becomes less meaningful as the number of dimensions increases, and the number of training samples needed to maintain performance grows exponentially.
   f. (f) False. A smaller k value can lead to overfitting, as the model becomes too sensitive to noise in the training data and may not generalize well to new data.

2. KNN Calculation

   a) Here is the Table1 for KNN distances:

| Label | Data | [2.75, 4.5] | [2.75, 5.5] |
|---|---|---|---|
| 0 | [2, 4.5] | 0.75 | 1.25 |
| 0 | [2.5, 3.5] | 1.031 | 2.016 |
| 0 | [3, 6] | 1.521 | 0.559 |
| 1 | [3, 5] | 0.559 | 0.559 |
| 1 | [3.5, 3] | 1.677 | 2.61 |
| 1 | [3.5, 5.5] | 1.25 | 0.75 |
| 1 | [4, 4] | 1.346 | 1.953 |

b) Table 2 which defines the data classification.
For the K=1 and [2.75, 5.5] we have tie and select randomly.

| K = | [2.75, 4.5] | [2.75, 5.5] |
|---|---|---|
| 1 | 1 | 1 |
| 3 | 0 | 1 |
| 5 | 1 | 1 |

c) As we can see from the first data point ([[2.75, 4.5]]) we have 1,0,1 class prediction for K=1,3,5 respectively. So, it is not consistent for different values of K. One of the reasons could be lack of sufficient training data. As we can see, for each class we have a maximum of 4 pieces of data which is not sufficient for using K>1. Also, as we can see from the Figure of data is not dense in these two diminutions so K-NN would not be consistent for different K.

## <u>**Question 2: K-Means Clustering**</u>

1. Clustering Concepts

a) True. Clustering is an unsupervised learning technique that groups similar data points together without requiring prior labeling.

b) False. While SSE is a commonly used objective function in clustering algorithms, it is not the only goal, and some algorithms may use different criteria.

c) False. By assigning a numeric value to the categorical data, clustering can be performed.

d) False. K-means is a specific clustering algorithm that falls under the larger umbrella of clustering techniques.

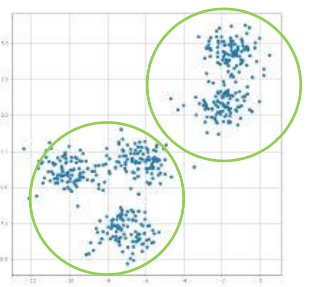
e) True. The value of K is typically determined prior to running the K-means algorithm and represents the number of clusters that the algorithm will try to form.

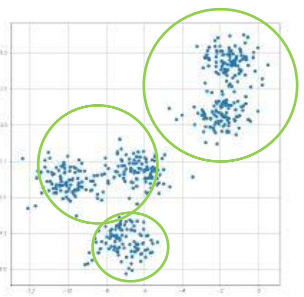
2. Clustering applications:
    a. Customer segmentation:
    Customer segmentation is the process of dividing a company's customer base into specific groups based on common characteristics, such as demographics or purchasing behavior, to tailor marketing campaigns more effectively.
    K-means clustering is commonly used in customer segmentation to group customers into distinct clusters based on similarities in their buying patterns or other relevant features.
    For example, "Customer Segmentation Using K-Means Clustering" is a research article that discusses the application of K-means clustering in customer segmentation
    Link:
    https://www.optimove.com/resources/learning-center/customer-segmentation-via-cluster-analysis#:~:text=In%20the%20context%20of%20customer,archetypes%E2%80%9D%20or%20%E2%80%9Cpersonas%E2%80%9D

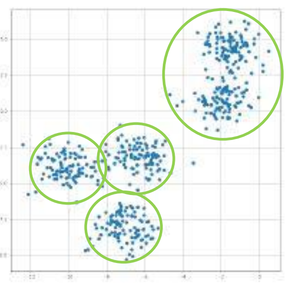
    b. Image Segmentation:
    Image segmentation is the process of dividing an image into multiple segments or regions, each representing a different object or part of the image. Clustering algorithms can be used for image segmentation by grouping pixels together based on their color, intensity, texture, and other features. One popular clustering algorithm used in image segmentation is the mean-shift algorithm.
    Link :
    https://towardsdatascience.com/understanding-mean-shift-clustering-and-implementation-with-python-6d5809a2ac40

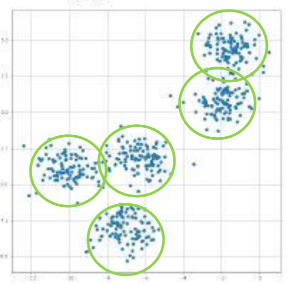
    c. Anomaly Detection:
    Anomaly detection involves identifying data points that deviate significantly from the expected pattern. Clustering algorithms can be used for anomaly detection by grouping together similar data points and identifying outliers. One example of a clustering algorithm used for anomaly detection is the DBSCAN algorithm.
    https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556#:~:text=e.g.%20anomaly%20detection.-,DBSCAN%20algorithm,many%20points%20from%20that%20cluster

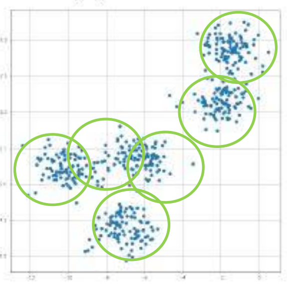
3. Visualizing a K-means problem

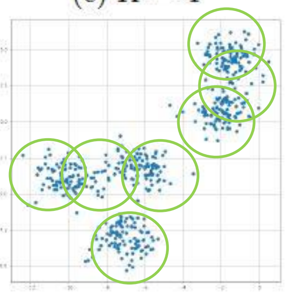

(a) $K = 2$


(b) $K = 3$


(c) $K = 4$


(d) $K = 5$


(e) $K = 6$


(f) $K = 7$

As we know, a low SSE indicates that the data points are tightly clustered around their respective centroids, while a high SSE indicates that the data points are widely spread out, and the clustering solution is not very effective. So, in most cases we should use the elbow method to define the best number of K. The elbow method is one commonly used technique for selecting the optimal number of clusters based on the SSE. As we can see from the SSE-K diagram we can see the elbow is on the K=5. After K=5 the SSE decrease not significantly and before that the decrease is significant. Therefore the k=5 is optimal number for the number of classes. In addition, we can see from the data that k=5 for the cluster number is a reasonable choice.

4. Calculating and minimizing SSE
   The centroid is going to be calculated by getting mean of the data:
   Centroid: [4.5, 1.75]
   With this Centroid the SSE is 10.940344597991736.

   Here you can find SSEs when we remove different points:

| Removed point. | SSE |
|---|---|
| A | 7.383185006927174 |
| B | 8.402373213129577 |
| C | 9.813435502970183 |
| D | 5.549703546891172 |

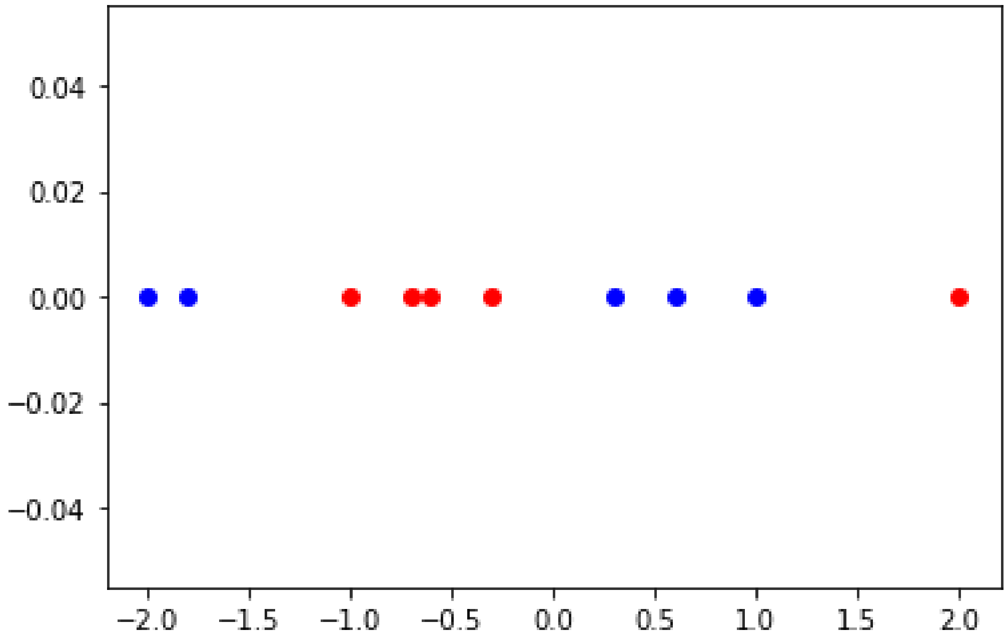
As we can see the biggest drop in the SSE is when removing the D point so we can consider D point as the outlier. Since this point is far from other point the SSE drops significantly when we remove this point which means that this point has had a huge portion of the SSE.

**Question 3: Linear Classifiers, SVM and Mapping Trick**

1. Linear Classifiers:
   (a) The two classes in this problem can be easily separated by a linear classifier. One possible linear classifier is the following:

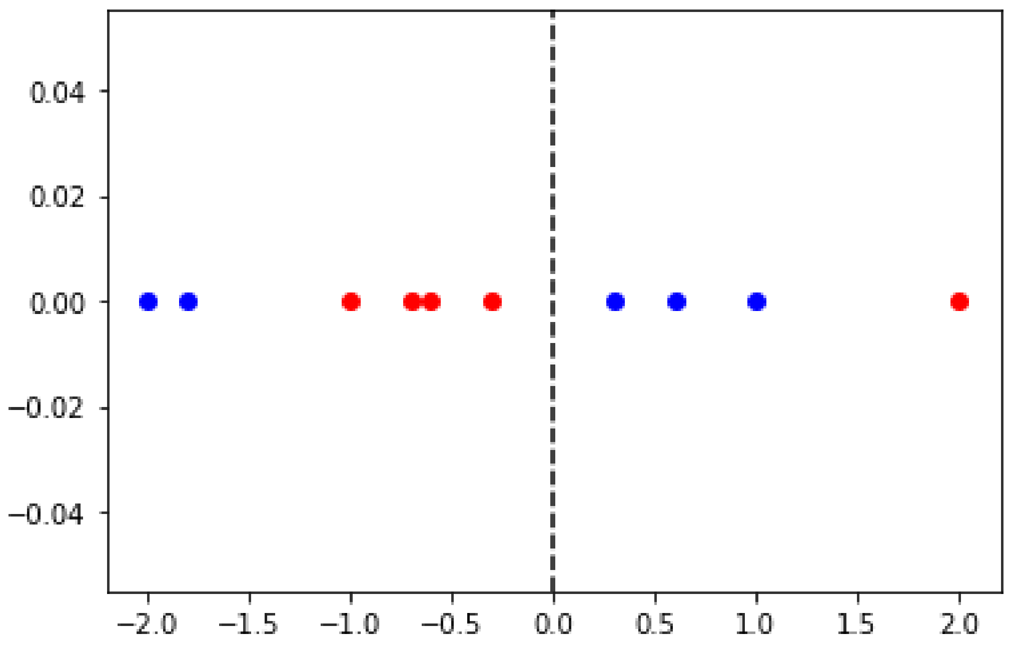
   If x1 + x2 <= 1, predict class 0.
   If x1 + x2 > 1, predict class 1.
   This classifier simply draws a diagonal line through the input space, separating the two classes. Therefore, linear classifiers can learn this pattern.
   (b) In this case, the two classes cannot be separated by a single straight line in the input space. If you draw these points you can see that you cannot find any straight line that separates these data.

2. 1-D Linear Classification
   Class A: (-2, -1.8, 0.3, 0.6, 1) Label: -1
   Class B: (-1, -0.7, -0.6, -0.3, 2) Label: 1



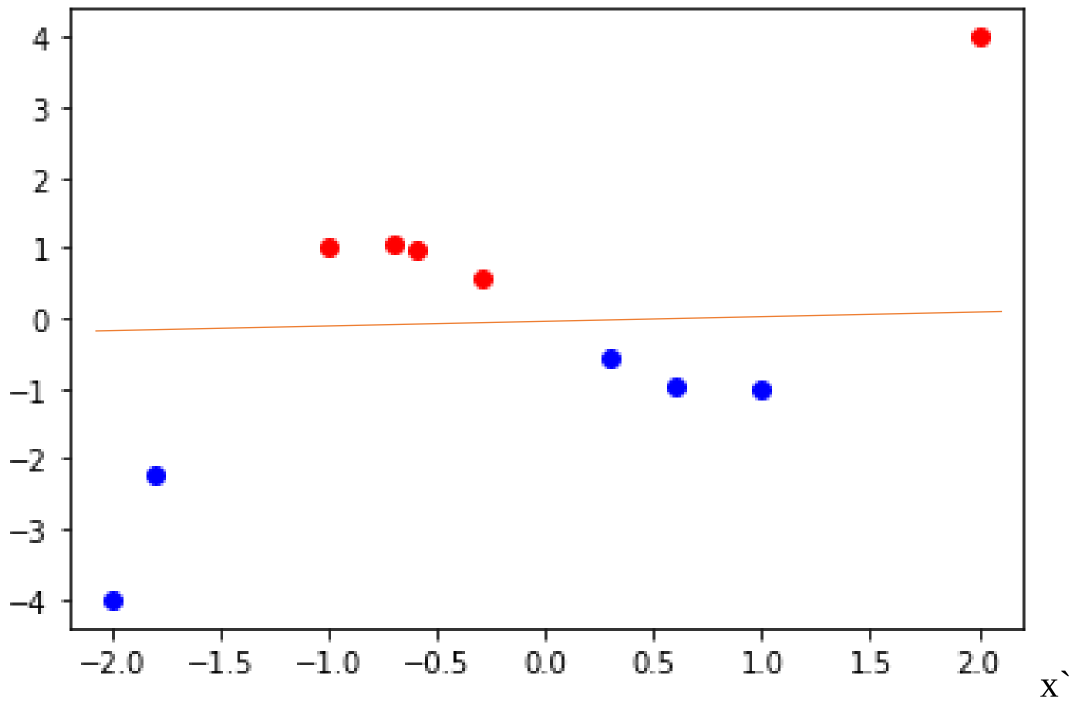
\

Here is a attempt to separate the data:



As we can see from the plot it cannot be separated with a straight line.

3. The Mapping Trick
   Here is the Mapping plot:



As you can see from the plot the orange line has separated two classes, it is linearly separable.

4. SVM and Hyper plane:
   Here is the code that I have used for finding w and b:

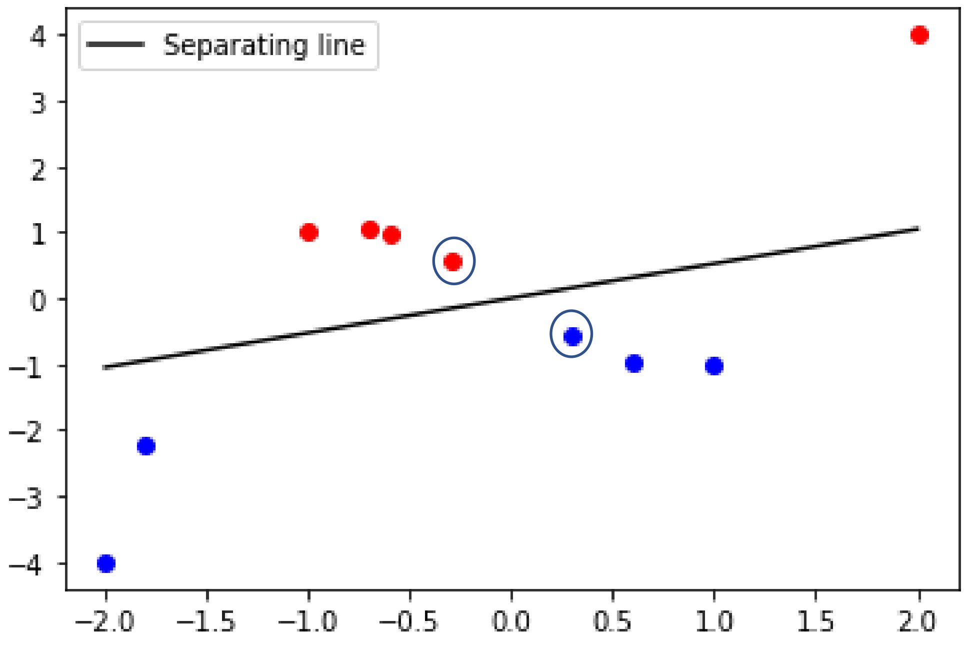
```
from sklearn.svm import SVC

# Define the points and labels
class_A = [-2, -1.8, 0.3, 0.6, 1]
class_B = [-1, -0.7, -0.6, -0.3, 2]
labels = [-1, -1, -1, -1, -1, 1, 1, 1, 1, 1]

# Define the mapping function
def map_func(x):
    return [x, x**3 - 2*x]

# Apply the mapping function to each point
X = [map_func(x) for x in class_A] + [map_func(x) for x in class_B]

# Build the SVM model
model = SVC(kernel='linear', C=1e10)

# Train the model on the data points and labels
model.fit(X, labels)

# Extract the optimal hyperplane parameters
w = model.coef_[0]
b = model.intercept_[0]

print("Optimal hyperplane parameters:")
print("w =", w)
print("b =", b)
```

Optimal hyperplane parameters:

w = [-0.7171389 , 1.3697353]

b = -0.0

Here is the following line with support vectors circled.



5. Margin and optimality:
   as we know from the slides:

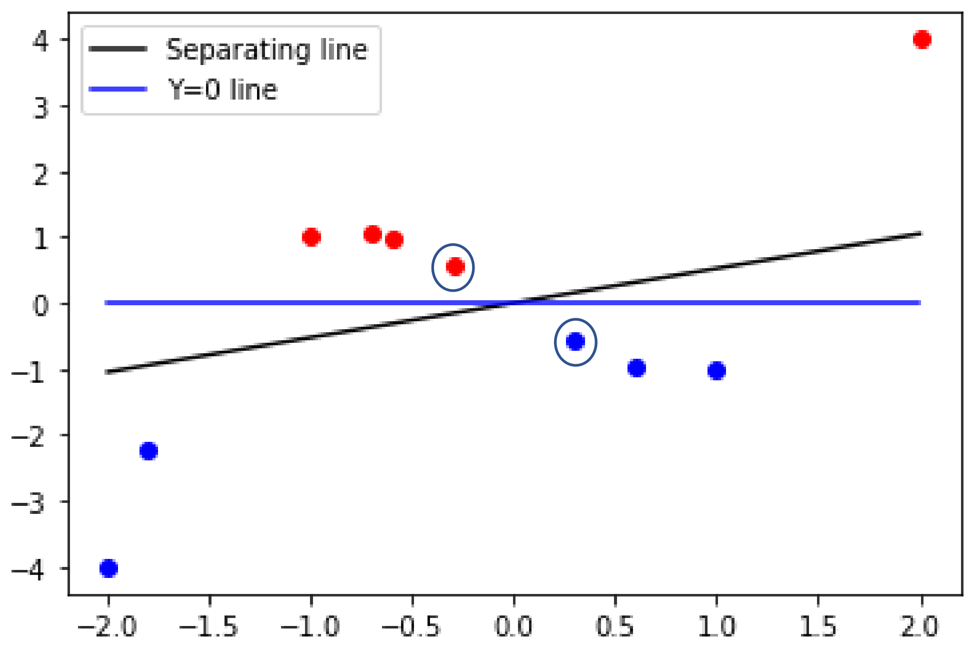
$$d_{SVM} = \frac{2}{\|w\|_2}$$

$\|w\| = \sqrt{(-0.7171389\ )^{\ 2} + (1.3697353)^2}\ = \ 1.546$

$D_{SVM} = 1.2936$

So, the margin is going to be half this number, which is 0.6468.

As you can see in the following picture the for the Y=0 line the two points which are most closest to the line are circled. It is obvious that their distance to the line is smaller than the 0.6468 that we had for the previous line. So it is not an optimal line for separation.


Separating line
Y=0 line
4
3
2
1
0
−1
−2
−3
−4
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

## **<u>Question4 : Neural Networks & Model Compression</u>**

1. Neural network for edge devices
   For network (a), we have nine dense layers with 32 nodes in each layer. Since the input has 16 features, the first layer will have 16 * 32 = 512 weights. Each subsequent layer will also have 32 * 32 = 1024 weights. Finally, the output layer will have 32 * 16 = 512 weights. Therefore, the total number of weights in network (a) is:

   512 + 8 *1024 + 512 = 9216

   If we use float32 to store each weight, then the memory requirements for network (a) will be:

   9472 * 4 bytes/weight = 36864 bytes

   For network (b), we have three dense layers with 128 nodes in each layer. The first layer will have 16 * 128 = 2048 weights, and each subsequent layer will have 128 * 128 = 16,384 weights. The output layer will have 128 * 16 = 2048 weights. Therefore, the total number of weights in network (b) is:

   2048 + 2 * 16,384 + 2048 = 36864

   If we use float32 to store each weight, then the memory requirements for network (b) will be:

   35,864 * 4 bytes/weight = 147456 bytes

   Therefore, the shallower network (b) requires more memory than the deeper network (a). So, it might be a better choice to have the deeper network to get run on the edge.

2. CNN vs MLP

   In the case of a Multi-layer Perceptron (MLP), the shift in the camera's view would have a significant impact on the classification accuracy. MLPs are not able to handle input images that are translated or rotated, and so any shift in the position of the object would result in a change in the input feature values, which would not match the training data. As a result, the MLP would likely fail to correctly classify the object and would suffer from a decrease in classification accuracy.

On the other hand, Convolutional Neural Networks (CNNs) are designed to handle image classification tasks that involve spatial transformations, such as translation and rotation. This is due to the property of translation invariance, which is inherent in the design of CNNs. Specifically, CNNs use convolutional layers that apply the same set of filters to different parts of the input image, allowing them to detect features regardless of their spatial position within the image. In other words, CNNs are able to learn features that are invariant to translations and can classify objects regardless of their position within the image.

3. Batch Normalization:

   Let gamma and beta be the learned scaling and shifting parameters in the batch normalization layer, respectively. During training, the mini-batch mean and variance are used to normalize the input before scaling and shifting:

   x_norm = (x - mu_B) / sqrt(var_B + eps)
   y = gamma * x_norm + beta

   where x is the input, mu_B and var_B are the mini-batch mean and variance, eps is a small constant for numerical stability, and y is the output.

   During inference, the learned scaling and shifting parameters are used to normalize the input:

   x_norm = (x - mu) / sqrt(var + eps)
   y = gamma * x_norm + beta

   where mu and var are the fixed population mean and variance estimated during training.

   To fuse the batch normalization layer into the convolutional layer, we can substitute the normalization and scaling operations with the weights and biases of the convolutional layer. Let alpha = gamma / sqrt(var + eps) and b = beta - mu * alpha, then we have:

y = gamma * x_norm + beta = (gamma / sqrt(var + eps)) * (x - mu) + beta = alpha * x + b

Therefore, the fused convolutional layer has new weights and biases:

w'i = alpha * wi
b'i = alpha * bi + b

4. Weight Quantization:

   Assuming that each weight is stored as a 32-bit (4-byte) FLOAT32 value in memory, the total number of bytes that need to be fetched is N * 4. Therefore, the data transfer latency can be calculated as:

   Data transfer latency = (N * 4) / S (in second)

   If we apply quantization to convert the model to INT8, each weight would be stored as an 8-bit (1-byte) INT8 value in memory. The total number of bytes that need to be fetched would then be N *1. Therefore, the data transfer latency for INT8 quantization can be calculated as:

   Data transfer latency = (N * 1) / S (in second)

5. Weight Quantization vs Weight Clustering:

   If we use clustering to compress the model, the weights can be represented by indices in a table of M centroids. Each index can be represented using log2(M) bits. Therefore, the total number of bits that need to be transferred is N * log2(M) which equals N * log2(M)/8 bytes.

   Let's compare the data transfer latencies for the quantized model and the clustered model:

   For the quantized model, the data transfer latency is (N * 1) / S = 64 / S.
   For the clustered model, the data transfer latency is (N * log2(M)) / 8 + (M * 4)) / S.
   To gain lower data transfer latency with clustering, we need the clustered model to have a lower data transfer latency than the quantized model:

$((N * \log_2(M)) / 8 + (M * 4)) / S < 64 / S$

Simplifying this inequality, we get:

$(N * \log_2(M)) / 8) + M * 4 < 64$

Substituting N = 64 and rearranging, we get:

$M < (16 - 2 * \log_2(M))$

We can use numerical methods to solve for the maximum value of M that satisfies this inequality. One approach is to use trial and error. Trying different values of M, we find that M = 8 is the largest power of 2 that satisfies the inequality.
Therefore, if we use clustering to compress the model, the maximum value of M to gain lower data transfer latency compared to the quantized model is M = 8.

**Question5: Backpropagation**

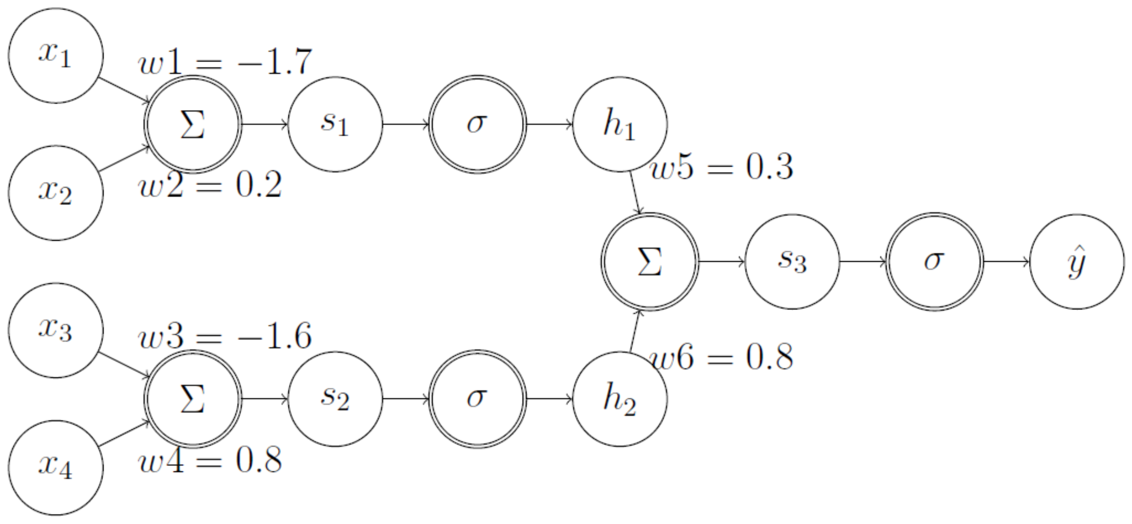


Figure 6: A Simplified Neural Network Example

---

At first, we are going to compute the amount of the variables of the network as follows: ((x1, x2, x3, x4) = (0.3, 1.4, 0.9, –0.6))

$$s_1 = x_1 w_1 + x_2 w_2 = -0.23$$

$$s_2 = x_3 w_3 + x_4 w_4 = -1.92$$

$$h_1 = \sigma(\, s_1) = \frac{1}{1+\ e^{-s1}} = 0.443$$

$$h_2 = \sigma(\, s_2) = \frac{1}{1+\ e^{-s2}} = 0.128$$

$$s_3 = h_{!3} w_5 + h_2 w_6 = \ 0.235$$

$$y = \sigma(\, s_3) = \frac{1}{1+e^{-s_3}} = 0.563$$

So we can compute the desired derivatives with the chain rule:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial s_3} * \frac{\partial s_3}{\partial h_1} * \frac{\partial h_1}{\partial s_1} * \frac{\partial s_1}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = (-2\|y-\hat{y}\|) * \big(1-\sigma(s_3)\big)\big(\sigma(s_3)\big) * w_5 * \ \big(1-\sigma(s_1)\big)\big(\sigma(s_1)\big) * x_1$$
$$= -0.002111$$

In the same way:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial s_3} * \frac{\partial s_3}{\partial h_1} * \frac{\partial h_1}{\partial s_1} * \frac{\partial s_1}{\partial w_2}$$

$$\frac{\partial L}{\partial w_2} = (-2\|y-\hat{y}\|) * \big(1-\sigma(s_3)\big)\big(\sigma(s_3)\big) * w_5 * \big(1-\sigma(s_1)\big)\big(\sigma(s_1)\big) * x_2$$

$$= -0.009851$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial s_3} * \frac{\partial s_3}{\partial h_2} * \frac{\partial h_2}{\partial s_2} * \frac{\partial s_2}{\partial w_3}$$

$$\frac{\partial L}{\partial w_3} = (-2\|y-\hat{y}\|) * \big(1-\sigma(s_3)\big)\big(\sigma(s_3)\big) * w_6 * \big(1-\sigma(s_2)\big)\big(\sigma(s_2)\big) * x_3$$

$$= -0.007605$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial s_3} * \frac{\partial s_3}{\partial h_2} * \frac{\partial h_2}{\partial s_2} * \frac{\partial s_2}{\partial w_4}$$

$$\frac{\partial L}{\partial w_3} = (-2\|y-\hat{y}\|) * \big(1-\sigma(s_3)\big)\big(\sigma(s_3)\big) * w_6 * \big(1-\sigma(s_2)\big)\big(\sigma(s_2)\big) * x_4$$

$$= -0.005089$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial s_3} * \frac{\partial s_3}{\partial w_5} = (-2\|y-\hat{y}\|) * \big(1-\sigma(s_3)\big)\big(\sigma(s_3)\big) * h_1$$

$$= -0.0042146$$

$$\frac{\partial L}{\partial w_6} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial s_3} * \frac{\partial s_3}{\partial w_6} = (-2\|y-\hat{y}\|) * \big(1-\sigma(s_3)\big)\big(\sigma(s_3)\big) * h_2$$

$$= -0.00121513$$

Explain vanishing gradients:

Vanishing gradients refer to the phenomenon where the gradients in a deep neural network become very small during backpropagation, making it difficult for the network to learn effectively. This occurs because the gradients are multiplied together as they propagate through the layers, and if the magnitudes of these gradients are very small, then the overall gradient can become exponentially small, effectively vanishing. As a result, the weights in the earlier layers of the network are not updated effectively, leading to poor performance.