ECE 479 ICC: IoT and Cognitive Computing
Spring 2024, Homework 2

Question 1: K-Means Clustering [20 pts]

1. **Clustering concepts [5 pts]**

   (a) Clustering does not require prior labeling, thus it is a kind of unsupervised learning. Please answer True or False. **[1 pts]**

   True

   (b) The goal of all clustering algorithms is to minimize the total within-cluster SSEs (sum of squared error). Please answer True or False and explain your answer. **[2 pts]**

   False. The goal is to separate data so that data similar to each other are in the same group, while data dissimilar with each other are in different groups . Objective is to minimize the SSE.

   (c) It is impossible to define similarity in terms of clustering for categorical data. Please answer True or False and explain your answer. **[2 pts]**

   False
   For numerical data, similarity measures like Euclidean distance is commonly used. However, these measures are not directly applicable to categorical data because categorical variables do not have a natural numerical interpretation. However, we can try Jaccard coefficient, the Hamming distance.

2. **Clustering applications [6 pts]**
   Please list at least **three real applications** of the clustering algorithm. Each of them should include three to four sentences:

   (a) a brief description of the application

   (b) a reference to support your statement (URL link or title of the articles, etc.)
   1. Clustering algorithms are widely used in marketing to segment customers

based on their behavior, preferences, and demographics. By grouping customers with similar characteristics together, businesses can tailor marketing strategies and product offerings to specific segments, thereby increasing customer satisfaction and retention.
Reference: https://towardsdatascience.com/clustering-algorithm-for-customer-segmentation-e2d79e28cbc3
2. Clustering algorithms play a crucial role in medical image analysis, particularly in image segmentation tasks. Medical images are often complex and contain multiple structures or tissues. Clustering algorithms can be applied to partition the image into different regions or segments based on similarities in intensity, texture, or other image features. This segmentation helps in identifying and analyzing specific structures or abnormalities within the image, aiding diagnosis, treatment planning, and medical research.
Reference: https://link.springer.com/article/10.1007/s11042-021-10594-9
3. Clustering algorithms are utilized in transportation systems for traffic analysis and route planning. By clustering traffic flow data from sensors, GPS devices, or traffic cameras, patterns and congestion hotspots can be identified on road networks. This information is valuable for traffic management authorities to optimize traffic flow, plan infrastructure improvements, and provide real-time navigation guidance to drivers. Clustering techniques also play a role in ride-sharing services for grouping passengers with similar destinations to optimize route planning and resource allocation.
Reference: https://www.sciencedirect.com/science/article/pii/S2666691X20300142

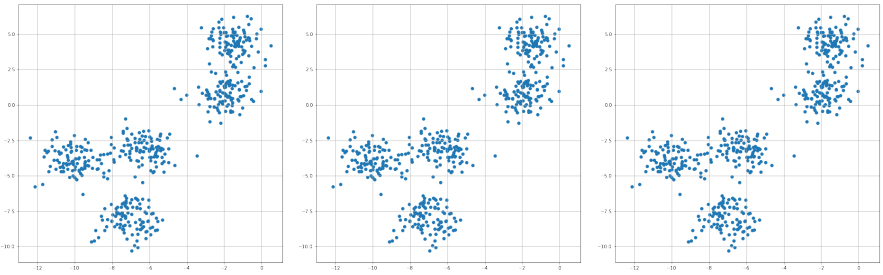
3. **Visualizing a K-means problem [4 pts]**

This part of the question aims to help you build an intuitive sense of how the "K-means algorithm" works. The following data points are randomly generated with a pre-set number of centroids.

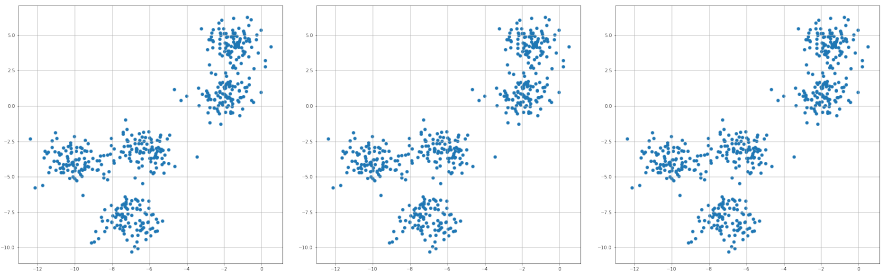
In the following diagrams, try to circle out clusters according to the given K in each of them and then comment on the results. Give your best guess on what should be the optimal K, and explain the reason using the "SSE vs K" diagram. (Discussed in the Lecture and please refer to Figure 2. For example: if "K=1", you should circle all data points to be the same cluster since there is only 1 cluster.)

circle by notebook

K = 5 is the optimal K. We use Elbow method to find the optimal "K". We can see from the "SSE vs K" diagram. As the number of clusters increases, the value tends to decrease because more clusters allow for tighter fits to the data. However, at some point, adding more clusters does not significantly decrease the value. The "elbow" point on the curve represents the optimal number of clusters, where adding more clusters does not lead to a significant improvement in value.



(a) $K = 2$ (b) $K = 3$ (c) $K = 4$



(d) $K = 5$ (e) $K = 6$ (f) $K = 7$

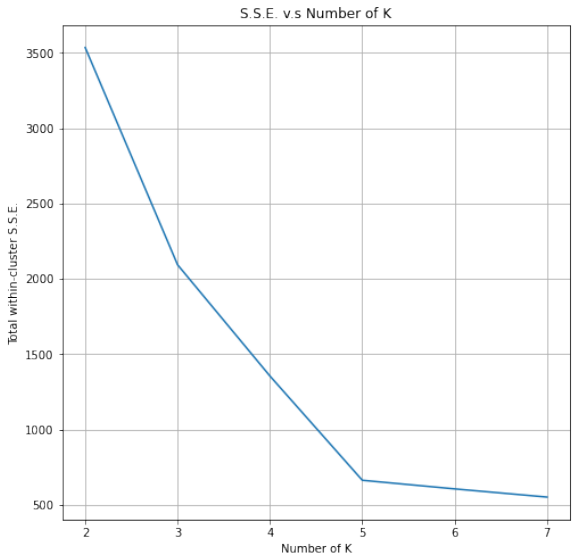


Figure 2: SSE v.s Number of K

4. **Calculating and minimizing SSE [5 pts]** In this question, calculate the within-cluster SSE with the given information. You should first find out the centroid that minimizes the SSE (We're using Euclidean distance). Please refer to Figure 4.

$$A(1, 1)$$
$$B(3, 3)$$

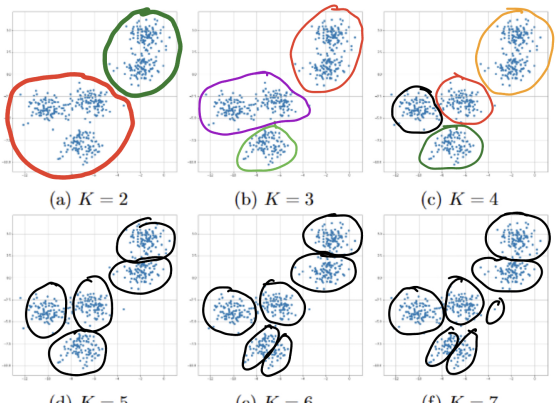


Figure 3: ANS

$C(5, 1)$
$D(9, 2)$

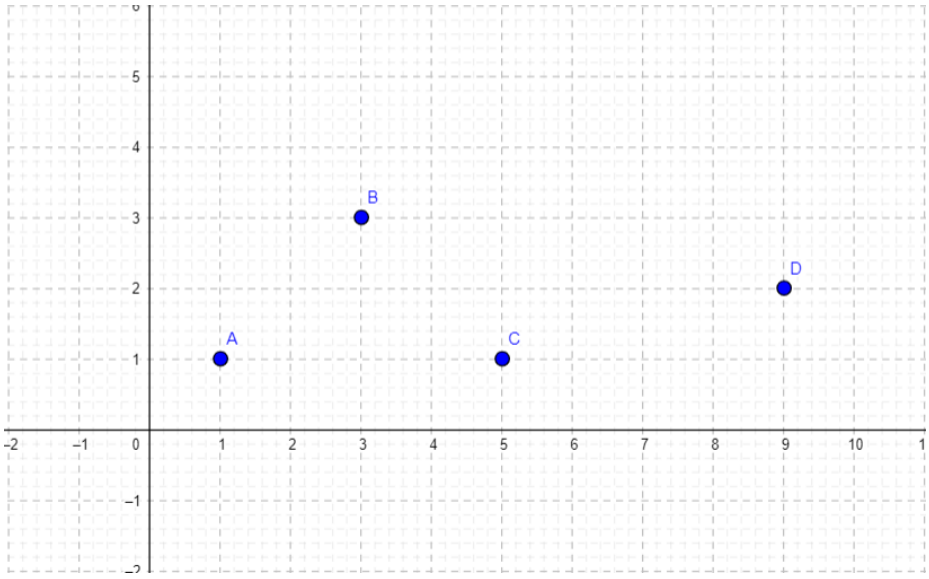


Figure 4: Points Within a Cluster

The detection of outliers (points that should be put into another cluster) is another challenge in clustering problems, and a naive way to identify the outlier is to remove one of the points and then observe the change of within-cluster SSE. Repeat this step until the outlier that leads to the largest drop in SSE is confirmed.

Assume we know there is an outlier in the given cluster, identify it, and calculate the new minimum within-cluster SSE. Then, explain why you choose this point

as the outlier.
D is outliers. From the graph, we can clearly see that D is away from the area that shaped by point A, B and C. If we add D, the centroid will move right which make SSE larger. And I have set outliers respectively to calculate their SSE and find that SSE is smallest when D is outliers.
centroid: [[3, 1.66666667]] SSE: 10.666666666666668

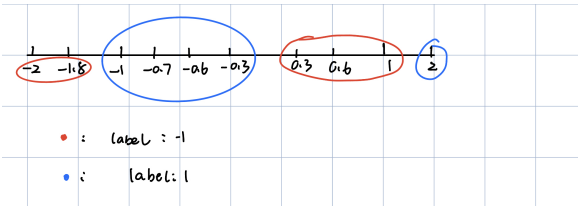


Figure 5: 1D Classification

Question 2: Linear Classifiers, SVM and Mapping Trick [21 pts]

1. **Linear Classifiers [4 pts]**
   Consider two classification problems in a 2D space:

   (a) Data points belonging to label `'0'`: {(0,0), (0,1)}.
   Data points belonging to label `'1'`: {(1,0), (1,1)}.

   (b) Data points belonging to label `'0'`: {(0,0), (1,1)}.
   Data points belonging to label `'1'`: {(1,0), (0,1)}.

   Can linear classifiers learn each pattern in these two classification problems? Justify your answer.
   For problem a, we can use $x = 1$ as classifier. Points with $x < 0.5$ would be classified as '0', and points with$x0.5$ would be classified as '1'. Therefore, a linear classifier can learn this pattern.
   For problem b, we can't find such a line which seperate two sets of data points.

2. **1-D Linear Classification [5 pts]**
   To demonstrate the mapping trick as we discussed in the lecture, we first consider a 1D classification problem. Suppose that we have the following points to be classified:

   Class A: (-2, -1.8, 0.3, 0.6, 1) Label: -1
   Class B: (-1, -0.7, -0.6, -0.3, 2) Label: 1

   Plot the points on a number axis, and comment on whether they can be perfectly classified with a 1D linear classifier.
   Hint: a linear classifier in 1D can be seen as assigning different labels to $x > b$ and $x < b$ where b is the division (hyperplane).

   They can't be classified with 1D linear classifier. Because when plot the points on number axis, we can see that the range of two dataset is intersecting which means we can't find a value b to divide them into two classes.

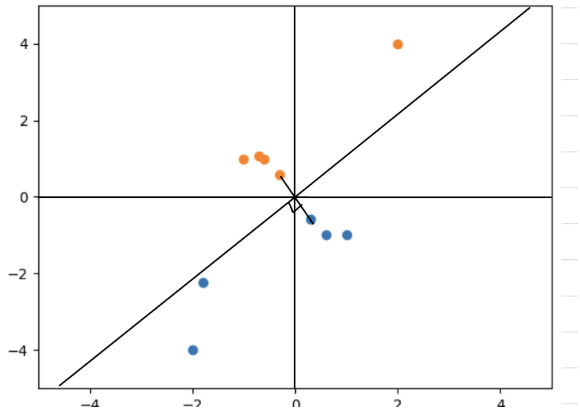


Figure 6: ANS

3. **The Mapping Trick [6 pts]**
   Now, transform the points into a 2D space using the following mapping function:

$$\psi(x) = (x, x^3 - 2x)$$

   Again, plot the points in a 2D coordinate system and comment on whether they can be classified with a simple linear classifier and explain why or why not. Make sure the scales of your x-axis and y-axis are the same; it will make your next questions much easier.

   They can be classified with a simple linear classifier. For example $y = 0$ can be a linear classifier. (Blue: label-1, Orange: label1)

4. **SVM and hyperplane [6 pts]**
   Based on the plot that you have obtained from the previous questions, follow the steps below and find the hyperplane for the SVM. (You don't need to solve for the line algebraically.)

   (a) What is the geometric interpretation of the hyperplane in SVM?

   In two or three dimensions, a hyperplane is simply a flat plane that separates the space into two halves. For instance, in 2D, a hyperplane is just a straight line, and in 3D, it's a flat plane. The key idea in SVM is to find the hyperplane that maximally separates the two classes of data points. In the lecture, we define it as $w^T * x + b = 0$. SVM chooses the decision hyperplane that sits in the middle of the two closest data samples. This hyperplane will maximize the margin between two classes.

   (b) Circle the point in each class that is the closest to the other class. What is the relationship between these two points and the hyperplane?

In Figure6, I connect two closest points. SVM chooses the decision hyperplane that sits in the middle of the two closest data samples The points that are closest to the other class define support vector whih define the margin of hyperplane.

(c) Based on the above relationship, describe how to find the hyperplane (which should simply be a straight line) and draw it in the plot.
1. Calculate the middle point between two support vectors. This point is on the hyperplane.
2. The hyperplane will perpendicular to line connecting support vector. We can derive the slope.
3. We can use the slope and the middle point to derive the hyperplane.

Question 3: Neural Network Basics [20 pts]

1. For a classification problem with 16 input features and 16 classes, compare the following two networks:

   (a) a deeper network with nine dense (fully connected) layers with 32 nodes in each layer

   (b) a shallower network that has three dense (fully connected) layers with 128 nodes in each layer

   If we use `float32` to store the weights, what are the memory requirements of these two configurations? Which one would you prefer to deploy on an edge device? (The networks have no bias. 32 bits is 4 bytes. The network in Slide 4 of Lecture 9 has four fully connected layers.) **[4 pts]**
   Hint: The input layer is not included in the above configuration.

   For a, weights that need to be stored are 2*16*32 + 9*32*32 = 9216 Memory requirement: 8192 * 4 bytes = 36864 bytes
   For b, weights that need to be stored are 16*128 + 2*128*128 = 36864 Memory requirement: 17408*4 bytes = 147456 bytes
   I would prefer network a because edge device need lite network which contain less weights. Network with nine layers have lower memory requirement.

2. You have trained a DNN model for static image classification applications. You deploy the model in the field with a camera. However, you find that the camera's view has shifted in space compared to the camera used for capturing training images. In other words, the area of interest has moved off-center while the objects in your training images are perfectly at the center. What would happen to the classification accuracy if you used (1) a Multi-layer Perceptron or (2) a Convolutional Neural Network as your model? Justify your answer (Think about Translation Invariance).
   What would happen to the performance of the model if you add max pooling layers into CNN in this scenario? **[3 pts]**

   MLP: The network doesn't posses spatial feature or translation invariance (the ability to recognize objects regardless of their position of the input data.). Therefore, MLP may perform poorly in off-center scenario. It doesn't have mechanisms to learn spatial features

   CNN: The network use convolution to capture local patterns and hierarchical structures in images. CNN is more robust to spatial transformations due to the shared weights and local connectivity of convolutional layers. This property enables them to learn spatial hierarchies and extract features that are invariant

to small translations, rotations, and other spatial transformations. It is likely to maintain high accuracy.

The network will become more robust. Becasue max pooling layers will capture the max value within each pooling region, the exact location of features becomes less important, as long as they are present within the pooling window. Max pooling layers help in downsampling the feature maps, which reduces the spatial dimensions and hence the number of parameters in subsequent layers. However, excessive pooling can result in a loss of spatial information, which may be crucial for tasks where precise localization or spatial relationships between features are important.

3. Batch normalization layers are used in most of today's state-of-the-art convolutional neural networks. We can *fuse* the batch normalization layer into the convolutional layer. Suppose we have w_i and b_i to denote the weight and bias in the convolutional layer. Write down the new w_i' and b_i' for the fused convolutional layer with the parameters from the batch normalization. **[4 pts]**

   For the weights: $w_i' = \frac{\gamma}{\sigma} \cdot w_i$
   For the biases: $b_i' = \frac{\gamma}{\sigma} \cdot (b_i - \mu) + \beta$
   β and γ are the scale and shift parameters of the batch normalization layer. σ and μ are the mean and standard deviation computed during the batch normalization process.

4. You are building a classifier and considering three different activation functions: ReLU, tanh, and unit step function.

   (a) Please sketch the three activation functions. **[3 pts]**

   (b) Which one is the most popular of the three? Name two advantages of this activation function over the other two. **[2 pts]**
   Relu is the most popular. Sparsity of Activation: ReLU introduces sparsity by zeroing out negative values. This sparsity can help in reducing overfitting by preventing the co-adaptation of neurons, thereby improving generalization performance.
   Avoidance of Vanishing Gradient: ReLU addresses the vanishing gradient problem more effectively than tanh and unit step function. In deep neural networks, gradients can diminish as they propagate backward through layers, leading to slow convergence or even halting of training. ReLU's gradient is 1 for positive values, which helps mitigate the vanishing gradient problem compared to tanh and unit step function

5. Assuming your design needs to fetch N weights from the memory and the average memory access bandwidth is B bytes per second, calculate the data transfer

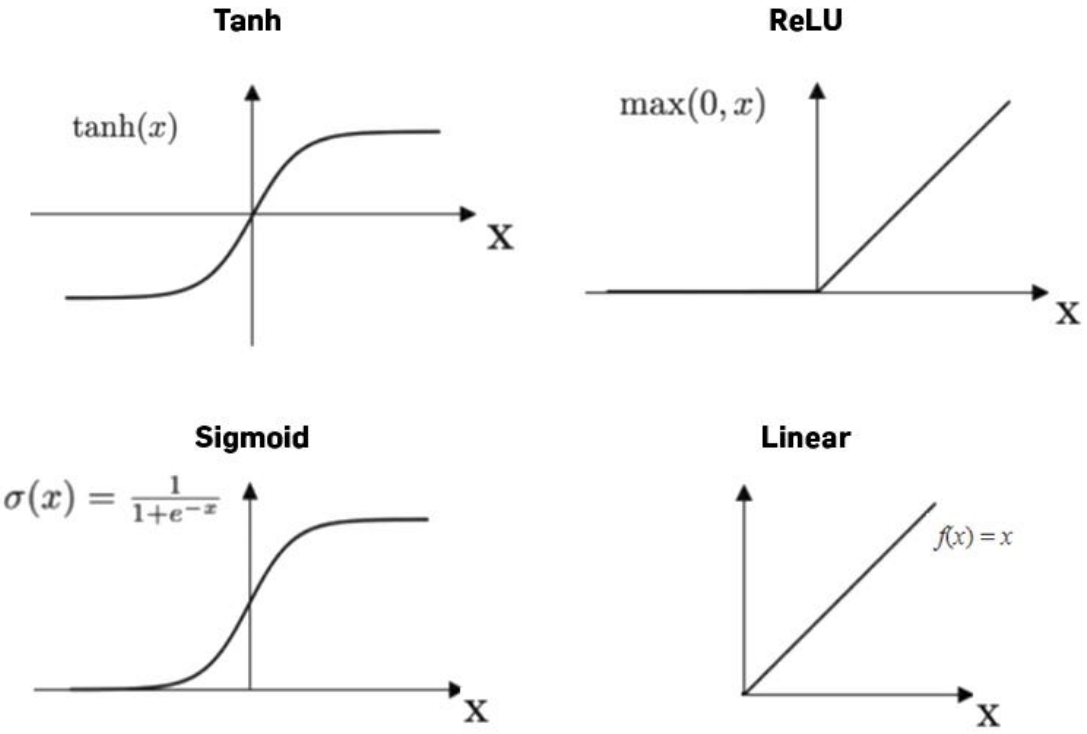


Figure 7: Activation Function

latency (in the unit of second) when using FLOAT32 to represent each weight. Then, calculate the data transfer latency if we apply quantization to convert the model to INT8. [**4 pts**]
Latency float32: N * 4 / B
Latency INT8: N/B

Question 4: Convolution [16 pts]

1. We want to perform a convolution operation on a feature map. This function takes the feature map and the kernel as inputs and returns the output features. Calculate and fill in the parentheses with proper numbers of the dimensions of the output feature map and loop ranges. Assume there is no padding to the input feature map.[**6 pts**]

```
def convolution(input_fm, kernel) {
    """
    Inputs:
    - input_fm:   A 3D numpy array of shape
                  (3, 223, 223).
    - weights:    A 4D numpy array of shape
                  (16, 3, 3, 3).
    - Stride:     2
    Returns:
    - output_fm:  A 3D numpy array of shape
                  (channel_out, height_out, width_out).
    """

    height_out = (     221       ) #height of the output feature map
    width_out = (      221      ) #width of the output feature map
    channel_out = (   16       ) #channel number of the output feature map

    output_fm = np.zeros((channel_out, height_out, width_out))
    #empty feature map in the designated shape

    for row in range(      height_out      ):
        for col in range(     width_out       ):
            for to in range(  channel_out           ):
                for ti in range(      weights[1]      ):
                    for ki in range(  weights[2]         ):
                        for kj in range(     weights[3]       ):
                            output_fm[to][row][col] += \
        weights[to][ti][ki][kj] * input_fm[ti][row + ki][col + kj]

    return output_fm
}
```

Figure 8: Q4.1 Stride 1 Convolution

2. If we wish to perform a stride-2 convolution instead of a stride-1 (unit stride) convolution such as the one above, which numbers will change? Please fill in the new numbers in the blanks below. Again, no padding for the input feature map.[**2 pts**]

```
def convolution_s2(input_fm, kernel) {
    """
    Inputs:
    - input_fm:   A 3D numpy array of shape
                    (3, 223, 223).
    - weights:    A 4D numpy array of shape
                    (16, 3, 3, 3).

    Returns:
    - output_fm:  A 3D numpy array of shape
                    (channel_out, height_out, width_out).
    """

    height_out = (     111        )
    output_out = (    111         )
    channel_out = (     16        )

    output_fm = np.zeros((channel_out, height_out, width_out))

    for row in range(     height_out       ):
        for col in range(     width_out       ):
            for to in range(      channel_out     ):
                for ti in range(   wieghts[1]         ):
                    for ki in range(    wieghts[2]        ):
                        for kj in range(       wieghts[3]      ):
                            output_fm[to][row][col] += \
        weights[to][ti][ki][kj] * input_fm[ti][S*row + ki][S*col + kj]

    return output_fm
}
```

Figure 9: Q4.2 Stride 2 Convolution

3. Consider the first question with the normal unit-stride convolution; we use that as one layer in our neural network. Please think about the following questions and answer in one to two sentences.

   (a) Can the loops that go through the kernel dimension (ki, kj) be swapped with the ones that go through the output dimensions (row, col) without causing any issues? Explain when it's okay to switch the order of loops and

how it might affect the speed and way of accessing memory. **[2 pts]**
No, we can't. If we swapped two value, weights value will be fixed when we calculate the output. Changing the loop order in this manner would lead to incorrect results because each output position would be calculated using the entire input data, rather than the localized region specified by the kernel. It might impact the efficiency of memory access and computational performance. When the value of kernel is symmetry, it will produce the same results regardless the order of data. It might affect the way of accessing memory. We can easily access output memory if we use original way because those memory location is consecutive. It reduce the time that we need to access the memory.

(b) If you wish to increase the **receptive field** of your network layer, what do you need to change? **[2 pts]**
We can use larger convolutional filters (kernels) in the network. We can also increase the number of layers or use pooling layers to get more information. The receptive field refers to the spatial extent of input data.

(c) What is one potential benefit of a larger receptive field? Can a larger receptive field do any harm in practical applications? **[2 pts]**
It can increase robust to detect local feature. A larger receptive field allows the network to perceive broader spatial relationships within the input data. But at the same time, it will increase the number of weights which increase complexity. This may result in longer training times, higher memory consumption. Expanding the receptive field excessively without appropriate regularization techniques can increase the risk of overfitting

(d) What is the significance of the output channel and how is the number of output channels determined? Why is it usually larger than the input channel? **[2 pts]**
It will be determined on the number of kernal. We use different kernal (filter) to capture different characteristic of object. By having multiple channels, the network can learn a diverse set of features, capturing various patterns, textures, and structures present in the input data. The additional output channels enable the network to aggregate information from multiple perspectives or viewpoints, leading to more robust feature.

Question 5: Backpropagation [15 pts]

Consider a simple neural network in Figure 10. Single-circled nodes denote variables: x_1 is an input variable, h_1 and h_2 are intermediate variables (first layer), and $\hat{y}$ is an output variable (second layer), etc. Double-circled nodes denote functions: Σ takes the sum of its inputs, and σ denotes the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$.

Suppose we have an MSE (Mean Square Error) loss $L(y, \hat{y}) = \|y - \hat{y}\|_2^2$, We are given a data point $(x1, x2, x3, x4) = (0.3, 1.4, 0.9, -0.6)$ with the true label 0.37. The gradient of the MSE loss function is $2\|y - \hat{y}\|$.

1. First perform forward propagation and compute s_1, s_2, s_3, h_1, h_2, and $\hat{y}$. Then, use the backpropagation algorithm introduced in Lecture 8 to compute the partial derivative $\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial w_5}, \frac{\partial L}{\partial w_6}$. **[12 pts]**

   Hint: Remember that during backpropagation, the gradient computed for a later layer can be reused when calculating the gradient for earlier layers.
   s1: -0.23 h1: 0.44275 s2: -1.92 h2: 0.12786 s3: 0.2351 $\hat{y}$: 0.5585
   $\frac{\partial L}{\partial w_1}$ = -2(target - predict)* (1-pre) * pre * (0.3) * (1-h1) * h1 * (x1) = 0.0119
   $\frac{\partial L}{\partial w_2}$ = -2(target - predict)* (1-pre) * pre * (0.3) * (1-h1) * h1 * (x2) =0.0412
   $\frac{\partial L}{\partial w_3}$ = -2(target - predict)* (1-pre) * pre * (0.8) * (1-h2) * h2 * (x3) = -0.005
   $\frac{\partial L}{\partial w_4}$ = -2(target - predict)* (1-pre) * pre * (0.8) * (1-h2) * h2 * (x4) = 0.0075
   $\frac{\partial L}{\partial w_5}$ = -2(target - predict)* (1-pre) * pre * (h1) = 0.0100
   $\frac{\partial L}{\partial w_6}$ = -2(target - predict)* (1-pre) * pre * (h2) =0.0020

2. Explain vanishing gradients and exploding gradients. You can watch this video before answering the question. **[3 pts]**
   Vanishing gradients occur when the gradient of the loss function with respect to the parameters become very small. The problem arises because during back propagation, gradients are multiplied across each layer. If gradients are less than 1, they can diminish exponentially as they move backward through the layers, eventually becoming effectively zero.
   Exploding gradients, on the other hand, occur when the gradients grow exponentially as they are propagated backward through the layers during training. It often happen when we set parametersminappropriately or if the learn rate is too high.

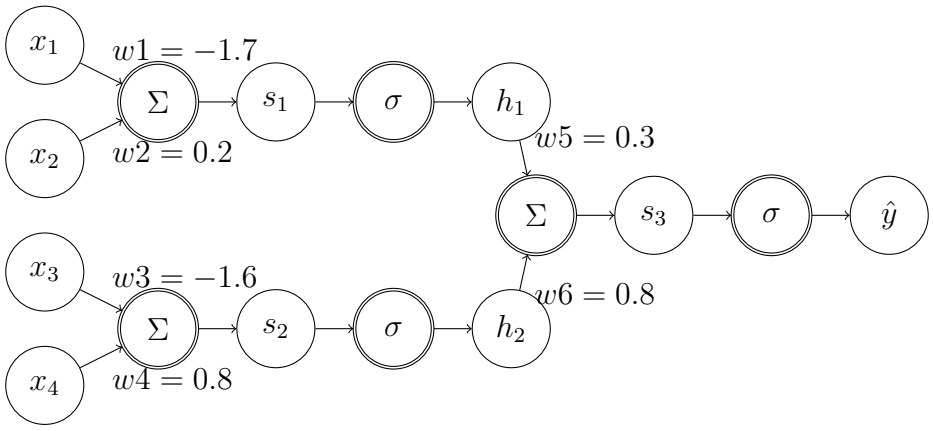


Figure 10: A Simplified Neural Network Example

Question 6: Find Bugs in Tensorflow Code [8 pts]

Bob has recently working on Lab 2. He is not satisfied with small datasets such as Fashion MNIST, and he wants to explore more interesting datasets. He decides to build a simple Multi-Layer Perceptron (MLP) classifier using TensorFlow with Keras to classify images from the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class, which is a cool choice for developing and testing image classification models.

However, Bob is relatively new to TensorFlow and neural network design. He wrote the following code to build, compile, and train his MLP model. However, Bob noticed that his model doesn't work, and he suspects there must be some bugs in his code.

As Bob's helpful friend, you would like to review Bob's code, identify any bugs that might be present, and **explain** how they could affect the model's performance by making a short comment next to the code. After identifying the bugs, suggest corrections to make the model work. (Hints: There are 4 bugs in the code. For information on possible loss functions, please refer to the link.)
Bug 1: Line 10. We need to dived train images and test images by 255.
If we mutiply 255, we can still train network but increase computation complexity.

```
1
2  # Normalize the images
3  train_images, test_images \
4          = train_images / 255.0, test_images / 255.0
5
```

Bug 2: Line 25. We need to use categorical cross entropy. Label have been converted into one hot encoding. Besides, binary cross entroy can only be used for binary classification.

```
1
2 model.compile(optimizer='adam',
3               loss=' categorical_crossentropy',
4               metrics=['accuracy'])
```

Bug 3: Line 20. Bob uses wrong activation function. We need to use softmax function instead of relu function. It is the last layer of MLP. We need to compute the probability of different classes.
Bug 4: Line 17. Removed input_shape for RGB images. The dataset is represented in RGB form which has 3 channel.

```
1
2 # Build the MLP model
3 model = models.Sequential([
4     layers.Flatten(input_shape=(32, 32, 3)),
5     layers.Dense(512, activation='relu'),
6     layers.Dense(256, activation='relu'),
7     layers.Dense(10, activation='softmax')
8 ])
```

```
1  import tensorflow as tf
2  from tensorflow.keras import layers, models
3  from tensorflow.keras.datasets import cifar10
4  from tensorflow.keras.utils import to_categorical
5
6  # Load and preprocess the CIFAR-10 dataset as int arrays
7  (train_images, train_labels), (test_images, test_labels) \
8          = cifar10.load_data()
9  # Normalize the images
10 train_images, test_images \
11         = train_images * 255.0, test_images * 255.0
12 train_labels, test_labels \
13         = to_categorical(train_labels), to_categorical(test_labels)
14
15 # Build the MLP model
16 model = models.Sequential([
```

```
    layers.Flatten(input_shape=(32, 32, 1)),
    layers.Dense(512, activation='relu'),
    layers.Dense(256, activation='relu'),
    layers.Dense(10, activation='relu')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=10, \
        batch_size=64, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc:.3f}')
```