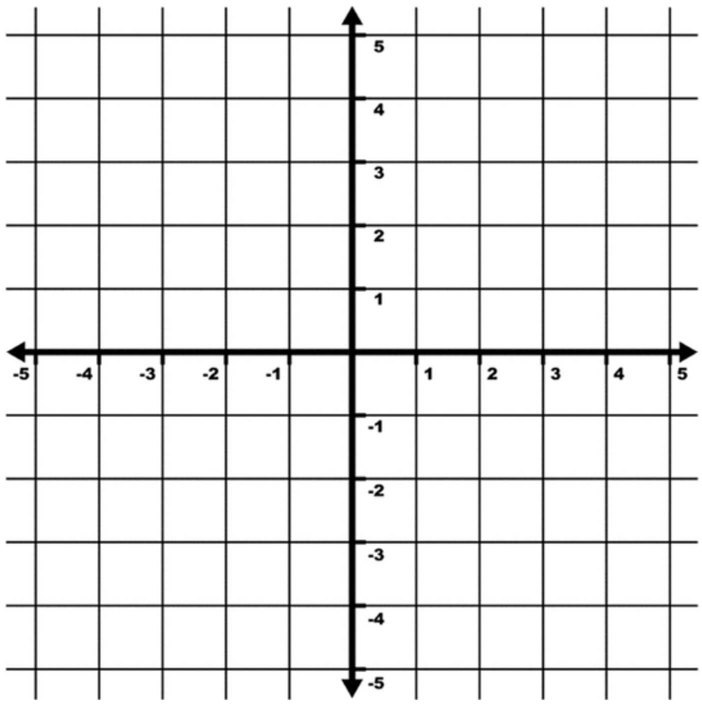
- Consider designing a one-layer Perceptron network to classify 4 classes. Assume that the data set includes 200 samples and each sample is 10 dimensional.
  What is the size of the weight matrix (bias should be included in the weight matrix)?

- Given a one node Perceptron with hard-limit activation function to classify the input in one of the possible two classes. Assuming that the input is a two dimensional vector and the weights and the bias are:

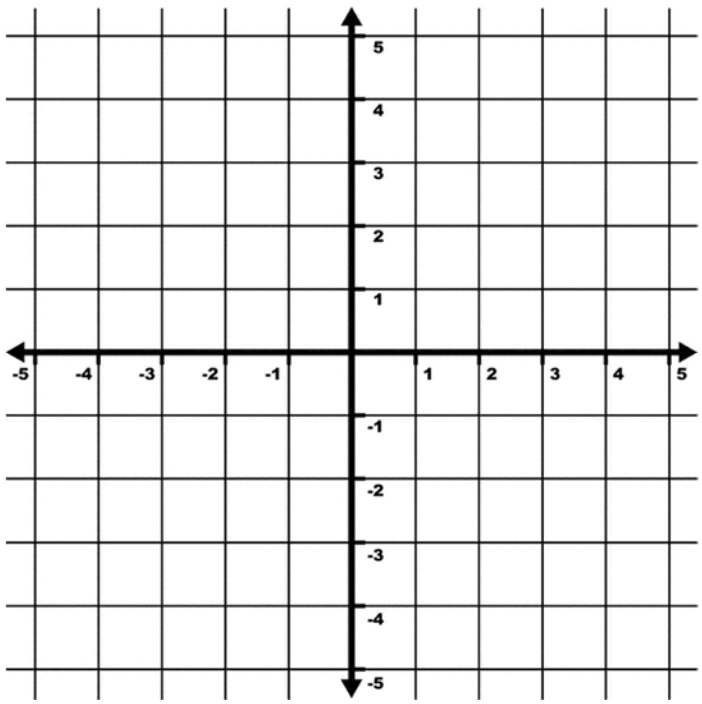
  $$\mathbf{w_1 = 3\ ; \quad w_2 = 2\ ; b = -6}$$

  Draw the boundary that separates the two classes.

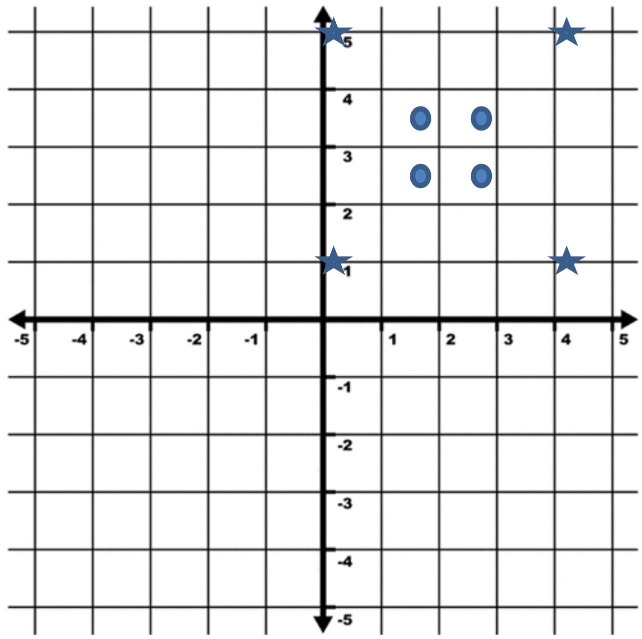


- Given a one-layer Perceptron with hard-limit activation function. The weight matrix which includes biases is shown below. Draw the boundary corresponding to each node and identify each region by its corresponding binary output code. Assume that bias is shown in the first column.

$$= \begin{bmatrix} -4 & -2 & 4 \\ -2 & 0 & 1 \\ 4 & 1 & 0 \\ 3 & 4 & 6 \end{bmatrix}$$

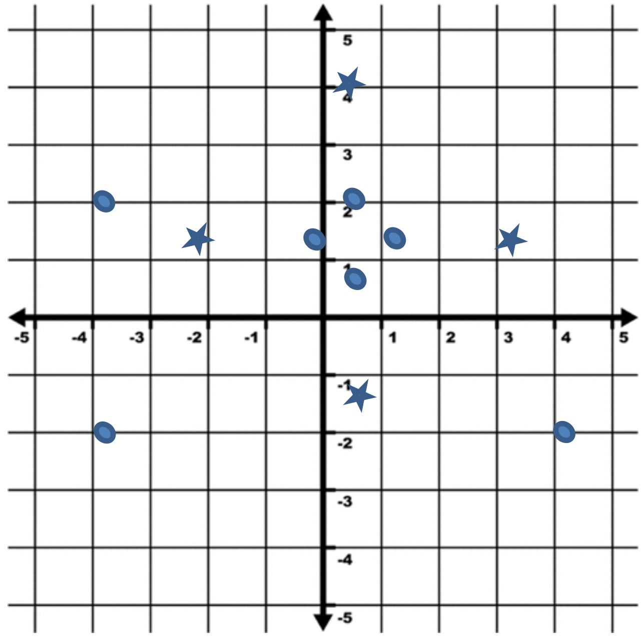


- Design a two-layer Perceptron neural network which will correctly classify the two classes (circles and stars) as shown below. Assume the activation (transfer) function for all the nodes are hard-limit with the output of 0 (star) and 1 (circle)



Show the weight matrices and biases for each layers.

- Design a multi-layer Perceptron neural network which will correctly classify the two classes (circles and stars) as shown below. Assume the activation (transfer) function for all the nodes are hardlimit with the output of 0 (star) and 1(Circle).



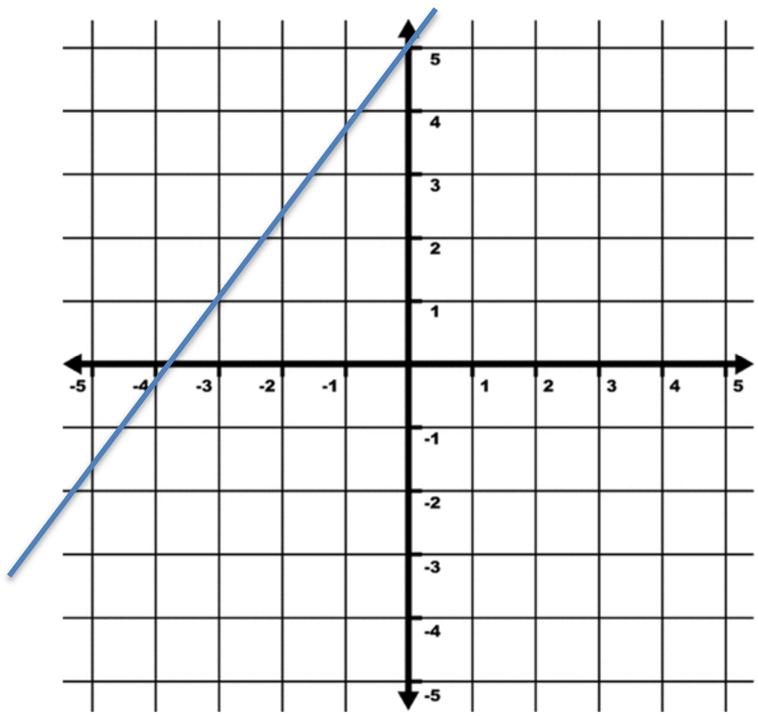
Show the weight matrices and biases for each layers.

- Consider the following training set for a Perceptron neural network.

$$\left\{p_1 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right\}, \left\{p_2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, t_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}, \left\{p_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\}, \left\{p_4 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, t_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}$$

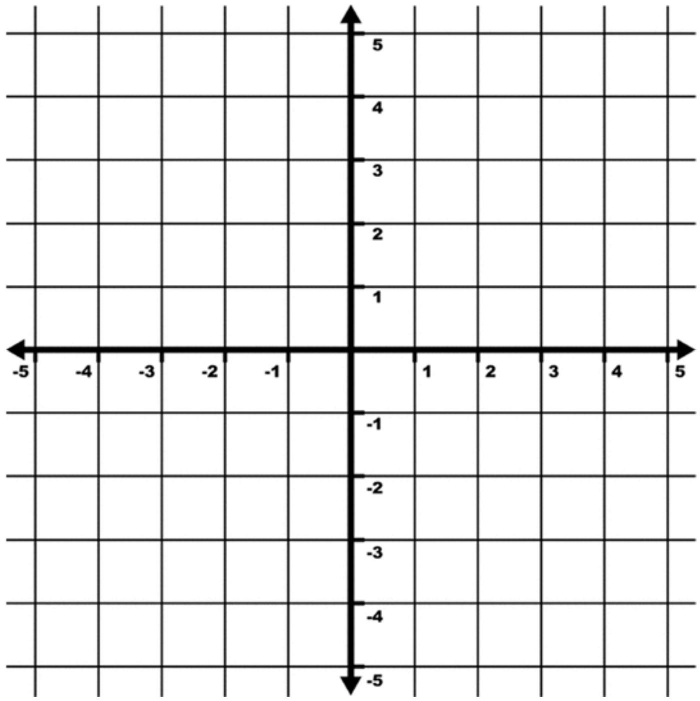
Design a **Perceptron** network with **two nodes** to solve this problem. i.e. find the weight matrix for this network. Bias should be included in the weight matrix.

- The boundary for a one node perceptron with two inputs is shown below. Find the numerical values of the weight matrix.
  Note: Bias should be included in the weight matrix.



- Given the weight matrix for a two-node Perceptron with hard-limit activation function, Draw the decision boundary for each node and label each region with the corresponding network output. Biases are included in the weight matrix

$$W = \begin{bmatrix} 3 & 4 & -2 \\ -1 & 5 & 3 \end{bmatrix}$$



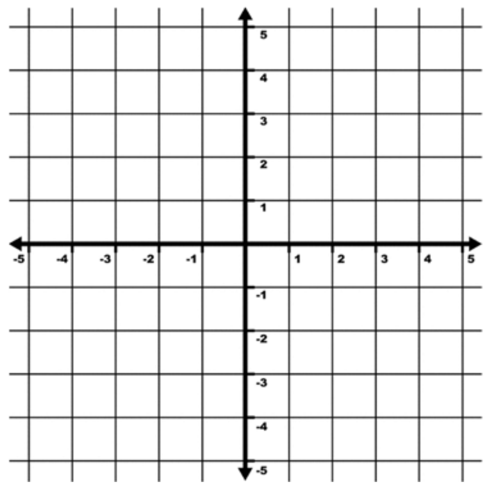
- Design a perceptron network to output a 0 (zero) when either of these two vectors are input to the network:

$$\left\{p_1 = \begin{bmatrix}1\\4\end{bmatrix}\right\}, \left\{p_2 = \begin{bmatrix}-3\\2\end{bmatrix}\right\}$$

and to output a 1 when either of the following vectors are input to the network:

$$\left\{p_1 = \begin{bmatrix}-3\\0\end{bmatrix}\right\}, \left\{p_2 = \begin{bmatrix}-1\\2\end{bmatrix}\right\}$$

a. Sketch a decision boundary for a network that will solve this problem.

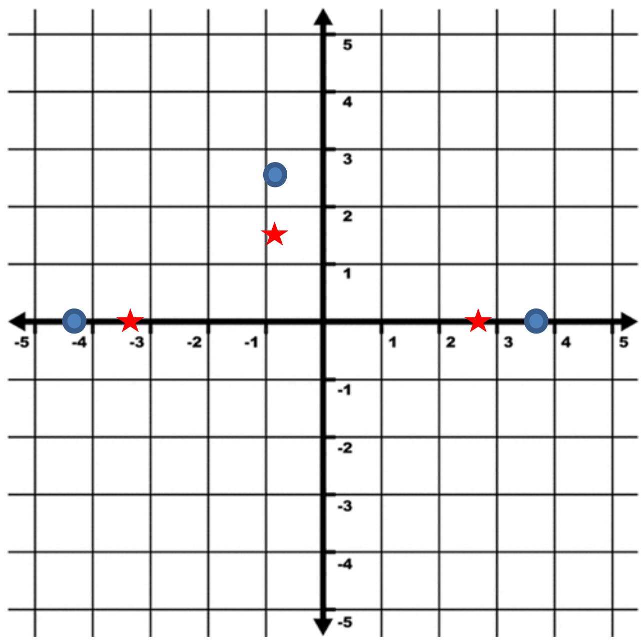


b. Find weights and biases that will produce the decision boundary in part a (Show the weight matrix. Include biases in the weight matrix

- Consider the following data where p is the input and t is the desired output

$$\left\{p_1 = \begin{bmatrix}4.5\\0\end{bmatrix}, t_1 = 0\right\}, \left\{p_2 = \begin{bmatrix}-3.5\\0\end{bmatrix}, t_2 = 0\right\}, \left\{p_3 = \begin{bmatrix}0\\2.5\end{bmatrix}, t_3 = 0\right\},$$
$$\left\{p_4 = \begin{bmatrix}3.5\\0\end{bmatrix}, t_4 = 1\right\}, \left\{p_5 = \begin{bmatrix}-2.5\\0\end{bmatrix}, t_5 = 1\right\}, \left\{p_6 = \begin{bmatrix}0\\1.5\end{bmatrix}, t_6 = 1\right\}$$

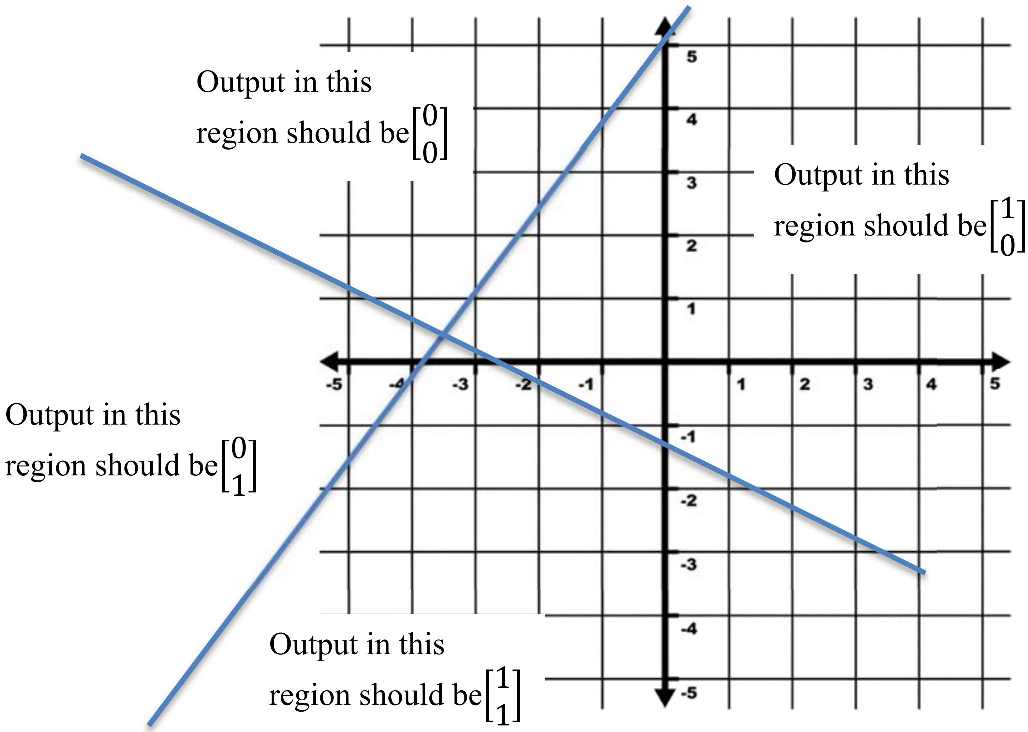
DESIGN a two-layer Perceptron neural network which will correctly classify the input data. Assume the activation (transfer) function for all the nodes are hardlimit with the output of 0 and 1
Show the weight matrices and biases for both layers. Biases should be included in the weight matrix in the first column.



Weight matrix for the first layer =

Weight matrix for the second layer =

- The boundaries for a two-node perceptron is shown below. The activations functions for both nodes are hard-limit with the output of 0 or 1. The input to this neural network is two dimensional. Find the numerical values of the weight matrix.
  Notes: Biases should be included in the weight matrix.
  Pay attention to the output regions.



The weight matrix =

- Consider the following training set for a Perceptron neural network.
  $\left\{p_1 = \begin{bmatrix}1\\3\end{bmatrix}, t_1 = \begin{bmatrix}0\\0\end{bmatrix}\right\}, \left\{p_2 = \begin{bmatrix}-1\\-1\end{bmatrix}, t_2 = \begin{bmatrix}0\\1\end{bmatrix}\right\}, \left\{p_3 = \begin{bmatrix}0\\4\end{bmatrix}, t_3 = \begin{bmatrix}1\\0\end{bmatrix}\right\}, \left\{p_4 = \begin{bmatrix}-1\\4\end{bmatrix}, t_4 = \begin{bmatrix}1\\1\end{bmatrix}\right\},$
  $\left\{p_5 = \begin{bmatrix}2\\-1\end{bmatrix}, t_5 = \begin{bmatrix}0\\1\end{bmatrix}\right\}$

  Design a **Perceptron** network with **two nodes** to solve this problem. i.e. find the weight matrix for this network. Bias should be included in the weight matrix.

Weight matrix including bias =

- Consider the following basis vectors in a 3-dimensional vector space. Find an orthogonal set using the Gram_Schmidt orthogonalization.

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, v_3 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

- Using reciprocal basis vectors, expand the vector $x = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$ in terms of the following basis set:

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad , \quad v_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

- Consider the following basis vectors in a 3-dimensional vector space. Find an orthogonal set using the Gram Schmidt orthogonalization.

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, v_3 = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}$$

- Consider the following basis vectors in a 4-dimensional vector space. Write a python program to find an orthogonal set using the Gram_Schmidt orthogonalization.

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 5 \end{bmatrix}, v_2 = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \end{bmatrix}, v_3 = \begin{bmatrix} 0 \\ 2 \\ 6 \\ 3 \end{bmatrix}, v_4 = \begin{bmatrix} 1 \\ 5 \\ 3 \\ 1 \end{bmatrix}$$

- Consider the set of all polynomials of degree 2 or less which represent a linear vector space. Given the polynomial $p = 8t^2 + 6t - 2$ and the basis set:

$$b_1 = 3t^2 + 2t - 2$$
$$b_2 = t^2 + 3$$
$$b_3 = t^2 + 5t - 6$$

a. Find the representation of the polygon p in terms of the given basis set.

b. If the basis set is changed to

$$b_1 = 2t^2 - 1$$
$$b_2 = t^2 + 3$$
$$b_3 = 5t - 6$$

Find the new representation of the polygon p in terms of the new basis set.

- Show that the set of 2 by 2 matrices is a linear vector space

- Consider a linear associator neural network with the following training set:

$$\left\{z_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right\} \left\{z_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\} \left\{z_1 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}$$

Using the Hebb rule, find the weight matrix

- Consider a neural network used for classification..

  If the output of the network is: $\begin{bmatrix} class\ 1\ score \\ class\ 2\ score \\ class\ 3\ score \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 3 \end{bmatrix}$

a. Using SVM loss, and assuming that the threshold (margin) is equal to 1, calculate the numerical value of the loss assuming that the target class is class 3:

Numerical value of the SVM loss is:

b. Using softmax and cross entropy loss, Calculate the numerical value of the loss assuming that the target class is class 3:

Numerical value of the cross entropy loss is:

- Consider a multi-layer neural network with four nodes at the last layer.. The output of this network for a given input is shown below:

$$\begin{bmatrix} 0.2 \\ 0.8 \\ 1.7 \\ -0.4 \end{bmatrix}$$

The desired output for this input is:

$$\begin{bmatrix} 0.1 \\ -0.3 \\ 0.8 \\ 0.7 \end{bmatrix}$$

a. Assuming that the threshold (margin) is equal to 1, can you calculate the SVM loss? If no explain why not. If yes, calculate the SVM loss.

b. Can you calculate the cross entropy loss? If no explain why not. If yes, calculate it and show the result.

- Complete the code for the following function (this is the same function in assignment 01)

```
import numpy as np
def calculate_percent_error(self,YA,YT):
""" Given a batch of input, actual outputs, and desired outputs, this function
calculates percent error. For each sample, if the actual output vector is not
exactly the same as the desired output, it is considered one error.
Percent error is 100*(number_of_errors/ number_of_samples)

:param YA: Array of actual outputs [number_of_nodes, ,n_samples]. Assume that
the each element of YA is either 0 or 1 (Result of hard-limit activation
function).
param YT: Array of desired (target) outputs [number_of_nodes ,n_samples]
Assume that all the values of array YT are zero except one of them which is
equal to 1.
return percent_error"""
```

- Consider the expression:

$$f(x, y, z) = max(x^2 y, z) + \text{xz}$$

Draw the computational graph for this expression and compute the numerical values of the partial derivatives with respect to x, y, and z given the inputs:

$$x = 2, \quad y = 3, \quad z = 5$$

$\frac{\delta f(x,y,z)}{\delta x} =$	$\frac{\delta f(x,y,z)}{\delta y} =$	$\frac{\delta f(x,y,z)}{\delta z} =$

- Write a program using Tensorflow to implement a neural network with one hidden layer and one output layer.

  The dimension of the input data data : **3**

  Number of nodes in hidden layer: **10**, activation: **sigmoid**

  Number of nodes in the output layer: **5** activation : **linear**

  Just implement the forward path to calculate and display output. **No training loop , or calculation of gradients.**

```
import tensorflow as tf
import numpy as np
input=np.array([[1,2,4]])
tf_input=tf.placeholder(tf.float32,shape=(None,3))




















with tf.Session() as sess:
    result = sess.run([output],feed_dict={tf_input:input})
    print(result)
```

- Consider the expression:

$$\boldsymbol{f(x,y)=\frac{1}{xy}+[\max(x\,,y))]^2}$$

Draw the computational graph for this expression and compute the partial derivatives with respect to x and y given the inputs:

$$\boldsymbol{x=-2,\quad y=3}$$

- Consider the expression:

$$\boldsymbol{f(x) = [10 - \max(x\,, y * (q + z))]^2}$$

Draw the computational graph for this expression and compute the partial derivatives with respect to x, y, q, and z via backpropagation given the input:

$$\boldsymbol{x = 1}, \quad \boldsymbol{y = 2}, \quad \boldsymbol{q = -1}, \quad \boldsymbol{z = 5}$$

- Consider the set of all 2x2 matrices. This set is a vector space, which we will call X If M is an element of this vector space, define the transformation A, such that $\boldsymbol{A(M) = M + M^T}$

  Consider the following basis set for the vector space X.

  $$\boldsymbol{v_1} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \boldsymbol{v_2} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \boldsymbol{v_3} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} \end{bmatrix}, \boldsymbol{v_4} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

  Find the matrix representation of the transformation $\boldsymbol{A}$ relative to the basis set $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}, \boldsymbol{v_4}\}$

- Consider the following performance surface:

$$\boldsymbol{F(X) = 4x_1^2 - 2x_2^2 + 3x_1x_2 - 2x_1 + 6}$$

Calculate the directional derivative of this function at point **(1,2)** in the direction of **(8,6)**

- Consider the following performance surface:

$$\boldsymbol{F(X) = x_1^2 x_2 x_3}$$

Find the second order Taylor series expansion of this function around point $P = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$

- Consider the following performance surface:

$$\boldsymbol{F(X) = 4x_1^3 + 6x_2^2x_3 + 3x_1x_2x_3}$$

Find the Hessian of this surface

- Consider the following performance surface:

$$\boldsymbol{F(X) = 2x_1^2 + 5x_2^2 - 3x_1x_2}$$

Assuming an initial point of **(1,2)** perform two steps of the steepest decent and show the result after each step. Assume learning rate $\boldsymbol{\alpha = 2}$

- Consider the following performance surface:

$$\boldsymbol{F(X) = 2x_1^2 - 5x_2^2 + 3x_1x_2 - 2x_1 + 6}$$

Calculate the directional derivative of this function at point (2,3) in the direction of (3,4)

- Consider the following performance surface:

$$\boldsymbol{F(X) = 4x_1^2 + 7x_2^2 + 4x_1x_2}$$

  - Find the stationary point of this surface
  - Determine if the stationary point is strong minima, weak minima, strong maxima, or weak maxima. Show your calculations and explain your conclusion.

- Consider the following performance surface:

$$\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{x}_1^2 + 4\boldsymbol{x}_2^2 - 2\boldsymbol{x}_1\boldsymbol{x}_2 + \boldsymbol{x}_1 - 10$$

  Assuming an initial point of **(2,1)** perform two steps of the steepest decent and show the result after each step. Assume learning rate $\boldsymbol{\alpha} = \mathbf{2}$

- Consider the following performance surface:

$$\boldsymbol{F(X) = 5x_1^4 - x_2^3 + 3x_2 - 5x_1 + 6}$$

Take two steps of the steepest descent algorithm, **minimizing along a line to calculate alpha**.
Use the following initial point: $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$
Show the resulting position after each step.
Hint: Direction along a line is in the direction of gradient

Position after the first step is:
Position after the second step is:

- Consider the following performance surface:

$$\boldsymbol{F(X) = 4x_1^2 - 2x_2^2 + 3x_1x_2 - 2x_1 + 6}$$

Calculate the first and second directional derivative of this function at point **(1,2)** in the direction of **(8,6)**

First directional derivative=

Second directional derivative=

- Consider the following performance surface

$$F(X) = 2x_1^2 - 6x_1x_2 + 5x_2^2 + 4x_1 + 3x_2$$

Given the initial point $\begin{bmatrix} -2 \\ 1 \end{bmatrix}$, take one step in the direction of $\begin{bmatrix} 5 \\ 1 \end{bmatrix}$ minimizing along a line to calculate alpha.

a. Show the resulting position after the step.

Position after the first step is:

b. Show that the gradient of $F(X)$ at the point after the first step is orthogonal to the direction along which the minimization occurred.

- Consider the following performance surface:

$$\boldsymbol{F(X) = 4x_1^2 + 7x_2^2 + 4x_1x_2}$$

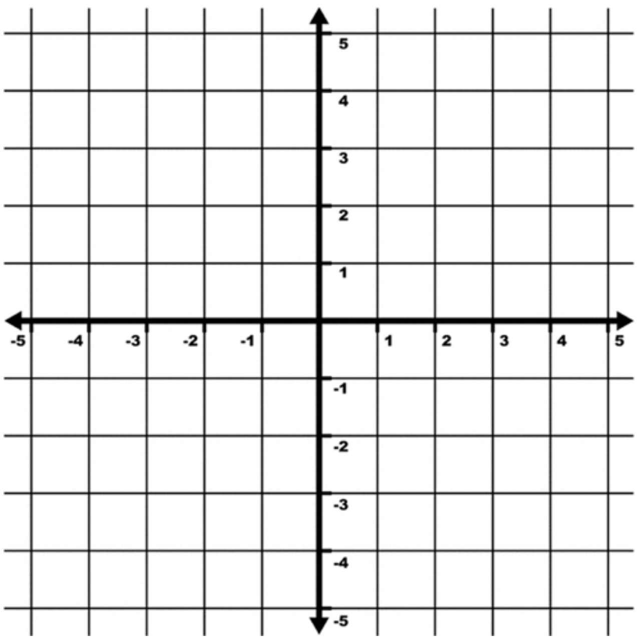
  - Find the stationary point of this surface
  - Determine if the stationary point is strong minima, weak minima, strong maxima, or weak maxima. Show your calculations and explain your conclusion.

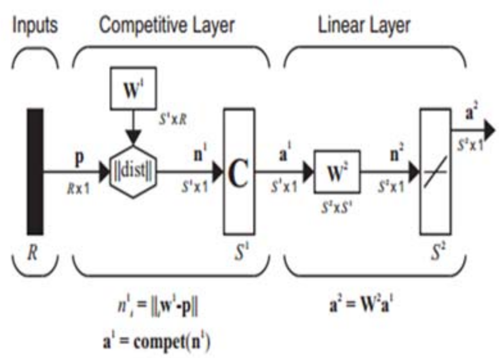
- Consider the LVQ network with the weight matrices as shown below:

Weight matrix for the first layer: $W^1 = \begin{bmatrix} 0 & 2 \\ 0 & 4 \\ 2 & 2 \\ -2 & 0 \end{bmatrix}$

Weight matrix for the second layer $W^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

Show the regions of the input space that make up each class.
Note: **YOU MUST** explicitly label each region according to **class**.





- Consider an LVQ network with the weight matrices as shown below:

Weight matrices: $w^1 = \begin{bmatrix} 0 & 2 \\ 0 & 4 \\ 2 & 2 \\ -2 & 0 \end{bmatrix}$ $\quad w^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

Consider the following inputs:

$\left\{p_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\}$; $\left\{p_2 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}, t_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}$

Assuming the LVQ training is used with $\boldsymbol{\alpha} = \mathbf{1}$

- Show the weight matrix after the first input, p_1 , is given:

- Show the weight matrix after the second input, p_2 , is given:

- Consider an RBF network with the following weights:

RBF layer weights: $w^1 = \begin{bmatrix} -2 & 1 \\ 2 & 0 \end{bmatrix}$ $b^1 = \begin{bmatrix} 0.5 \\ 2 \end{bmatrix}$

Linear layer weights: $w^2 = [10 \quad 1]$ $b^2 = 5$

- 1. What is the architecture of the network (number of inputs ?, Number of RBF nodes?, number of output nodes?) (2 points)

2. Given the following inputs, compute the output of the network. You may leave things un-simplified (no calculator required) (4 points each)

A grid is provided on the back in case you want to plot the data.

a) $p = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$

b) $p = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$

- Consider the following training set for an ADALINE neural network. Using LMS algorithm, find the equation of the performance function.

$$\left\{z_1 = \begin{bmatrix}2\\3\\1\end{bmatrix}, t_1 = 5\right\} \left\{z_2 = \begin{bmatrix}3\\4\\1\end{bmatrix}, t_2 = 2\right\}$$

Hint: performance function has the general form of:

$$F(x) = c + d^T x + \frac{1}{2} x^T A x$$

- Design an RBF network with two inputs and one output such that:

$$\left\{p_1 = \begin{bmatrix}0.1\\0.1\end{bmatrix}, t_1 = 2\right\}, \left\{p_2 = \begin{bmatrix}-0.1\\0.1\end{bmatrix}, t_2 = 2\right\}, \left\{p_3 = \begin{bmatrix}0\\0.1\end{bmatrix}, t_3 = 0\right\}$$

  Show the structure of your network (input, outputs, nodes in each layer) and numerical values of all the weight and biases for all the layers. (All numbers must be accurate to 3 digits after the decimal point).

- Consider the expression:

$$\boldsymbol{f(x,y,z) = min(xy,z) + \mathbf{xz}}$$

Use the computational graph for this expression to compute the numerical values of the partial derivatives with respect to x, y, and z given the inputs:

$$\boldsymbol{x = 2, \quad y = 7, \quad z = 20}$$

$\frac{\delta f(x,y,z)}{\delta x} =$	$\frac{\delta f(x,y,z)}{\delta y} =$	$\frac{\delta f(x,y,z)}{\delta z} =$

- Consider the following performance surface:

$$\boldsymbol{F(X) = 2x_1^2 - 5x_2^2 + 3x_1x_2 + 6}$$

Take one steps of the steepest descent algorithm, **minimizing along a line to calculate alpha**.
Use the following initial point: $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$
Show the resulting position after one step.

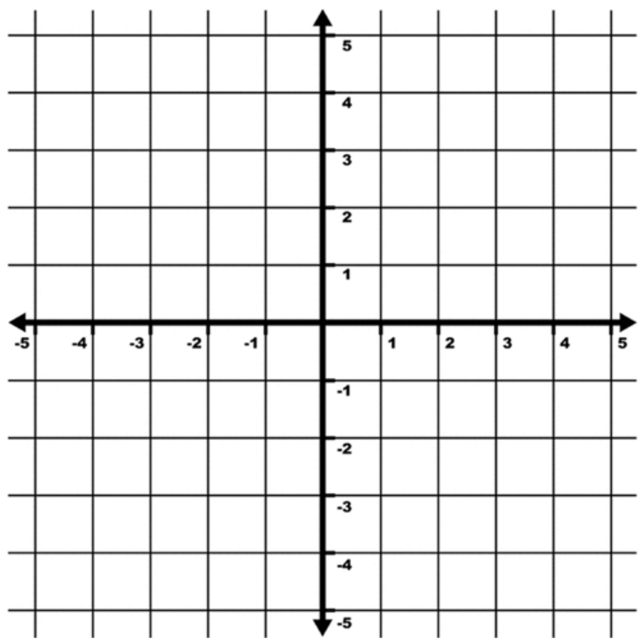
Position after one step is:

- Consider the LVQ network with the weight matrices as shown below:

Weight matrix for the first layer: $W^1 = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ -1 & 1 \\ 1 & 3 \end{bmatrix}$

Weight matrix for the second layer $W^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Show the regions of the input space that make up each class.
Note: **YOU MUST** explicitly label each region according to **class**.



- Consider the following performance surface:

$$\boldsymbol{F(X) = 5x_1^4 - x_2^3 + 3x_2 - 5x_1 + 6}$$

Take two steps of the steepest descent algorithm, **minimizing along a line to calculate alpha**.
Use the following initial point: $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$
Show the resulting position after each step.

- Complete the following function. Assume this function will be called by the main program shown below.

```
import numpy as np
import tensorflow.keras as keras
def increment_biases(model, layer_number=None, layer_name=None, incr=1.0):
"""This function increments all the biases for a layer by incr (if biases
exist)
:param model: keras model
:param layer_number: Layer number starting from layer 0
:param layer_name: Layer name (if both layer_number and layer_name are
specified, layer number takes precedence).
:param incr: The increment value
:return: new biases for the given layer (If the given layer does not have
bias then None should be returned)"""


















my_model = keras.applications.VGG16(weights='imagenet', include_top=True)
for k in range(23):
    print(increment_biases(model=my_model,layer_number=k))
print(get_biases(model=my_model,layer_name="fc1"))
```

- Complete the following code. The code should compute the output, loss, and gradients, and perform a single weight update of a neural network with **784 inputs**, **100 nodes with sigmoid activation in the first layer, and 10 output nodes with linear activation**.
  Assume you have a batch of inputs called **X** that has dimensions **[16, 784]**, and a batch of targets called **y** that has dimensions **[16, 1]**

```
import tensorflow as tf
import numpy as np
def my_loss(y, y_hat):
    return tf.reduce_mean(
           tf.nn.sparse_softmax_cross_entropy_with_logits(
           labels=y,logits=y_hat))
# Create random weights and biases
W_1 = tf.Variable(np.random.randn(                 ), trainable=True)
b_1 = tf.Variable(np.random.randn(                 ), trainable=True)
W_2 = tf.Variable(np.random.randn(                 ), trainable=True)
b_2 = tf.Variable(np.random.randn(                 ), trainable=True)
with tf.GradientTape(persistent=True) as tape:
    # Calculate output
    ______________________________________________________________
    ______________________________________________________________
    ______________________________________________________________
    ______________________________________________________________
    ______________________________________________________________
    ______________________________________________________________
    output=
    # Calculate loss
    ______________________________________________________________
    ______________________________________________________________
    ______________________________________________________________
    loss
    # Calculate gradients
    ______________________________________________________________
    dW_1, db_1 =
    ______________________________________________________________
    dW_2, db_2 =
# Calculate new values of weights and biases
______________________________________________________________
______________________________________________________________
______________________________________________________________
______________________________________________________________
______________________________________________________________
______________________________________________________________
______________________________________________________________
```

- Complete the following function. Assume this function will be called by the main program shown below.

```
import numpy as np
import tensorflow.keras as keras
def get_biases(model, layer_number=None, layer_name=None):
"""This function returns the biases for a layer.
:param model: keras model
:param layer_number: Layer number starting from layer 0
:param layer_name: Layer name (if both layer_number and layer_name are specified, layer number takes precedence).
:return: biases for the given layer (If the given layer does not have bias then None should be returned)"""

















my_model = keras.applications.VGG16(weights='imagenet', include_top=True)
for k in range(23):
    print(get_biases(model=my_model,layer_number=k))
print(get_biases(model=my_model,layer_name="fc1"))
```

- Consider a single neuron with linear activation : $\boldsymbol{output = Wx + b}$

  Write a Python function, using **Tensorflow** to adjust the weights and biases (one step) and return the gradients of loss with respect to $\boldsymbol{W}$ and gradients of loss with respect to $\boldsymbol{b}$

  Notes:

  Assume loss is the defined as the **MSE**

  You must use Tensorflow without using Keras

```
import tensorflow as tf
import numpy as np
def calculate_loss(x,y,w,b):
    # x: input batch (input_dimensions, batch_size)
    # y: desired output (output_dimensions, batch_size)
    # w: weight matrix (input_dimensions, output_dimensions)
    # b: bias (1, output_dimensions)
```

- Consider a convolutional neural network.
  Note: **DO NOT consider biases**.

Input layer:
Input to this CNN are color images of size 64x64x3 with the batch size = 30

Next layer is Conv2D layer:
number of filters: 100, filter size: 7x7 ; stride: 1x1 ; padding: "same"

What is the shape of the weight matrix (tensor) for this layer? __________________

What is the shape of the output (tensor) of this layer? __________________

Next layer is MaxPool2D:
pool size: 2x2 strides: 2x2 padding: "valid"

What is the shape of the output (tensor) for this layer? __________________

Next layer is Flatten layer:

What is the shape of the output (tensor) for this layer? __________________

Next layer is Dense layer:
number of nodes: 400

What is the shape of the weight matrix (tensor) for this layer? __________________

What is the shape of the output (tensor) for this layer? __________________

- Consider a convolutional neural network. The dimensions of layer K are given as 63x63x32.

  Layer K+1 is a convolutional layer with 16 filters of size 5 by 5 and stride=2 and padding =1

  a. Find the dimensions of the layer K+1

  b. How many distinct weights is shared between the filters in layer K+1. Assume there is no bias.

- Complete the code for the following method/function (this is the same function in assignment 04).

  Assume that the object that you are working with, i.e. self, is the same as the object that you used in your assignment 04. For example, if you used a Keras model in your assignment 04 then assume your object (self) is a Keras model.

  Hint: you may use `layers[]` and `get_layer()` and `get_wights()` property and methods

```
Import numpy as np
def get_biases(self,biases,layer_number=None,layer_name=""):
""" This function gets the biases for the layer specified by layer_number or by
layer_name.
:param layer_number: Layer number starting from layer 0.
:param layer_name: Layer name (if both layer_number and layer_name are
specified, layer number takes precedence).
:return: biases for the given layer (If the given layer does not have bias then
None should be returned) """
```

- Assuming that the actual output and the desired output of a neural network are given. Complete the code for the following function to calculate the overall cross entropy loss for a batch of data.y_hat is a matrix presenting the actual output. Each row of this matrix is the actual output of the neural network for an input sample. The number of rows in this matrix is equal to the number of samples in the input batch. y is the desired output. Each row is a one-hot representation of the desired class. This means that all entries in each row are zeros except one of them which is equal to 1 indicating the correct class.
  Notes:
  Do NOT use TensorFlow or Keras.
  You may use the numpy helper functions `np.nonzero()` and `np.log()` and `np.sum()`

```
def calculate_overall_cross_entropy_loss(y_hat,y):
Import numpy as np
""" This function calculates the overall cross entropy loss.
layer numbers start from zero.
:param y_hat: actual output [number_of_samples,number_of_classes].
:param Y: Desired output [number_of_samples,number_of_classes].
:return: overall cross-entropy loss
"""
```

- Consider a convolutional neural network.
  Note: **DO NOT consider biases**.

Input layer:

Input to this CNN are color images of size 64x64x3 with the batch size = 30

Next layer is Conv2D layer:

number of filters: 100, filter size: 7x7 ; stride: 3x3 ; padding: 3

What is the shape of the weight matrix (tensor) for this layer? __________________

What is the shape of the output (tensor) of this layer? __________________

Next layer is MaxPool2D:

pool size: 2x2 strides: 2x2 padding: 0 (Valid)

What is the shape of the output (tensor) for this layer? __________________

Next layer is Flatten layer:

What is the shape of the output (tensor) for this layer? __________________

Next layer is Dense layer:

number of nodes: 300

What is the shape of the weight matrix (tensor) for this layer? __________________

What is the shape of the output (tensor) for this layer? __________________

- Consider a convolutional neural network.
  Input to this CNN are color images of size 65x65x3.
  Batch size =100
  Notes:
  DO NOT consider biases.

First layer: filter size 11x11 ; stride: 2x2 ; padding: 0, number of filters: 18

What is the shape of the weight matrix (tensor) for the first layer? ___________________

What is the shape of the output (tensor) of the first layer? ________________________

Second layer: filter size 3x3 ; stride: 3x3 ; padding: 1, number of filters: 50

What is the shape of the weight matrix (tensor) for the second layer? ________________

What is the shape of the output (tensor) of the second layer? __________________

max_pool after the second layer: size: 4x4 strides: 2x2 padding: 0

What is the shape of the output (tensor) after max_pool layer? ___________________

Third layer: FC (fully connected) number of nodes=10.

What is the shape of the weight matrix (tensor) for the third layer? _________________

- Complete the following function. Assume this function will be called by the main program shown below.

```
import numpy as np
import tensorflow.keras as keras
def get_biases(model, layer_number=None, layer_name=None):
"""This function returns the biases for a layer.
:param model: keras model
:param layer_number: Layer number starting from layer 0
:param layer_name: Layer name (if both layer_number and layer_name are specified, layer number takes precedence).
:return: biases for the given layer (If the given layer does not have bias then None should be returned)"""



















my_model = keras.applications.VGG16(weights='imagenet', include_top=True)
for k in range(23):
    print(get_biases(model=my_model,layer_number=k))
print(get_biases(model=my_model,layer_name="fc1"))
```

- Complete the following code. The code should compute the output, loss, and gradients, and perform a single weight update of a neural network with 784 inputs, 100 nodes with sigmoid activation in the first layer, and 10 output nodes with linear activation.
  Assume you have a batch of inputs called **X** that has dimensions **[16, 784]**, and a batch of targets called **y** that has dimensions **[16, 1]**

```
import tensorflow as tf
import numpy as np
def my_loss(y, y_hat):
    return tf.reduce_mean(
           tf.nn.sparse_softmax_cross_entropy_with_logits(
           labels=y,logits=y_hat))
# Create random weights and biases
W_1 = tf.Variable(np.random.randn(                 ), trainable=True)
b_1 = tf.Variable(np.random.randn(                 ), trainable=True)
W_2 = tf.Variable(np.random.randn(                 ), trainable=True)
b_2 = tf.Variable(np.random.randn(                 ), trainable=True)
with tf.GradientTape(persistent=True) as tape:
    # Calculate output
    ________________________________________________________________
    ________________________________________________________________
    ________________________________________________________________
    ________________________________________________________________
    ________________________________________________________________
    ________________________________________________________________
    output=
    # Calculate loss
    ________________________________________________________________
    ________________________________________________________________
    ________________________________________________________________
    loss
    # Calculate gradients
    ________________________________________________________________
    dW_1, db_1 =
    ________________________________________________________________
    dW_2, db_2 =
# Calculate new values of weights and biases
________________________________________________________________
________________________________________________________________
________________________________________________________________
________________________________________________________________
________________________________________________________________
________________________________________________________________
________________________________________________________________
```

- Consider the expression: $\boldsymbol{f(x, y) = x^2 + y}$
  Given the inputs $\boldsymbol{x = 2}$, $\boldsymbol{y = 5}$, write a Python program, using Tensorflow to print the value of the output $\boldsymbol{f(x, y)}$ and partial derivatives of the $\boldsymbol{f(x, y)}$ with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$

```
import tensorflow as tf
import numpy as np
```

- Consider a convolutional neural network.
  Note: **DO NOT consider biases**.

Input to this CNN are color images of size 32x32x3.
Batch size =10

First layer: filter size 5x5 ; stride: 1x1 ; padding: “SAME”, number of filters: 50

What is the shape of the weight matrix (tensor) for the first layer? ____________

What is the shape of the output (tensor) of the first layer? ______________

max_pool after the first layer: size: 2x2 strides: 2x2 padding: “SAME”

What is the shape of the output (tensor) after max_pool layer? _______________

Second layer: filter size 3x3 ; stride: 1x1 ; padding: “SAME”, number of filters: 64

What is the shape of the weight matrix (tensor) for the second layer? _________

What is the shape of the output (tensor) of the second layer? ______________

- Write a program using Tensorflow to implement a neural network with one hidden layer and one output layer.

  The dimension of the input data data : **3**

  Number of nodes in hidden layer: **10**, activation: **sigmoid**

  Number of nodes in the output layer: **5** activation : **linear**

  Just implement the forward path to calculate and display output. **No training loop , or calculation of gradients.**