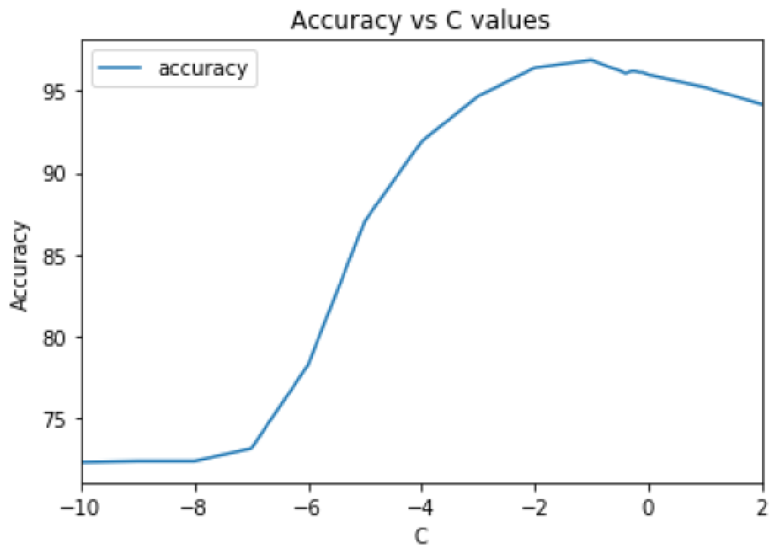
REPORT

PART 1: REGULARIZATION

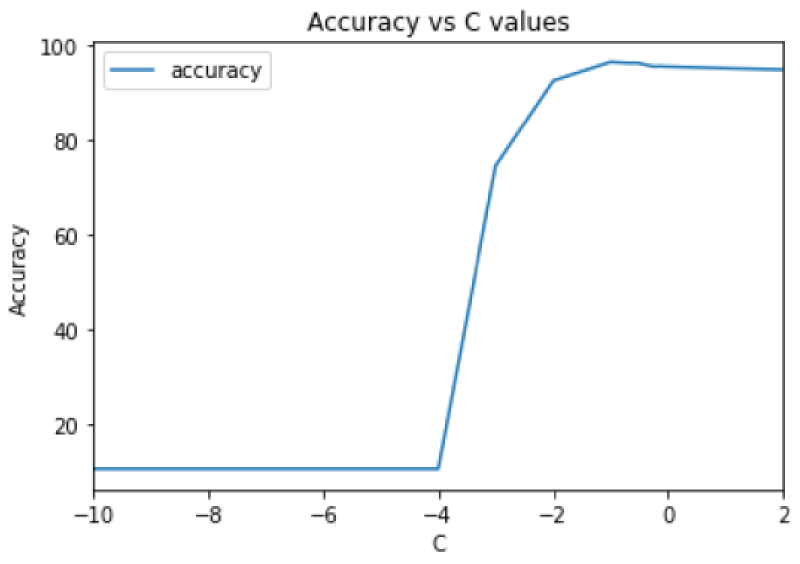
After loading the DIGITS data to variable “digits” and I divided it into test and train sets in 20:80 ratio.

Trained the data on Logistic Regression on “l1” and “l2” norm penalties for a range of c values (alphas) to see where I get highest accuracy. Cross Validation through Stratified K Fold, splitting the data 5 times. Result:

- **“l2” penalty gave 96.868% accuracy at value of c = -1.00**



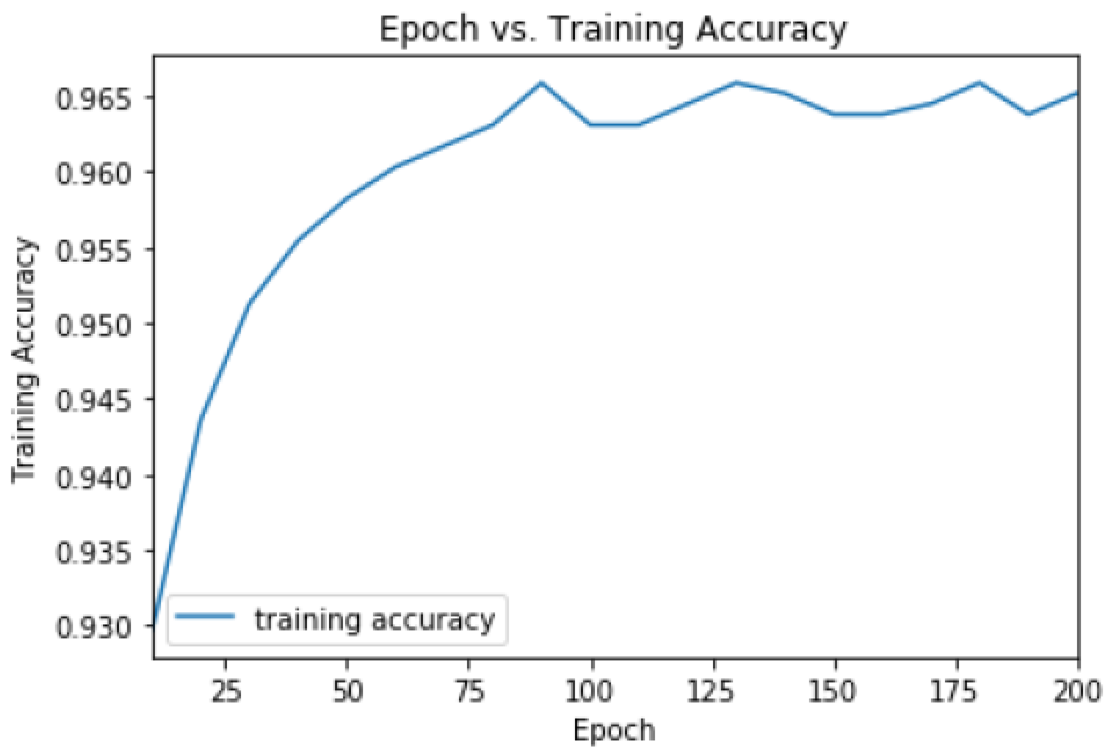
- **“l1” penalty gave 96.524% accuracy at value of c = -1.00**



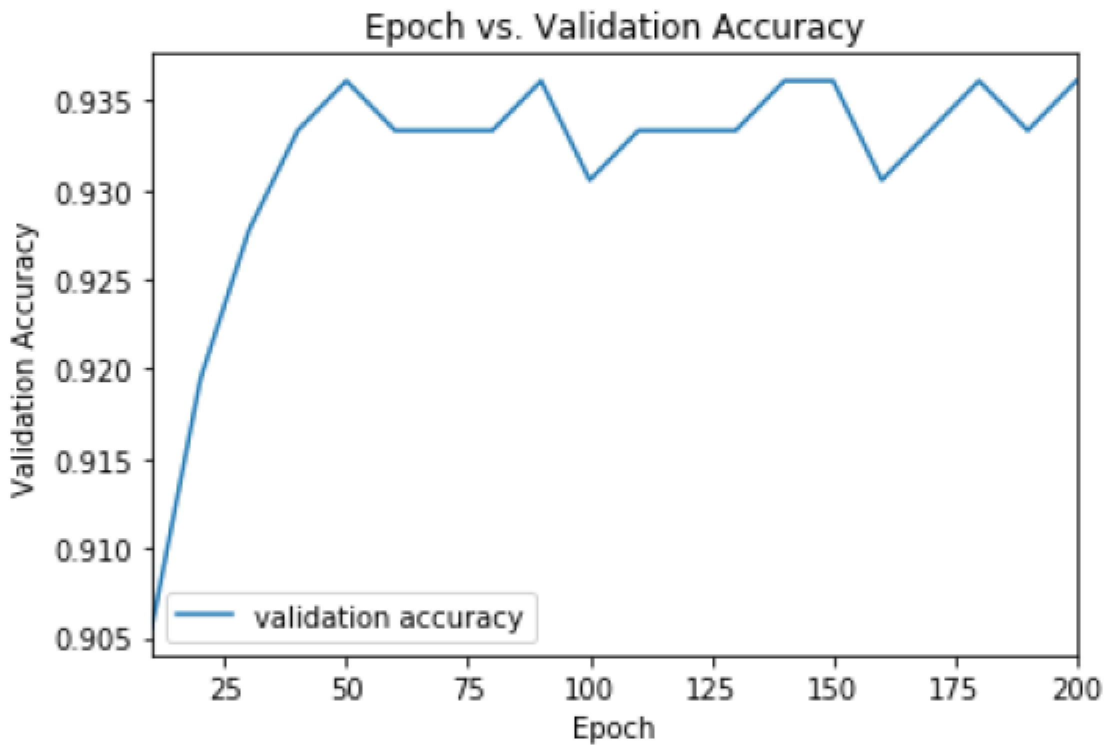
PART 2: NEURAL NETWORKS: Q1

After I set the input and output placeholders, with 64(8*8) and 10 inputs and outputs respectively, I set the weights (64) and the biases(10). I then defined the loss function and did softmax regression. Then I performed Cross Entropy to reduce loss function. I then defined the optimizer with learning rate = 0.001. I defined a function to make OneHotEncoder for training and testing data. I then defined a function to make mini batches of the training data. Then I trained the data on mini batch gradient descent with each batch being 100 data units for 200 epochs. Then I defined lists and eventually graphs for Epoch vs Training Accuracy and Epoch vs Validation Accuracy.

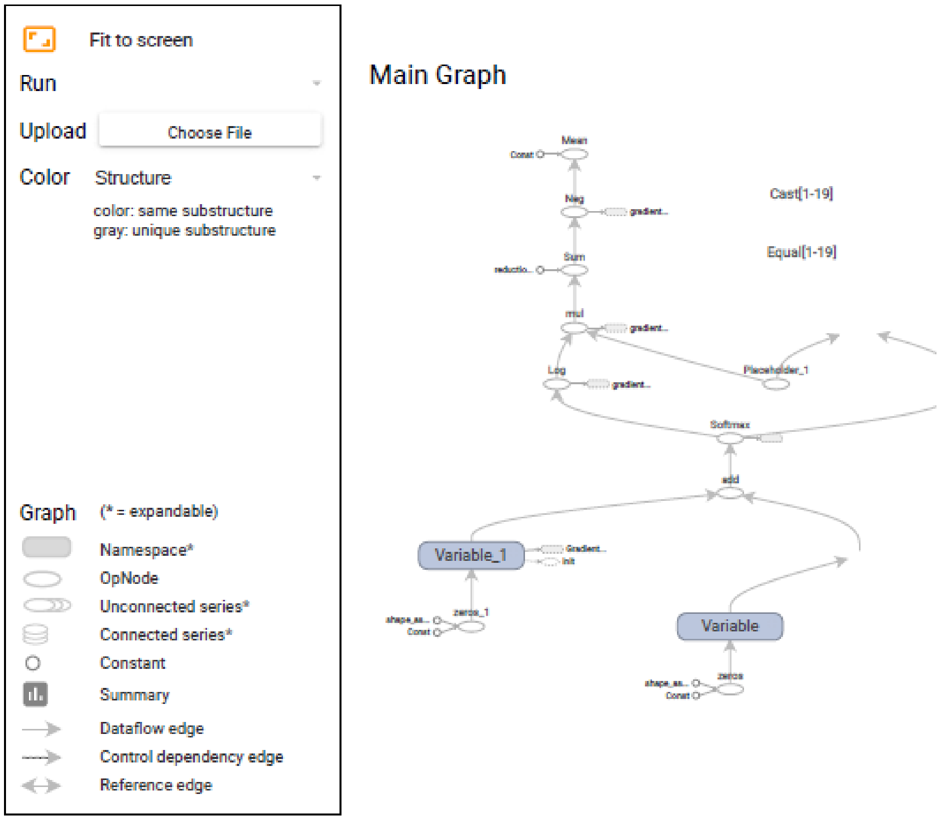
TRAINING ACCURACY: 96.52%



VALIDATION ACCURACY: 93.61%



- I then displayed the Tensorboard.



PART 2: NEURAL NETWORKS: Q2

Then I set the input and output placeholders, with 64(8*8) and 10 inputs and outputs respectively.

I also defined the hyperparameters for the hidden layers.

I then defined a function to take in value of neurons to define weights(664) and biases(610). The function also included dropout value = 10% and the activation function provision. Hence, also the loss function.

I then defined a DNN with input, output and the three hidden layers through the function defined before.

I then did softmax regression.

Then I performed Cross Entropy to reduce loss function.

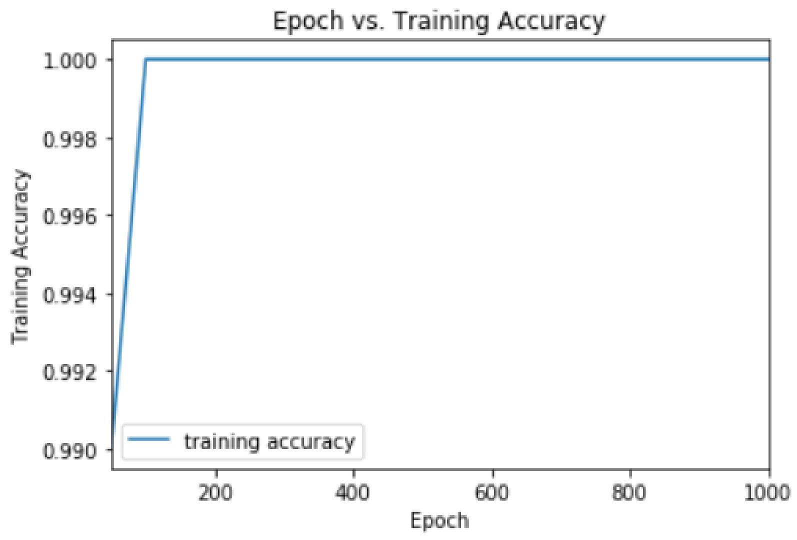
I then defined the optimizer with learning rate = 0.001

I then defined a function to make mini batches of the training data.

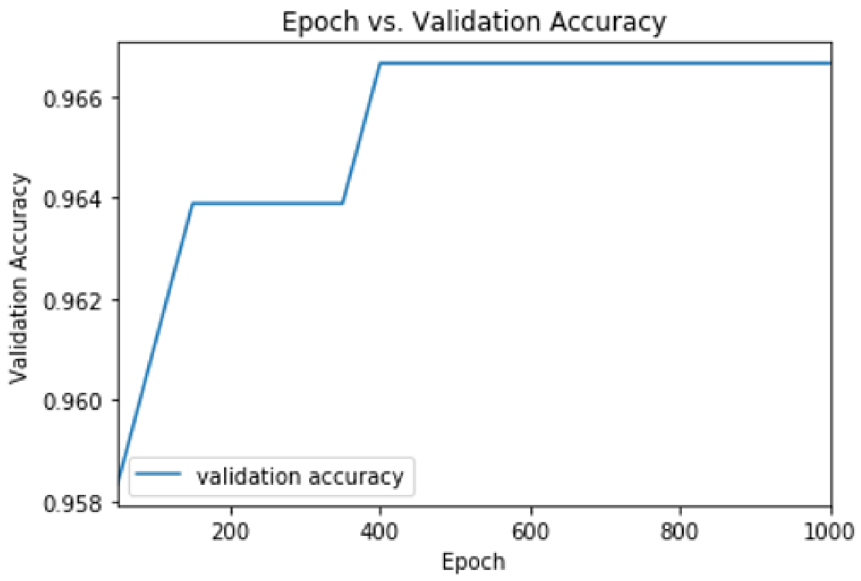
Then I trained the data on mini batch gradient descent with each batch being 100 data units for 1000 epochs.

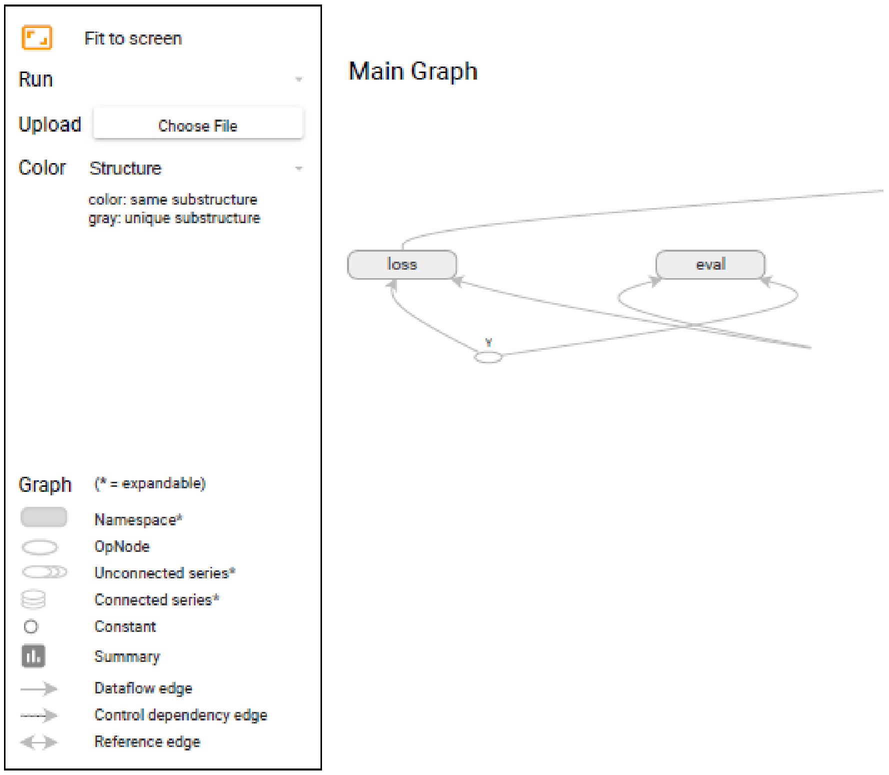
Then I defined lists and eventually graphs for Epoch vs Training Accuracy and Epoch vs Validation Accuracy.

TRAINING ACCURACY: 100%



VALIDATION ACCURACY: 95.5%




Fit to screen
Run
Upload
Choose File
Color
Structure
color: same substructure
gray: unique substructure
Graph
(* = expandable)
Namespace*
OpNode
Unconnected series*
Connected series*
Constant
Summary
Dataflow edge
Control dependency edge
Reference edge
Main Graph
loss
eval
y

Part 3
[Extra Credits]

Question 1. Neuron Saturation slows down the training of neural networks, batch normalization is a method of preventing neuron saturation by scaling and centering the inputs to each layer in a neural network. Explain in 100 words what Neuron Saturation is and why it slows down the training process. Explain in 500 words how Batch Normalization is done. (20p)

When using the tanh activation function, if the hidden nodes have values close to -1.0 or 1.0, i.e., extremely high or extremely low, and the output nodes have values close to 0.0 or 1.0, is called a saturated neural network and the phenomenon is called Neuron saturation.

A saturated neuron in the output layer makes the entire network useless. The biggest problem with saturation is the gradient falling to 0. Since there should be a non-zero gradient for the weights to be updated, the gradient falling to 0 causes the weights to change very slowly with the iterations. Hence the network stops learning and the training process slows down. Also, the saturated models are often overfitted.

Batch Normalization –

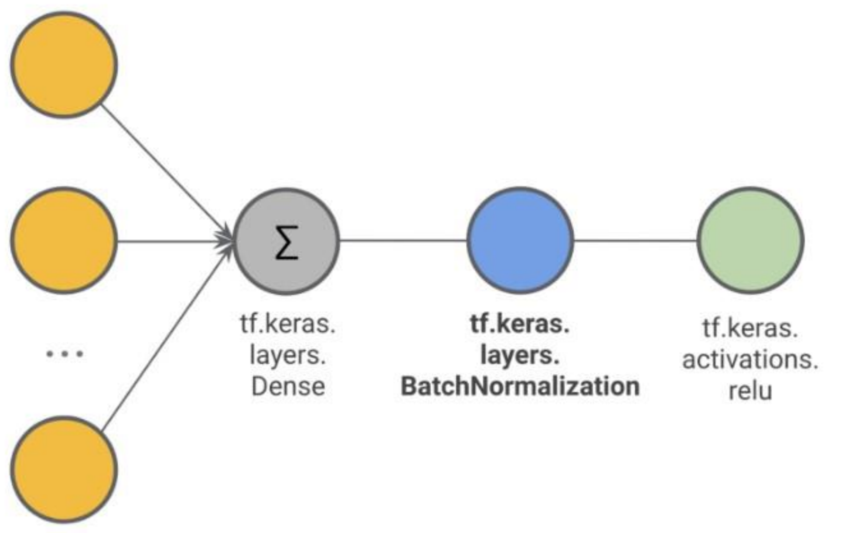
It is a method of preventing neuron saturation by scaling and centring the inputs to each layer in a neural network. It is a simple and an effective way to improve the performance of a neural network. The idea behind Batch Normalization is that neural networks tend to learn better when their input features are uncorrelated with zero mean and unit variance. As each layer within a neural network takes the activations of the previous layer as inputs, the same idea can be applied to each layer. Batch normalization does exactly this by normalizing the activations over the current batch in each hidden layer, generally right before the non-linearity.

Basically, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

Batch normalizing can be done in three ways in TensorFlow using,

- tf.keras.layers.BatchNormalization
- tf.layers.batch_normalization
- tf.nn.batch_normalization

In TensorFlow, batch normalization is implemented as an additional layer.


Σ
...
tf.keras. layers. Dense
tf.keras. layers. BatchNormalization
tf.keras. activations. relu

The code for applying Batch normalization is as follows-

- ⇨ in_training_mode = tf.placeholder(tf.bool)
- ⇨ hidden = tf.keras.layers.Dense(n_units,   activation=None)(X)
- ⇨ batch_normed = tf.keras.layers.BatchNormalization()(hidden, training = in_training_mode)
- ⇨ output = tf.keras.activations\.relu(batch_normed)

The training variable in the Batch Normalization function is required because Batch Normalization operates differently during training vs. the application stage– during training the z score is computed using the batch mean and variance, while in inference, it's computed using a mean and variance estimated from the entire training set.

Question 2: Why do we use activation functions in Neural Networks? List three activation functions used in Neural Networks and derive their first order derivatives. Explain the difference among these functions. (20p)

Activation Function -

In an artificial neural network (ANN), given a set of inputs, the activation function of a neuron defines the output of that neuron, which is then used as a input in the next layer in the stack. These functions are really important to learn and make sense of a complicated and non-linear complex functional mappings between the inputs and response variable.

We use activation function in neural networks because they enables the model to learn complex functional mappings from the data. In fact, a neural network without an Activation function would be nothing more than a Linear regression model with limited power. The activation function introduces non-linear properties into the network and enables it to deal with much more complicated data like images, videos , speech, etc.

Three activation functions used in Neural Networks are –

- ReLu – Rectified Linear Units
- Tanh – hyperbolic tangent
- Sigmoid or Logistic

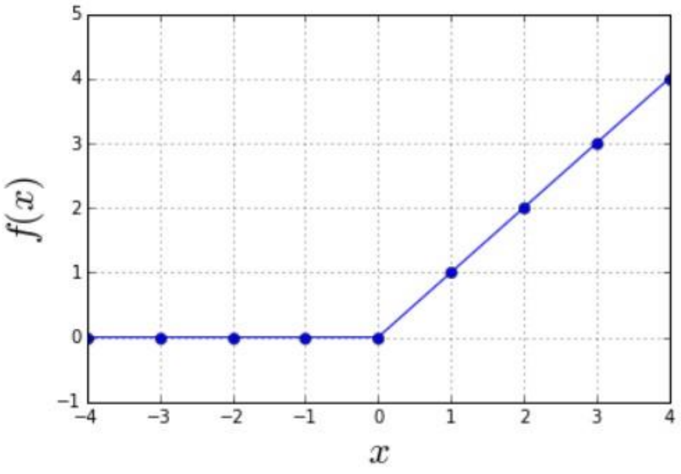
ReLu –

The ReLu activation function is basically, max(0, x), i.e., it sets anything less than or equal to 0 (negative numbers) to be 0 and keeps all the same values for any values > 0.

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

where x is the input to a neuron.

For the first order derivative of the ReLu function, we need to find the slope of the graph of the ReLu function at x < 0 and at x ≥ 0.



Derivative is the slope of the graph at a certain point. Visually looking at the graph, we can conclude that for every negative value of x, the slope of the graph is 0, and for every value of x such that x ≥ 0, the slope of the graph is 1. Hence, the first derivative of ReLu is,

$$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Tanh –

It is a hyperbolic function with a range from -1 to +1. Its mathematical formula is,

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

First order derivative of tanh function-

⇨ $f'(x) = \frac{dy}{dx} \tanh(x)$

⇨ $f'(x) = \frac{dy}{dx} \frac{\sinh(x)}{\cosh(x)}$

$dydy\ \cosh(x)\ .\qquad \sinh(x)\ -$

⇨ $f'(x) = \frac{\sinh(x)\ .\ \_\cosh(x) \qquad \_\_}{dx \qquad cosh2(x)\ dx}$

⇨ $f'(x) = \frac{\cosh(x)\,.\,\cosh(x) - \sinh(x)\,.\sinh(x)}{cos}$

$h^2(x)$

⇨ $f'(x) = cos\_\_\_\_h2(cosx)\ h{-2}\ \ (sinx)h^2(x)$

⇨ $f'(x) = 1 - tanh^2(x)$

⇨ $f'(x) = 1 - f^2(x)$

Sigmoid –

It is a S-shaped curve. It exists between 0 to 1 and hence is used for models where we have to predict a probability as the output. Mathematically, its of the form,

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

First order derivative of the Sigmoid function –

$\Rightarrow f'(x) = {}^{dy}\_\ \frac{1}{1+e^{-}}{}_{dx} \quad x$

$\Rightarrow f'(x) = \frac{(1+e^{-x}) \,.\, \frac{dy}{}(1) \;-\; (1) \,.\, \frac{dy}{}(1+e^{-x})}{(1+e^{-x})^2}{}_{dx \quad dx}$

$\Rightarrow f'(x) = \frac{(1+e^{-x}) \,.\, (0) \;-\; (1) \,.\, (-e^{-x})}{(1+e^{-x})^2}$

$\Rightarrow f'(x) = (\_\_\_\_ 1{+}e\ e{-}{-}xx)2$

$'(x) = \_\_\_\_\ 1{-}1{+}e{-}x$

$\Rightarrow f$

$(1+e^{-x})^2$

$\Rightarrow f'(x) = \quad 1+ \qquad 1 \quad \_\_\_\_ \quad \_\_\_\_\ e{-}x\ -$

$(1+e^{-x})^2 \qquad (1+e^{-x})^2$

$\Rightarrow f'(x) = (\_\_\_\_ \quad \_\_\_\_ 1{+}\ 1e{-}x) - (1{+}\ e1{-}x)2$

$\Rightarrow f'(x) = (\_\_\_\_ 1{+}\ {}^{1}e{-}x) \,.\, \left(1 - \frac{1}{(1+e^{-x})}\right)$

$\Rightarrow f'(x) = f(x) \,.\, (1 - f(x))$

Difference Between ReLu, tanh and sigmoid –

- ReLu gives the maximum of the two values, i.e., max(0, x)

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

  Range = [0, ∞]

- Tanh is a hyperbolic function,

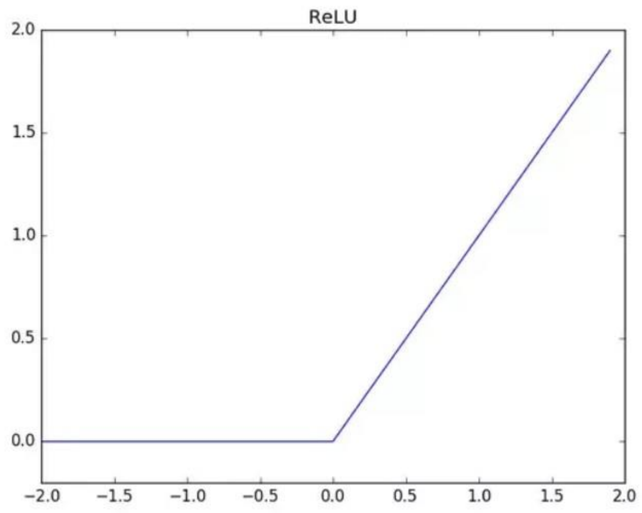
$$f(x) = \tanh(x) = \frac{e_x - e_{-x}}{e_x + e_{-x}}$$

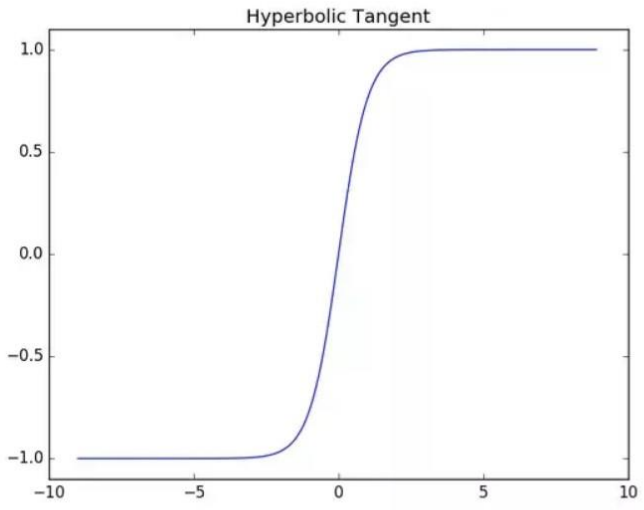
  Range = (-1, 1)

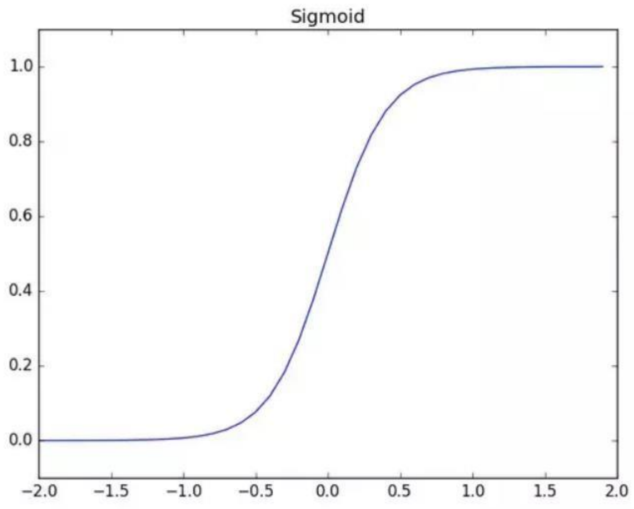
- Sigmoid is a S-shaped curve,

$$f(x) = \sigma(x) = \frac{1}{1 + e_{-x}}$$

  Range = (0, 1)


ReLU
2.0
1.5
1.0
0.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Hyperbolic Tangent
1.0
0.5
0.0
−0.5
−1.0
−10
−5
0
5
10


Sigmoid
1.0
0.8
0.6
0.4
0.2
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0