1. Explain the Activation Functions in your own language

1. sigmoid
2. tanh
3. ReLU
4. ELU
5. LeakyReLU
6. Swish

a)sigmoid

1. Sigmoid or Logistic Activation Function

The Sigmoid Function curve looks like a S-shape.

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

The function is differentiable.That means, we can find the slope of the sigmoid curve at any two points.

The function is monotonic but function’s derivative is not.

The logistic sigmoid function can cause a neural network to get stuck at the training time.

The softmax function is a more generalized logistic activation function which is used for multiclass classification.

This function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below

b)tanh

Tanh Activation is an activation function used for neural networks: f ( x ) = e x − e − x e x + e − x. Historically, the tanh function became preferred over the sigmoid function as it gave better performance for multi-layer neural networks.

c)ReLU

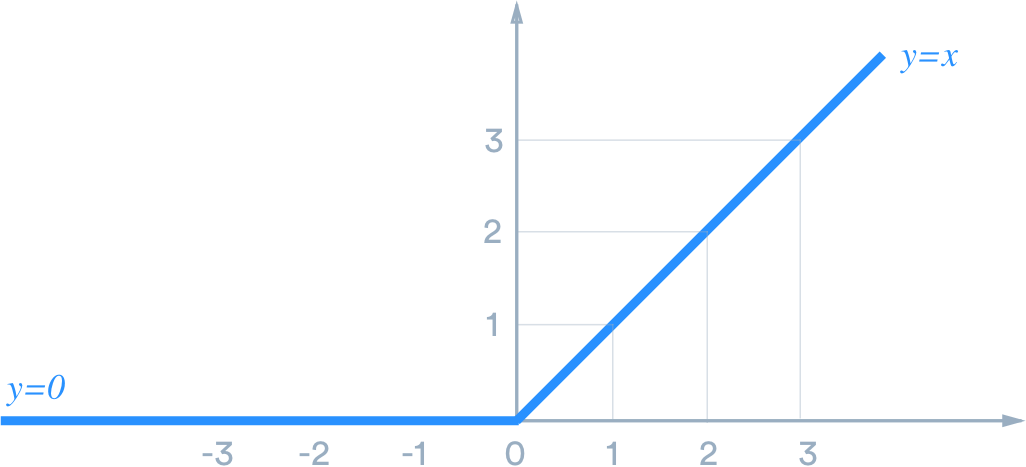
The rectified linear activation function or ReLU is a non-linear function or piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.  
It is the most commonly used activation function in neural networks, especially in Convolutional Neural Networks (CNNs) & Multilayer perceptrons.

It is simple yet it is more effective than it's predecessors like sigmoid or tanh.

Mathematically, it is expressed as:

eq

Graphically it is represented as,



Implementing ReLU function in Python

We can implement a simple ReLU function with Python code using an if-else statement as,

def ReLU(x):

if x>0:

return x

else:

return 0

Python

Copy

or using the max() in-built function over the range from 0.0 to x:

def relu(x):

return max(0.0, x)

Python

Copy

The positive value is returned as it is and for values less than (negative values) or equal to zero, 0.0 is returned.

D)ELU

An ELU activation layer performs the identity operation on positive inputs and an exponential nonlinearity on negative inputs. The layer performs the following operation: f ( x ) = { x , x ≥ 0 α (exp( x ) - 1) , x < 0. The default value of α is 1. Specify a value of α for the layer by setting the Alpha property.

E) LeakyReLU

f(x)=max(0.01\*x , x). This function returns x if it receives any positive input, but for any negative value of x, it returns a really small value which is 0.01 times x. Thus it gives an output for negative values as well.

F)SWISH

Swish is an activation function, f ( x ) = x ⋅ sigmoid ( β x ) , where a learnable parameter. Nearly all implementations do not use the learnable parameter , in which case the activation function is x σ ( x ) ("Swish-1").

2. What happens when you increase or decrease the optimizer learning rate?

Generally, a large learning rate allows the model to learn faster, at the cost of arriving on a

sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train.

What happens when you decrease the learning rate?

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower we travel along the downward slope.

increasing

Increasing the learning rate further will cause an increase in the loss as the parameter updates cause the loss to "bounce around" and even diverge from the minima. Remember, the best learning rate is associated with the steepest drop in loss, so we're mainly interested in analyzing the slope of the plot.

3. What happens when you increase the number of internal hidden neurons?

This notebook investigates how the number of hidden neurons affect the model performance. We will see that increasing the number of hidden neurons increases the performance of a model using the MNIST dataset. The MNIST dataset is a common standard dataset used to evaluate machine learning models performance, which is just a task of recognizing digits from 0 to 9.

This notebook has dependencies on Keras, Scikit-Learn and MatPlotLib.

In [30]:

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

%matplotlib inline

from keras.models import Sequential

from keras.layers.core import Dense, Activation, Dropout

from keras.optimizers import SGD, Adam, RMSprop

from sklearn.preprocessing import \*

from sklearn.cross\_validation import \*

from sklearn.metrics import \*

In [31]:

TRAIN\_FILE = 'data/train.csv'

TEST\_FILE = 'data/test.csv'

In [32]:

train\_data = np.loadtxt(TRAIN\_FILE, skiprows = 1, delimiter = ',', dtype = 'float')

X = train\_data[:, 1:]

# Preprocess the data to make features fall between 0 and 1. Neural networks perform a lot better in this way.

X = X/255

raw\_Y = train\_data[:, ].reshape(-1, 1)

In [33]:

X\_test = np.loadtxt(TEST\_FILE, skiprows = 1, delimiter = ',', dtype = 'float')

# Preprocess the data to make features fall between 0 and 1. Neural networks perform a lot better in this way.

X\_test = X\_test/255

In [34]:

X\_train, X\_cv, raw\_Y\_train, raw\_Y\_cv = train\_test\_split(X, raw\_Y, test\_size = 0.20)

# Converter to transform input into one hot encoding, i.e. [3] => [0, 0, 1, 0, 0, 0, 0, 0, 0, 0].

# Can use the np\_utils from Keras instead.

Y\_expander = OneHotEncoder().fit(raw\_Y)

Y\_train = Y\_expander.transform(raw\_Y\_train).astype(int).toarray()

Y\_cv = Y\_expander.transform(raw\_Y\_cv).astype(int).toarray()

In [35]:

n\_hiddens = [512, 256, 128, 64, 32, 16, 8, 4, 2, 1]

scores = []

for n\_hidden in n\_hiddens:

# Build a simple neural network.

model = Sequential()

model.add(Dense(input\_dim = X.shape[1], output\_dim = n\_hidden))

model.add(Activation('tanh'))

model.add(Dense(output\_dim = 10))

model.add(Activation('softmax'))

sgd = SGD(lr=0.2, decay=1e-7, momentum=0.1, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer='sgd')

model.fit(X\_train, Y\_train, nb\_epoch = 10, batch\_size = 10, show\_accuracy = True, verbose = 1, validation\_split = 0.05)

Y\_cv\_pred = model.predict\_classes(X\_cv, batch\_size = 10, verbose = 1)

score = accuracy\_score(raw\_Y\_cv, Y\_cv\_pred)

scores.append(score)

print('Using [%d] number of hidden neurons yields. Accuracy score: %.4f' % (n\_hidden, score))

print('')

4. What happens when you increase the size of batch computation?

According to popular knowledge, increasing batch size reduces the learners' capacity to generalize. Large Batch techniques, according to the authors of the study “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima,” tend to result in models that become caught in local minima.

5. Why we adopt regularization to avoid overfitting?

Regularization comes into play and shrinks the learned estimates towards zero. In other words, it tunes the loss function by adding a penalty term, that prevents excessive fluctuation of the coefficients. Thereby, reducing the chances of overfitting.

6. What are loss and cost functions in deep learning?

In other words, the loss function is to capture the difference between the actual and predicted values for a single record whereas cost functions aggregate the difference for the entire training dataset. The Most commonly used loss functions are Mean-squared error and Hinge loss.

In mathematical optimization and decision theory, a loss function or cost function (sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.

7. What do ou mean by underfitting in neural networks?

Underfitting is a scenario in data science where a data model is unable to capture the relationship between the input and output variables accurately, generating a high error rate on both the training set and unseen data.

8. Why we use Dropout in Neural Networks?

Dropout layers have been the go-to method to reduce the overfitting of neural networks. It is the underworld king of regularisation in the modern era of deep learning. In this era of deep learning, almost every data scientist must have used the dropout layer at some moment in their career of building neural networks.