**Semeion Data Classifier**

Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science & Engineering

By

**Nikhil Bhardwaj**

To



Maharaja Surajmal Institute of Technology

Affiliated to Guru Gobind Singh Indraprastha University

Janakpuri, New Delhi-58

2018-2022

**i**

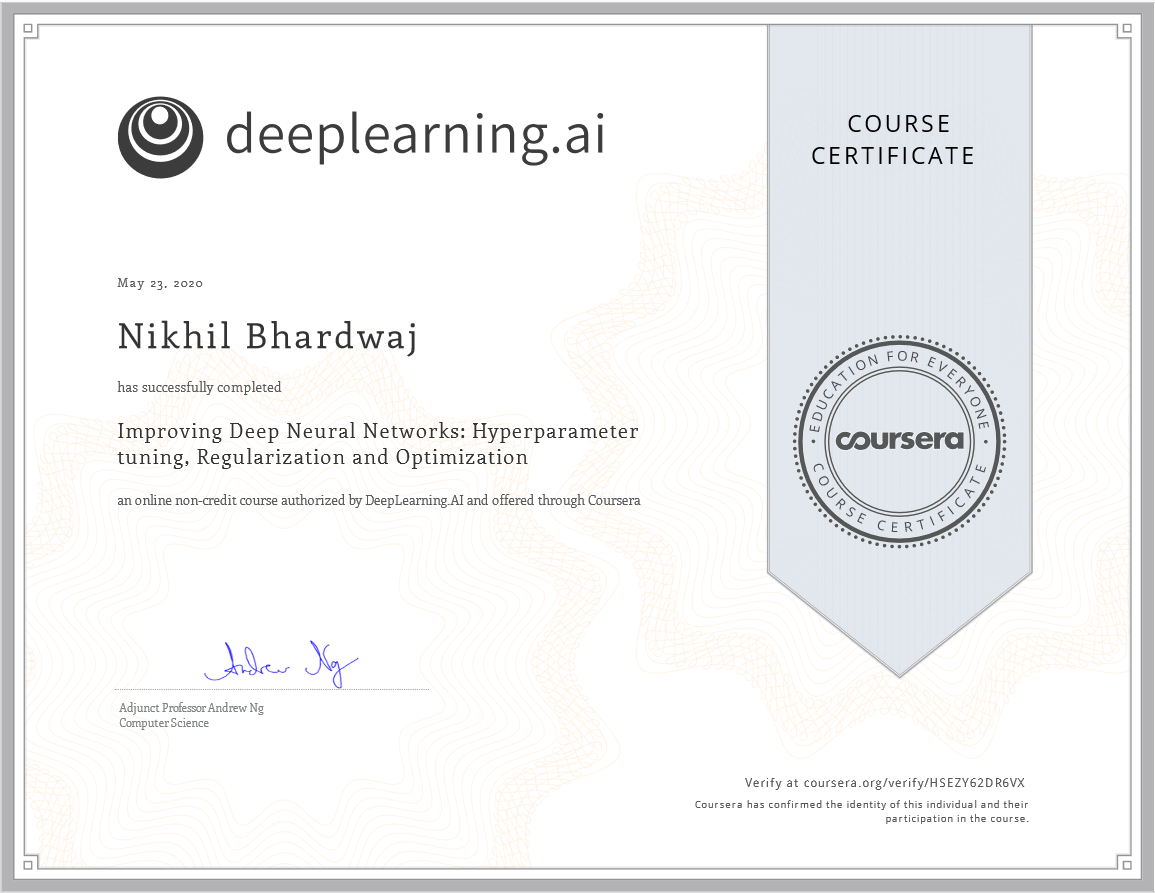
**Certificate (Self declaration)**

I, Nikhil Bhardwaj, Serial Number 06515002718,CSE-2, hereby declare that the report of the project titled ‘Semeion Data Classifier’, is uniquely made by me after the completion of ‘Deep Learning’ course on Coursera.

I hereby certify that the project is intended for academic purposes only and of no further use.

**ii**

**Certificate (Organization)**



**iii**

**Acknowledgement**

I would like to express my immense gratitude and thanks to Mr. Andrew Ng, for guiding me through all the practical aspects required in building up of this project.

I would also like to thank my college, Maharaja Surajmal Institute of technology for giving me an opportunity for showcasing my skills through this project and guiding me throughout the project.

Nikhil Bhardwaj

CSE-2, Morning,

Serial Number – 06515002718,

Batch- 2018-2022

**iv**

**Abstract**

Machine learning and deep learning plays an important role in computer technology and artificial intelligence. With the use of deep learning and machine learning, human effort can be reduced in recognizing, learning, predictions and many more areas. Deep learning systems are revolutionizing technology around us, from voice recognition that pairs you with your phone to autonomous vehicles that are increasingly able to see and recognize obstacles ahead.

This project presents recognizing the handwritten digits (0 to 9) from the famous MNIST dataset, comparing classifiers on basis of performance, accuracy, time, sensitivity, positive productivity, and specificity with using different parameters with the classifier.

Digit recognition system is the working of a machine to train itself or recognizing the digits from different sources like emails, bank cheque, papers, images, etc. and in different real-world scenarios for online handwriting recognition on computer tablets or system, recognize number plates of vehicles, processing bank cheque amounts, numeric entries in forms filled up by hand and so on.

Each image is 16 pixels in height and 16 pixels in width, for a total of 256 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. There are 256 inputs for the given neural network. The neural network developed has 3 layers with number of neurons as [100,32 ,10]

**v**

**INDEX**

**Content Page Number**

**Chapter 1**

**1.Objective .1**

**2.Methodology .2**

**3.Software Used .3**

**4.About Coursera .4**

**Chapter 2**

**5.Project Design .5**

**Dataset**

**Forward Propagation**

**Backpropagation**

**Learning rate and need of Backpropagation**

**Activation Function**

**Architecture of Neural Network**

**Chapter 3**

**6.Implementation in Python…………………………………………..13**

**Chapter 4**

**7.Results………………………………………………………………...16**

**8.Equivalent in TensorFlow………………………………….…….….20**

**9.Discussion………………………………………………………….....21**

**Chapter 5**

**10.Future Scope………………………………………………………..27**

**11.Conclusion……………………………………………………..........28**

**12.References………………………………………………………….29**

**Chapter 1**

**Objective**

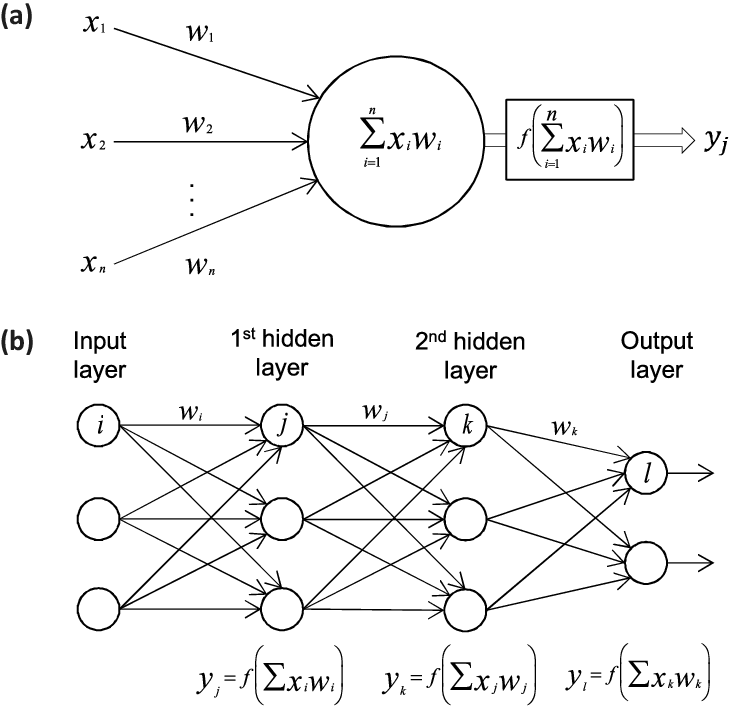
Digit Recognition System can be used in applications where there is a need to automate current processes. For example,

* A post office can use a recognition system which can segregate the packages based on their pin codes or area codes.
* It can be used at personal garages to identify incoming cars so that we can identify incoming cars and we can automate the process of opening garage door or house door.
* Autonomous cars for traffic signs and speed limit recognition.
* Text to speech conversion for the teaching visually impaired and physically handicapped people.

**Methodology**

A Deep Neural Network uses neurons for learning a complex mathematical function. It is the

basic building block of a neural network.



In a neural network, a neuron is connected to all the neurons present in the previous layers of the neural network. The neurons in the input layer are given the input parameters. All the neurons are initialized with random weights. For each and every feature, there is a corresponding weight specified by the neuron.

Each neuron would have different weights for all features. As visible in Fig.a, the values given as input to the neuron is then multiplied to the corresponding weights. After the multiplication of all values, they are added and an activation function is then applied to the value.

**Software Used**

The main language used for development of the neural network is Python which has further used pandas library for handling and manipulation of the dataset, Numpy for multiplication for high dimensional matrices and matplotlib for displaying the test results of the training of the neural network.







**About Coursera**

Coursera was founded by Daphne Koller and Andrew Ng with a vision of providing life-transforming learning experiences to anyone, anywhere. It is now a leading online learning platform for higher education, where 71 million learners from around the world come to learn skills of the future.

More than 200 of the world’s top universities and industry educators’ partner with Coursera to offer courses, Specializations, certificates, and degree programs. Thousands of companies trust the company’s enterprise platform Coursera for Business to transform their talent. Coursera for Government equips government employees and citizens with in-demand skills to build a competitive workforce. Coursera for Campus empowers any university to offer high-quality, job-relevant online education to students, alumni, faculty, and staff. Coursera is backed by leading investors that include Kleiner Perkins, New Enterprise Associates, Learn Capital, and SEEK Group.

Users can enrol in guided projects to gain hands-on experience in a subject, such as building a data science web app or creating JavaScript animations. Another option for Coursera users is signing up for a specialization, in which they complete a series of courses and projects on a topic, such as supply chain management or search engine optimization.

The company offers more than 4,300 courses, more than 450 specializations, more than 440 projects, more than 30 certificates and 20 degrees, according to its website.

Coursera is based in Mountain View, California, in the heart of Silicon Valley.

Users can access materials for more than 1,600 free Coursera courses and opt to pay for certificates upon completion. Many courses also can be audited for free, but auditors may not be able to submit some assignments or receive grades for their work.



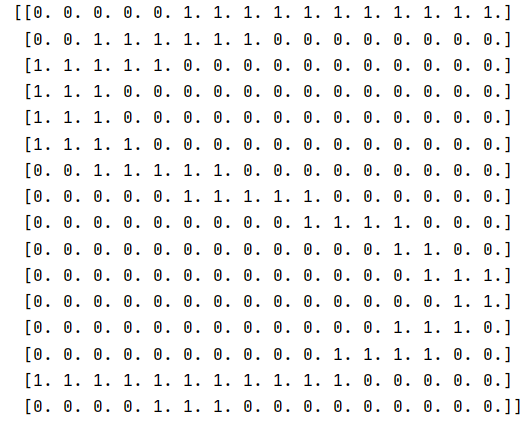
**Chapter 2**

**Project Design**

Handwritten character recognition is one of the practically important issues in pattern recognition applications. The applications of digit recognition includes in postal mail sorting, bank check processing, form data entry, etc. The heart of the problem lies within the ability to develop an efficient algorithm that can recognize hand written digits and which is submitted by users by the way of a scanner, tablet, and other digital devices.

**Dataset**

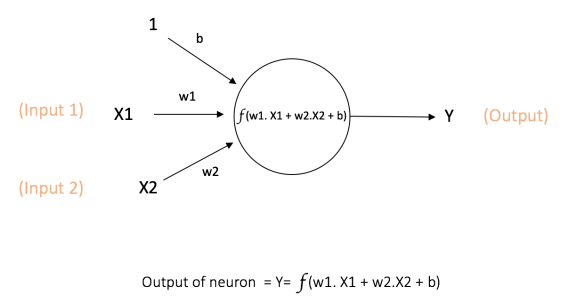
The data for the model has been read with the help of helper functions from pandas. Each and every sample is of the form of a 16x16 matrix. The output layer of the neural network has 10 neurons because the number of values that the output can take is equal to 10. Therefore, for each case there is only one neuron that has the maximum value and that tells us about the digit that has been predicted by the model.



**A sample for the digit 5**

**Forward Propagation**

The flow of data in a deep neural network is in forward direction. Each and every neuron is fully connected to each neuron of the previous layer and the layer next to it. Therefore, it has a set of weights for all inputs it receives from previous layer and a corresponding bias. The weights of neural network are multiplied by the input values and the corresponding bias are added. After this, an activation function is applied to the calculated value. Then the output of this neuron becomes the input for the upcoming layers of the neural network.



As shown in the figure, w1 is the weight of neuron for input feature X1 and w2 is weight of neuron for input feature X2. b is the overall bias of the neuron. The weights of the neural network are randomly and normally initialized to values less than 1 and greater than -1.

The weights are randomly initialized so that each and every neuron is unique and thus it calculates a different feature of the input sample. The weights are initialized normally so that it avoids the problem of exploding gradients and vanishing gradients. If the weights are too small or too large, then in backpropagation we might update the weights of the neurons to very large or very small values. Therefor they are initialized to small values.

The neurons in the model are fully connected to all the neurons present in the previous and next layer. Thus, the weights(W) of each neuron are multiplied by the inputs(X) of the neurons present in the previous layers and the bias is added. This calculated value acts as input for next layer from the current neuron to next layer.

Y=f (WT.X +b)

, where f is the activation function used. Each neuron in the inner layers use ReLU function and output layer uses SoftMax classifier as there are multiple classes for the data.

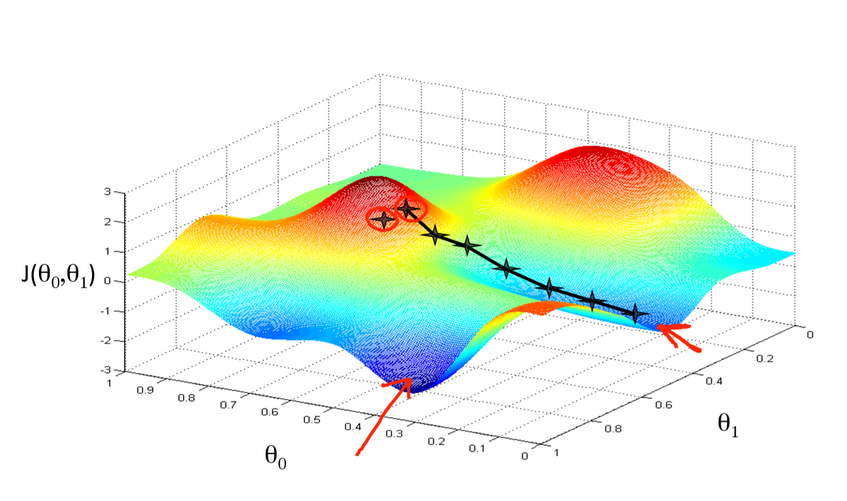
The number of neurons were [100,32,10] in each 3 layers. The learning rate of the model was varied between 0.1 to 0.9.A cache was kept for storing the calculated values by the neural network to be later used for backpropagation step.

The model uses stochastic gradient descent for backpropagation for correction of its weights of all neurons. After each sample is fed through the neural network, we correct the neurons according to the loss incurred in prediction.

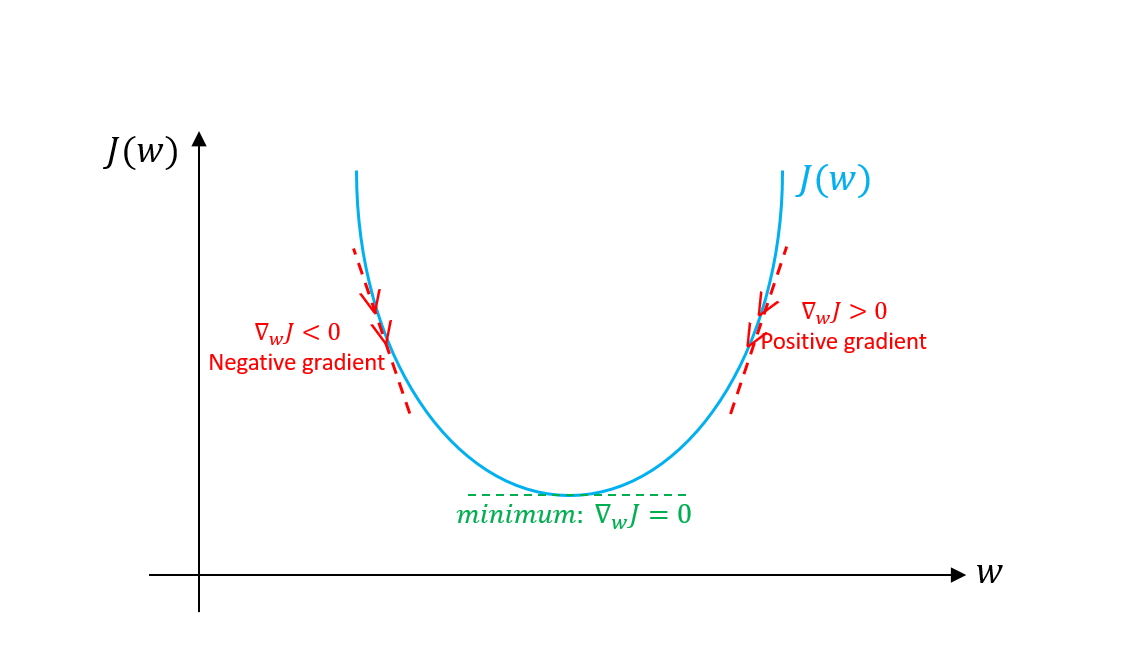
**Backpropagation**

After processing each sample, we calculate a loss corresponding to the prediction made by the model. This has been demonstrated with the help of graphs corresponding to different learning rates. Calculation of loss is and integral and important part of training a model and we can make certain conclusions from the behavior of loss in the training phase.

Corresponding to the model, we use different loss functions. The choice of loss function is made keeping in mind the fact that the function should have only one global minimum. Otherwise we would get stuck in a local minima and we would not get the optimum set of parameters for the model.



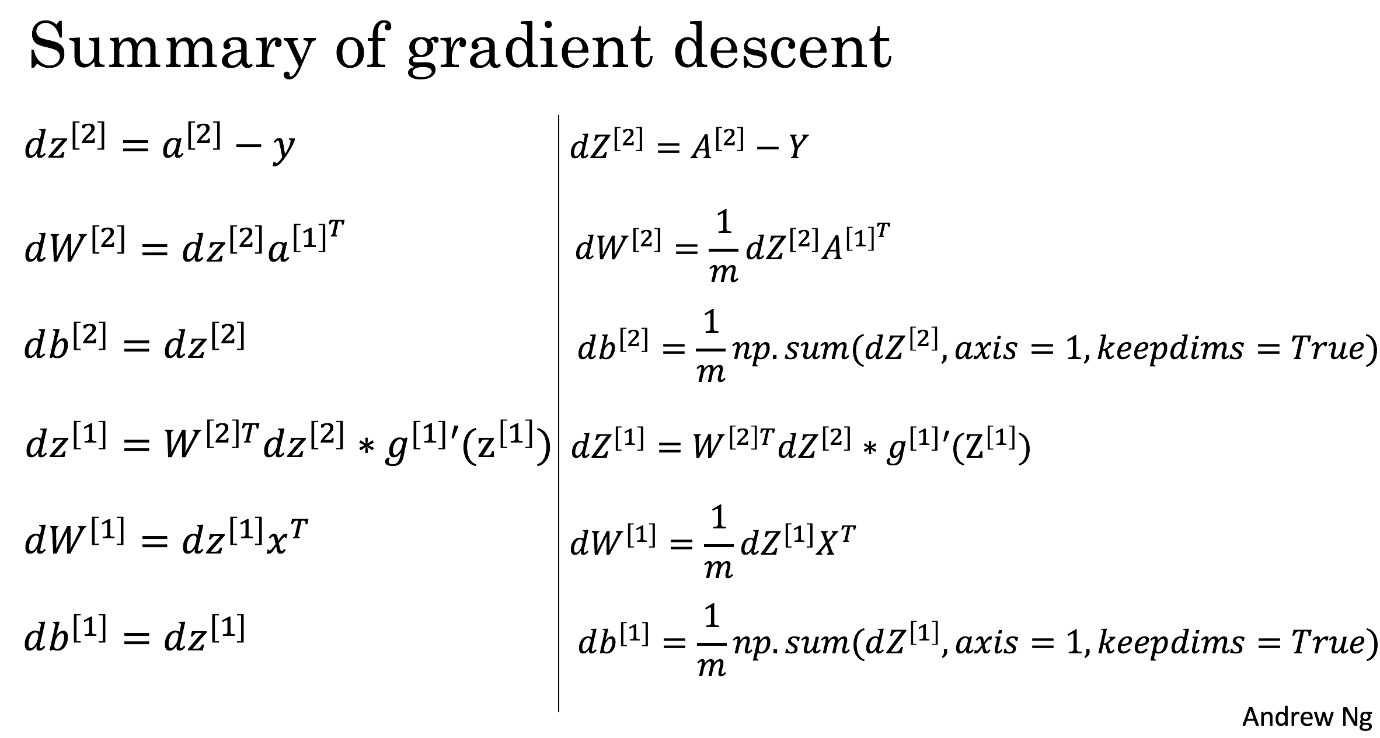
**Loss calculation and the application of gradient descent**



**Need of convex loss function**

The property that the loss function has only one global minima ensures the fact that the set of parameters that we have correspond to the most optimum model that is possible with the given set of parameters of the model.

Gradient Descent is a characteristic if improvised learning where we make predictions based on a set of parameters. For algorithms such as k-NN, we use a different strategy in which we vary the amount of points we consider for making a prediction



**Image taken from course undertaken at Coursera by Mr. Andrew Ng**

We have to keep a track of the calculated results by neurons at each layer as they are used in backpropagation while correcting the weights. It depends on the choice of the programmer to whether use stochastic, batch or group gradient descent.

**Learning Rate and need of backpropagation**

Correction of neuron parameters

W=W- *n .*dW

B=B- *n .dB*

*,* where *n* is the learning rate*.*

W is the set of weights

B is the corresponding bias

The weights of a neural network cannot be calculated using an analytical method. Instead, the weights must be discovered via an empirical optimization procedure called stochastic gradient descent.

The optimization problem addressed by stochastic gradient descent for neural networks is challenging and the space of solutions (sets of weights) may be comprised of many good solutions (called global optima) as well as easy to find, but low in skill solutions (called local optima).

The amount of change to the model during each step of this search process, or the step size, is called the “learning rate” and provides perhaps the most important hyperparameter to tune for your neural network in order to achieve good performance on your problem.

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.

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.

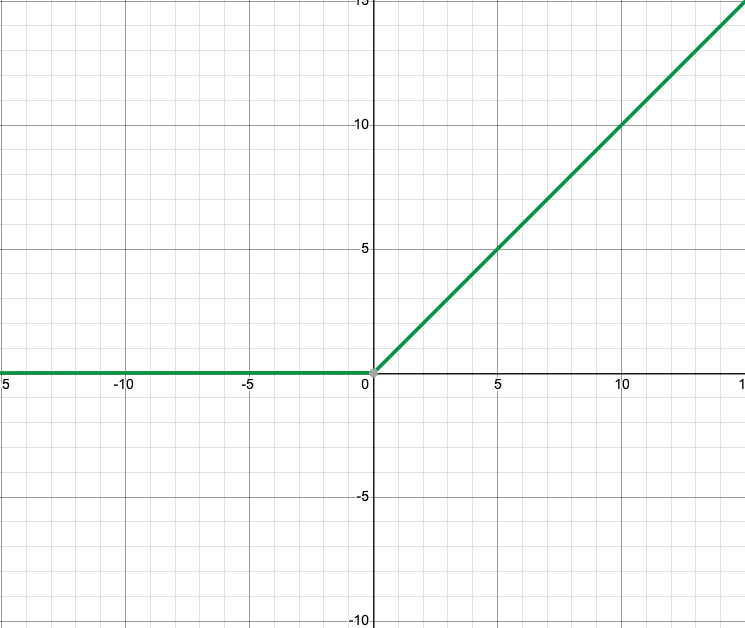
**Activation Function**

The activation function is a mathematical “gate” in between the input feeding the current neuron and its output going to the next layer. It can be as simple as a step function that turns the neuron output on and off, depending on a rule or threshold. Or it can be a transformation that maps the input signals into output signals that are needed for the neural network to function.

ReLU activation function has been used for each neuron in the neural network. It only activates the neuron when the final output of the neuron is positive. Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated or not, based on whether each neuron’s input is relevant for the model’s prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.

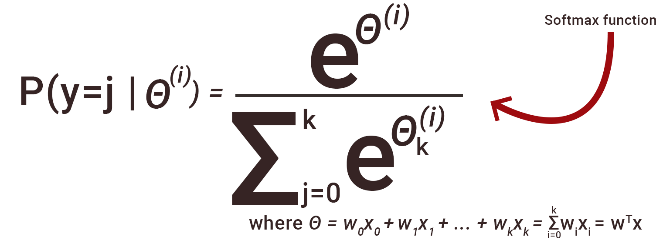
An additional aspect of activation functions is that they must be computationally efficient because they are calculated across thousands or even millions of neurons for each data sample. Modern neural networks use a technique called backpropagation to train the model, which places an increased computational strain on the activation function, and its derivative function.

ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.



**y=ReLU(x)=max(0,x)**

**The output layer of the neural network uses SoftMax classifier, which is ideal for multiclass regression analysis. In digit recognition system, the number of possible outputs is 10 (digits from 0-9). Therefore, the number of neurons in output layer are fixed and are equal to 10.**



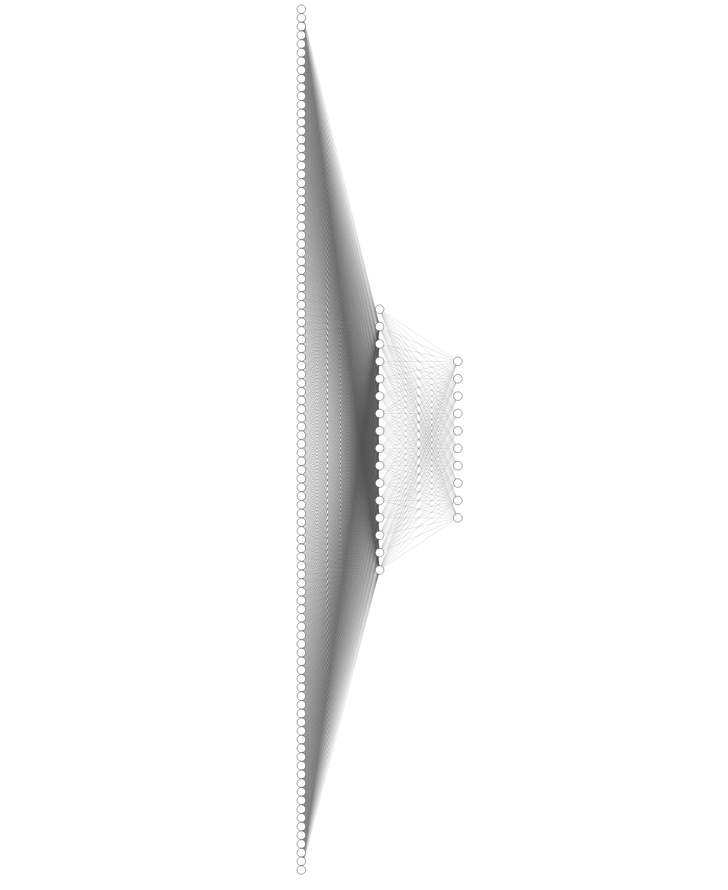
**Formula for SoftMax classifier**

The SoftMax classifier calculates the probability or likelihood of a particular sample belonging to a particular class from the given possible classes. The term in the denominator is greater than term in numerator thus it is similar to probability. We assign the class to a sample which has the maximum probability calculated by the sample.

SoftMax is a generalization of logistic regression that we can use for multi-class classification (under the assumption that the classes are mutually exclusive). In contrast, we use the (standard) Logistic Regression model in binary classification tasks. Many multi-layer [neural networks](https://deepai.org/machine-learning-glossary-and-terms/neural-network) end in a penultimate layer which outputs real-valued scores that are not conveniently scaled and which may be difficult to work with. Here the SoftMax is very useful because it converts the scores to a normalized [probability distribution](https://deepai.org/machine-learning-glossary-and-terms/probability-distribution), which can be displayed to a user or used as input to other systems. For this reason, it is usual to append a SoftMax function as the final layer of the neural network.

**SoftMax** extends the idea of probability into a multi-class world. That is, SoftMax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would.

**Architecture of neural network**



**Illustration for architecture of Neural Network**

Number of neurons in layer 1=100(ReLU)

Number of neurons in layer 2=16(ReLU)

Number of neurons in layer 3=10(SoftMax)

Number of edges=1660

Total number of neurons=126

**Chapter 3**

**Implementation in Python**

import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
from mnist import MNIST  
  
  
def initialise\_weights\_bias(layer\_dims, input\_features, Parameters):  
 u = layer\_dims[0]  
 Parameters[**"W1"**] = np.random.randn(u, input\_features) \* .001  
 Parameters[**"b1"**] = np.random.randn(u, 1)  
  
 for i in range(2, layer\_dims.shape[0] + 1):  
 Parameters[**"W"** + str(i)] = np.random.randn(layer\_dims[i - 1], layer\_dims[i - 2]) \* .001  
 Parameters[**"b"** + str(i)] = np.random.randn(layer\_dims[i - 1], 1)  
  
  
def relu(temp):  
 return np.maximum(temp, 0)  
  
  
def softmax(x):  
 den = np.sum(x, axis=1)  
 den = den.reshape(10, 1)  
 exp = np.exp(den)  
  
 exp\_sum = np.sum(exp)  
 exp = exp / exp\_sum  
  
 return exp

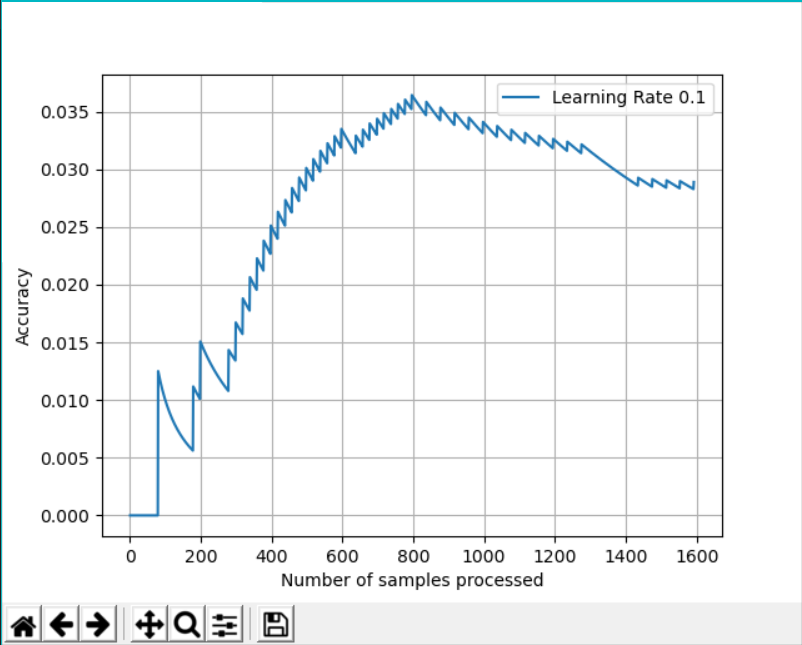
def forward\_propagation(layer\_dims, Parameters, X, Y, cache, num\_of\_iterations):  
 learning\_rate = 0.2;  
 for e in range(1, 2):  
 learning\_rate = learning\_rate + 0.05;  
 initialise\_weights\_bias(layer\_dims, 16, Parameters)  
 correct = 0  
 count = 0  
 pred\_list = []  
 act\_list = []  
 err\_list = []  
 *# np.random.shuffle(X)* for i in range(num\_of\_iterations \* X.shape[0]):  
 i = i % (X.shape[0] - 1)  
 count = count + 1  
  
 input = X[i]  
 for j in range(1, layer\_dims.shape[0]):  
 *# Regression* temp = np.dot(Parameters[**"W"** + str(j)], input) + Parameters[**"b"** + str(j)]  
  
 *# Cache formation* apple = (input, Parameters[**"W"** + str(j)], Parameters[**"b"** + str(j)])  
  
 final = relu(temp)  
 finalApple = (final)  
  
 *# Storing for future* cac = (apple, finalApple)  
 cache.append(cac)  
  
 input = final  
  
 temp = np.dot(Parameters[**"W"** + str(layer\_dims.shape[0])], input) + Parameters[  
 **"b"** + str(layer\_dims.shape[0])]  
 final = softmax(temp)  
  
 *# Loss calculation* actual = Y[i]  
 actual = actual.reshape(10, 1)  
  
 act = (np.argmax(actual))  
 pred = (np.argmax(final))  
  
 act\_list.append(act)  
 pred\_list.append(pred)  
  
 if (act == pred):  
 correct = correct + 1  
  
 dZ = -(final - actual)\*\*2   
  
 dW = np.sum(dZ \* temp)  
 db = np.sum(dZ, axis=1, keepdims=True)  
  
 Parameters[**"W"** + str(layer\_dims.shape[0])] = Parameters[**"W"** + str(layer\_dims.shape[0])] - learning\_rate \* dW  
 Parameters[**"b"** + str(layer\_dims.shape[0])] = Parameters[**"b"** + str(layer\_dims.shape[0])] - learning\_rate \* db  
  
 for k in range(layer\_dims.shape[0] - 1, 0, -1):  
 caches = cache[k - 1]  
 A\_prev, W, b = caches[0]  
 final = caches[1]  
  
 final[final < 0] = 0  
 final[np.where(final > 0)] = 1  
 dZ = final  
  
 dW = np.dot(dZ, A\_prev.T)  
 db = np.sum(dZ, axis=1, keepdims=True)  
  
 Parameters[**"W"** + str(k)] = Parameters[**"W"** + str(k)] - learning\_rate \* dW  
 Parameters[**"b"** + str(k)] = Parameters[**"b"** + str(k)] - learning\_rate \* db  
  
 err\_list.append(correct / count)  
  
 *# lest.append(list(range(1, len(err\_list)+1)),err\_list)* plt.plot(list(range(1, len(err\_list) + 1)), err\_list, label=**'Learning Rate '** + str(round(learning\_rate, 3)))  
 plt.legend()  
 plt.ylabel(**'Accuracy'**)  
 plt.xlabel(**'Number of samples processed'**)plt.grid()  
 plt.show()

##Driver Code

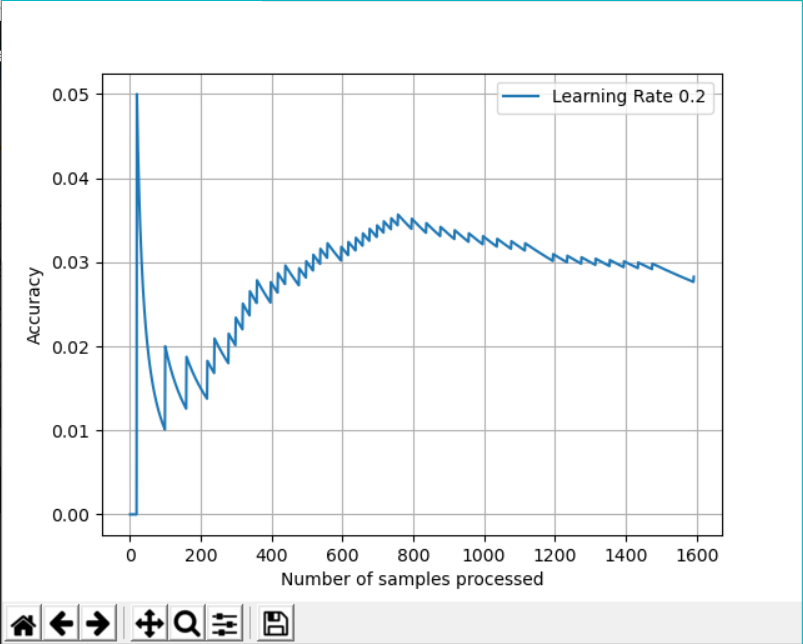
layer\_dims = np.array([100, 32, 10])  
  
*##Reading data*exchange\_rates = pd.read\_csv(**"semeion.data"**, sep=**" "**)  
a = exchange\_rates.to\_records()  
d = exchange\_rates.reset\_index().values  
f = d[:, 1:257]  
num = d[:, 257:267]  
e = np.reshape(f, (d.shape[0], 16, 16))  
  
Parameters = {}  
input\_features = 16  
cache = []  
  
*##Starting forward propagation*initialise\_weights\_bias(layer\_dims, input\_features, Parameters)  
forward\_propagation(layer\_dims, Parameters, e, num, cache, 1)

**Chapter 4**

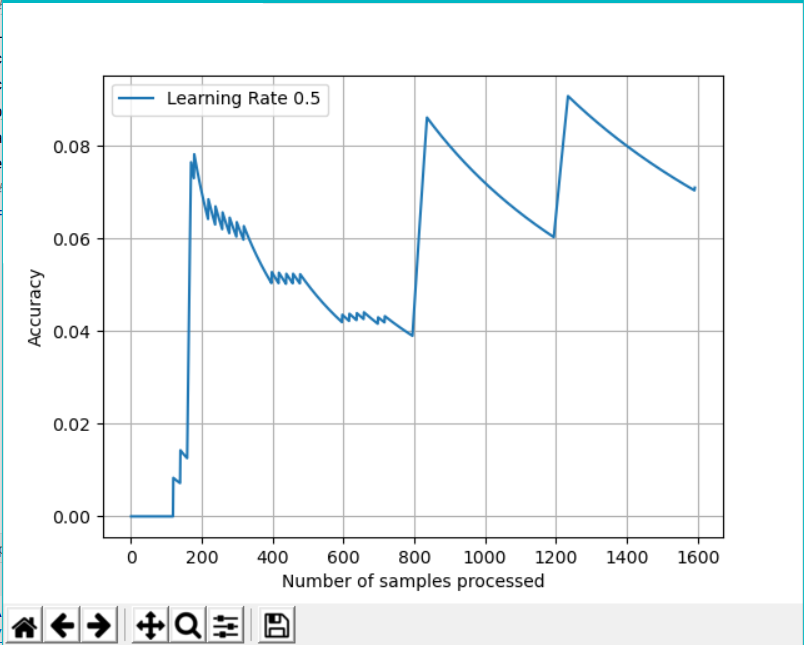
**Results**



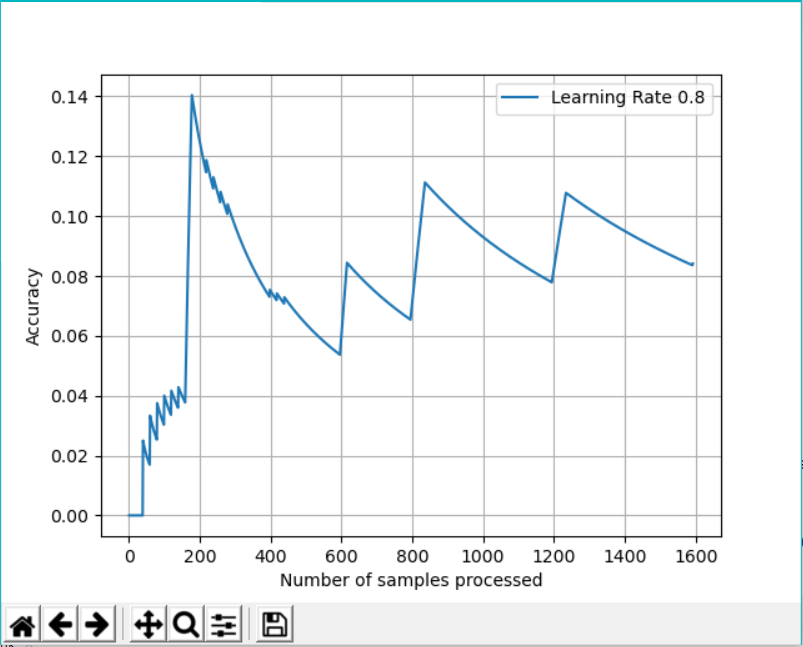
**Learning rate 0.1**



**Learning rate 0.2**



**Learning rate 0.5**



**Learning Rate 0.8**

**Equivalent Model in TensorFlow**

import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

mnist = tf.keras.datasets.mnist

(x\_train, y\_train),(x\_test, y\_test) = mnist.load\_data()

x\_train = tf.keras.utils.normalize(x\_train, axis=1)

x\_test = tf.keras.utils.normalize(x\_test, axis=1)

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))

model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))

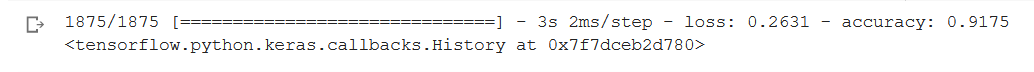
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

opt = tf.keras.optimizers.SGD(learning\_rate=0.45)

model.compile(optimizer="SGD", loss="sparse\_categorical\_crossentropy", metrics=["accuracy"])

model.fit(x=x\_train, y=y\_train, epochs=1)

**Output:**



**Discussion**

There are three ways a neural network can [learn](https://kids.kiddle.co/Machine_learning): [Supervised learning](https://kids.kiddle.co/Supervised_learning), Unsupervised learning and [Reinforcement learning](https://kids.kiddle.co/Reinforcement_learning). These methods all work by either minimizing or maximizing a [cost function](https://kids.kiddle.co/Loss_function), but each one is better at certain tasks.

### **Supervised Learning**

In [Supervised learning](https://kids.kiddle.co/Supervised_learning), the neural network is trained using example inputs and the correct output. The network can then work out the relationship between the input and output. For example, a network could be trained by showing it details about houses and the sale price. Once it has finished training it could estimate the sale price of another house by analysing information like the number of bedrooms and local crime rate.

Another example is the ALV (Autonomous Land Vehicle). [DARPA](https://kids.kiddle.co/DARPA) funded this project in the 1980s. In a demonstration in 1987 it travelled 600 metres at 3 km/h over difficult land, with sharp rocks, vegetation and steep ravines. This vehicle could drive itself as fast as 30 km/h. This network watched a 'teacher' drive, and saw the road using laser radar. The learning process was repeated for different road types. ALV used a kind of neural network called a multi-layer perceptron in which multiple layers of neurons are connected in series.

**Unsupervised Learning**

Unsupervised learning only trains using inputs, and the network has to figure out how they relate to each other. This method is used to solve Clustering problems, estimation problems, and self-organising maps. For example, a self-organizing map can be used to categorize [iris](https://kids.kiddle.co/Iris) flowers by stem size and colour.

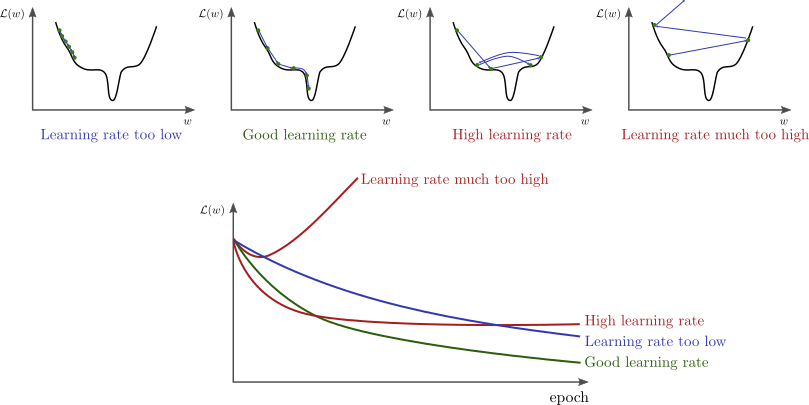
### **Reinforcement Learning**

A [reinforcement learning](https://kids.kiddle.co/Reinforcement_learning) neural network learns by watching a teacher's actions. It works out the smallest cost and tries to use this to work out how to make the smallest cost in the future. It can be thought of as a Markov decision process. Another simple way to think of this is as "[carrot and stick](http://en.wiktionary.org/wiki/carrot_and_stick)" learning (learning that rewards good behaviour and punishes bad behaviour).

Recently, a research team from the University of Hertfordshire, UK used reinforcement learning to make an iCub [humanoid](https://kids.kiddle.co/Humanoid) robot learn to say simple words by babbling.

The neural network seemed to perform good on learning rate near 0.5 and it used supervised learning. If the learning rate was kept low then it seemed to learn very slowly and if the learning rate was high then we had good accuracy in the initial iterations itself which might be a sign of overfitting.

**Learning Rates**



**Image taken from Stanford CS231(by Andrew Ng)**

Choice of learning rate, a hyperparameter, is very important because

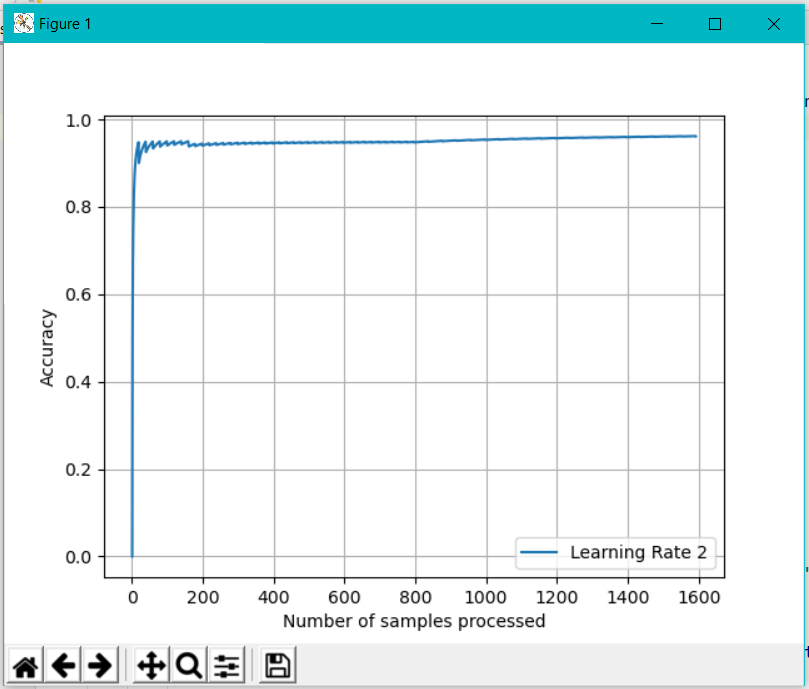
* A learning rate that is too low will take a long time to converge. This is especially true if there are a lot of saddle points in the loss-space. Along a saddle point, gradient will be close to zero in many directions. If the learning rate η
* is also very low, it can slow down the learning substantially.
* A learning rate that is too high can “jump” over the best configurations
* A learning rate that is much too high can lead to divergence

In setting a learning rate, there is a trade-off between the rate of convergence and overshooting. While the [descent direction](https://en.wikipedia.org/wiki/Descent_direction) is usually determined from the [gradient](https://en.wikipedia.org/wiki/Gradient_descent) of the loss function, the learning rate determines how big a step is taken in that direction A too high learning rate will make the learning jump over minima but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum.

**Overfitting**

In statistics, overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably". An overfitted model is a statistical model that contains more parameters than can be justified by the data. The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e. the noise) as if that variation represented underlying model structure.[3]:45

In other words, the model remembers a huge number of examples instead of learning to notice features. This is possible because of the fact that out dataset is small with only 1592 samples. Due to this problem, the model would perform poorly on samples that are not yet seen. Therefore, we would have high error on unseen data.The problem is that these parameters are not optimum and do not apply to new data thus negatively impact the models ability to generalize.



**Overfitting as we are approaching accuracy of nearly 97% early.**



If we have a larger learning rate then it leads to overfitting. Thus, the model would not be able to perform good on samples that are not from the training dataset. We would get wrong answers when the input features vary even slightly from the desired class. The set of parameters which we obtain after training are only optimum for the training dataset.

Overfitting can be caused due to certain reasons, which include:

* Overtraining on the given dataset
* High learning rate
* Small dataset

**Underfitting**

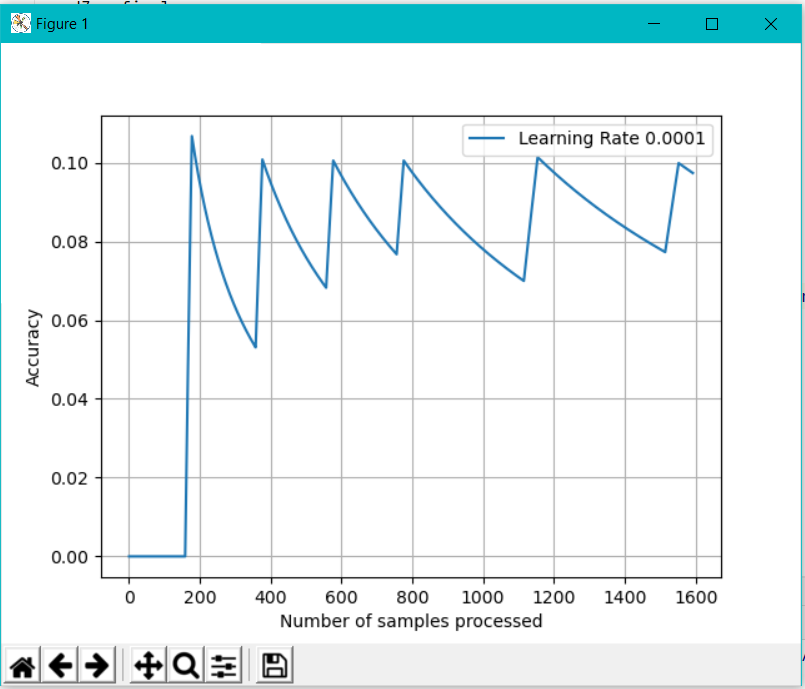
Underfitting occurs when a statistical model cannot adequately capture the underlying structure of the data. An under-fitted model is a model where some parameters or terms that would appear in a correctly specified model are missing. Under-fitting would occur, for example, when fitting a linear model to non-linear data. Such a model will tend to have poor predictive performance.

Underfitting occurs due to a variety of reasons:

* Low iterations on the dataset
* Low learning rate
* Large dataset as compared to number of epochs

Over-fitting and under-fitting can occur in machine learning, in particular. In machine learning, the phenomena are sometimes called "over-training" and "under-training".

The possibility of over-fitting exists because the criterion used for selecting the model is not the same as the criterion used to judge the suitability of a model. For example, a model might be selected by maximizing its performance on some some set of [training data](https://en.wikipedia.org/wiki/Training_data) and yet its suitability might be determined by its ability to perform well on unseen data; then over-fitting occurs when a model begins to "memorize" training data rather than "learning" to generalize from a trend.



**Low Accuracy of less than 10% showing underfitting.**

As shown above, due to low learning rate we were not able to get high accuracy as the parameters were not updated accordingly.

We know our model parameters, we feed known data to the neural networks and how they are put together. But we usually do not understand how they arrive at a particular solution. Neural networks are essentially black boxes and researchers have a hard time understanding how they deduce conclusions.

The lack of ability of neural networks for reason on an abstract level makes it difficult to implement high-level cognitive functions. Also, their operation is largely invisible to humans, rendering them unsuitable for domains in which verification of process is important. Deep Learning models, once trained, can deliver tremendously efficient and accurate solution to a specific problem. However, in the current landscape, the neural network architectures are highly specialized to specific domains of application.

Most of our systems work on this theme, they are incredibly good at solving one problem. Even solving a very similar problem requires retraining and reassessment. Researchers are working hard in developing Deep Learning models which can multitask without the need of reworking on the whole architecture.

**Vanishing and Exploding Gradients**

A gradient in the context of a neural network refers to the gradient of the loss function with respect to the weights of the network.

This gradient is calculated using backpropagation. The goal here is to find the optimal weight for each connection that would minimise the overall loss of the network.

While in principle the recurrent network is a simple and powerful model, in practice, it is, unfortunately, hard to train properly.  The recurrent connections in the hidden layer allow information to persist from one input to another.

The exploding gradients problem refers to the large increase in the norm of the gradient during training. Such events are caused by the explosion of the long term components, which can grow exponentially more than short term ones.

The vanishing gradients problem refers to the opposite behaviour, when long term components go exponentially fast to norm 0, making it impossible for the model to learn the correlation between temporally distant events.

Focussing on gradients is important because:

* Exploding gradients are obvious. Gradients become NaN (not a number) eventually and whole training comes to a halt.
* Since the weights are updated proportional to the gradient, a vanishing gradient or a small value will result in a small change in the value of weight. The value of a weight, which is multiplied with the input, decides whether a certain input needs to be taken seriously or not. No change in value means the network isn’t learning anything. In short, there is no point in training.
* The solution to exploding gradients are simple and effective such as having a pre-defined threshold
* The uncertainty of vanishing gradients makes them difficult to handle.

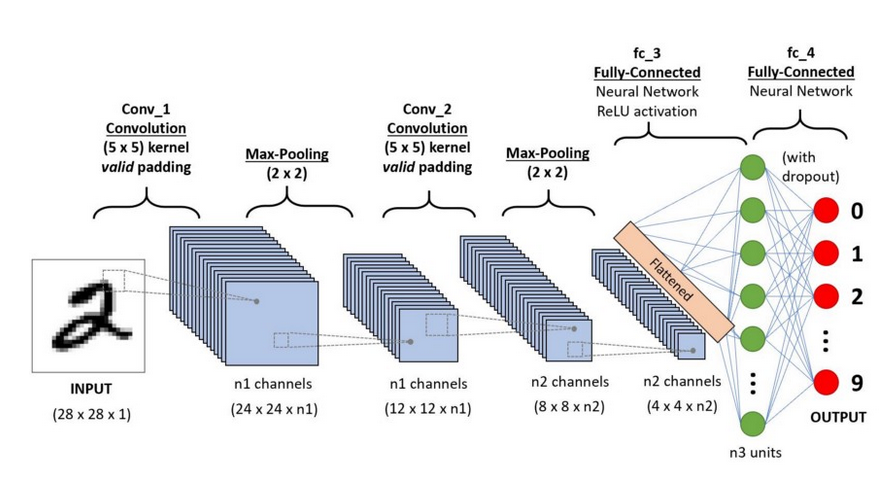
**Chapter 5**

**Future Scope**

The model can be trained on the original NIST dataset which consists of 60,000 images for training set and 10,000 images in the testing set. Thus, the model can be extended with more layers and even more neurons to get even more accuracy. But that requires tremendous computational power and storage. Also, it would take a large amount of time to the model to fully train itself.

We can also create a GUI (Graphical User Interface) using Java which lets a user to input a given number by drawing it on the screen, then we can then convert that image to suitable size for feeding into a neural network. This will be useful for visually handicapped and can be further extended to letter recognition for different languages.

Further, we can use a CNN (Convolutional Neural Network) which uses filters in place of neurons for feature extraction. In a CNN, each filter extracts a feature of the input image and then the filtered features are ultimately fed into a neural network.

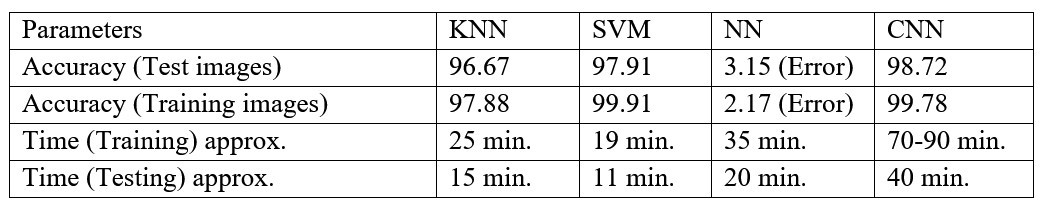


**A Convolutional Neural Network for Digit Recognition**

**Conclusion**

In this project, I have tried to build up a neural network with minimal use of pre-written libraries. The accuracy achieved so far was nearly 90%-95% which is fairly good considering the number of layers and neurons. With access to more memory and hard disk space, we would be able to make even larger models with larger datasets. To ensure better efficiency and less time consumption, data scientists switch to multi-core high performing GPUs and similar processing units. These processing units are costly and consume a lot of power.

Deep Learning may be one the primary research verticals for Artificial Intelligence, but it certainly is not flawless. While exploring new and less explored territories of cognitive technology, it is very natural to come across certain hurdles and difficulties.



**Test results with other algorithms** **taken from** **Handwritten Digit Recognition using Machine Learning published in International Journal of Computer Science and Engineering in June 2018**

**References-**

* **Machinelearningmastery.com**
* **Deeplearning.ai**
* **Wikipedia.org**
* **Coursera (Specialization in Deep Learning by Mr. Andrew Ng)**
* **Medium.com**
* **Tensorflow.org**
* **Deepai.org**
* **Alexnenail.me (Illustration)**