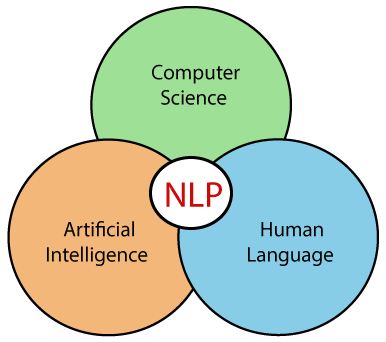
**Natural Language Processing**

**What is NLP?**

NLP stands for **Natural Language Processing**, is a part of **Computer Science, Human language,** and **Artificial Intelligence**. It is the technology that is used by machines to understand, analyze, manipulate, and interpret human languages. The main agenda of NLP is to help developers to organize knowledge for performing tasks such as **translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction,**and**topic segmentation.**

****

**Figure 1: NLP as an interdisciplinary domain**

**Component of NLP:**

*There are Two component of NLP:*

1. **Natural Language Understanding (NLU)**
2. **Natural Language Generation (NLG)**

**Natural Language Understanding (NLU):**

Natural Language Understanding (NLU) helps the machine to understand and analyze human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.

NLU is mainly used in business applications to understand customer problems in both spoken and written language.

NLU involves the following tasks -

* It is used to map the given input into useful representation.
* It is used to analyze different aspects of the language.

**Natural Language Generation (NLG):**

NLG, a subfield of artificial intelligence (AI), is a software process that automatically transforms data into plain-English content. This technology can actually narrate a story – exactly like that of a human analyst – by writing sentences and paragraphs for you. NLG is one of the fastest growing technologies to be adopted in the enterprise. There are many use-cases for NLG, but where it is seen to be most effective is when deployed to automate time-intensive data analysis and reporting activities.

**What is Artificial Intelligence (AI):**

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include [expert systems](https://www.techtarget.com/searchenterpriseai/definition/expert-system), natural language processing, speech recognition and [machine vision](https://www.techtarget.com/searchenterpriseai/definition/machine-vision-computer-vision).

**How Does AI Work?**

AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states. In this way, a chatbot that is fed examples of text chats can learn to produce lifelike exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

AI programming focuses on three cognitive skills: **learning, reasoning and self-correction.**

1. **Learning processes**

This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called [algorithms](https://whatis.techtarget.com/definition/algorithm), provide computing devices with step-by-step instructions for how to complete a specific task.

1. **Reasoning processes.** This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.
2. **Self-correction processes.** This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

**Advantages of NLP:**

* NLP helps users to ask questions about any subject and get a direct response within seconds.
* NLP offers exact answers to the question, which means it does not offer unnecessary and unwanted information.
* NLP helps with human computer interaction.
* It is extremely time efficient.
* Most of the companies use NLP to improve the efficiency of documentation processes, accuracy of documentation, and identify the information from large databases.

**Disadvantages Of NLP:**

Some Disadvantages of NLP are-

* NLP may not show context.
* NLP is unpredictable.
* NLP may require more keystrokes.
* NLP is unable to adapt to the new domain, and it has a limited function that's why NLP is built for a single and specific task only.

**Natural Language Toolkit (NLTK)-**

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

**What is Tokenization?**

Tokenization is a process of splitting a text object into smaller units which are also called tokens. Examples of tokens can be words, numbers, engrams, or even symbols. The most commonly used tokenization process is **White-space Tokenization.**

**Applications Of NLP:**

## Email filters:

Email filters are one of the most basic and initial applications of NLP online. It started out with spam filters, uncovering certain words or phrases that signal a spam message. But filtering has upgraded, just like early adaptations of NLP. One of the more prevalent, newer applications of NLP is found in Gmail's email classification. The system recognizes if emails belong in one of three categories (primary, social, or promotions) based on their contents. For all Gmail users, this keeps your inbox to a manageable size with important, relevant emails you wish to review and respond to quickly.

## Search results:

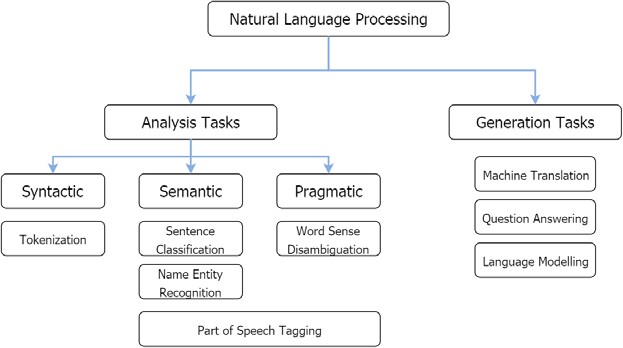
Search engines use NLP to surface relevant results based on similar search behaviors or user intent so the average person finds what they need without being a search-term wizard. For example, Google not only predicts what popular searches may apply to your query as you start typing, but it looks at the whole picture and recognizes what you’re trying to say rather than the exact search words. Someone could put a flight number in Google and get the flight status, type a ticker symbol and receive stock information, or a calculator might come up when inputting a math equation. These are some variations you may see when completing a search as NLP in search associates the ambiguous query to a relative entity and provides useful results.

And others are **Predictive Text** and **Smart Assistants** etc.

**Tasks of Natural Language Processing**

NLP has a multitude of real-world applications. A good NLP system is that which performs many NLP tasks. When you search for today's weather on Google or use Google Translate to find out how to say, "How are you?" in French, you rely on a subset of such tasks in NLP. We will list some of the most ubiquitous tasks here, and this book covers most of these tasks:

* **Tokenization:** Tokenization is the task of separating a text corpus into atomic units (for example, words). Although it may seem trivial, tokenization is an important task. For example, in the Japanese language, words are not delimited by spaces nor punctuation marks.
* **Word-sense Disambiguation (WSD):** WSD is the task of identifying the correct meaning of a word. For example, in the sentences, The dog barked at the mailman, and Tree bark is sometimes used as a medicine, the word bark has two different meanings. WSD is critical for tasks such as question answering.
* **Named Entity Recognition (NER):** NER attempts to extract entities (for example, person, location, and organization) from a given body of text or a text corpus. For example, the sentence, John gave Mary two apples at school on Monday will be transformed to *[John]name gave [Mary]name [two]number apples at [school]organization on [Monday.]time*. NER is an imperative topic in fields such as information retrieval and knowledge representation.
* **Part-of-Speech (PoS) tagging:** PoS tagging is the task of assigning words to their respective parts of speech. It can either be basic tags such as noun, verb, adjective, adverb, and preposition, or it can be granular such as proper noun, common noun, phrasal verb, verb, and so on.
* **Sentence/Synopsis classification:** Sentence or synopsis (for example, movie reviews) classification has many use cases such as spam detection, news article classification (for example, political, technology, and sport), and product review ratings (that is, positive or negative). This is achieved by training a classification model with labeled data (that is, reviews annotated by humans, with either a positive or negative label).
* **Language generation:** In language generation, a learning model (for example, neural network) is trained with text corpora (a large collection of textual documents), which predict new text that follows. For example, language generation can output an entirely new science fiction story by using existing science fiction stories for training.
* **Question Answering (QA):** QA techniques possess a high commercial value, and such techniques are found at the foundation of chatbots and VA (for example, Google Assistant and Apple Siri). Chatbots have been adopted by many companies for customer support. Chatbots can be used to answer and resolve straightforward customer concerns (for example, changing a customer's monthly mobile plan), which can be solved without human intervention. QA touches upon many other aspects of NLP such as information retrieval, and knowledge representation. Consequently, all this makes developing a QA system very difficult.



**Figure 2: A taxonomy of the popular tasks of NLP categorized under broader categories**

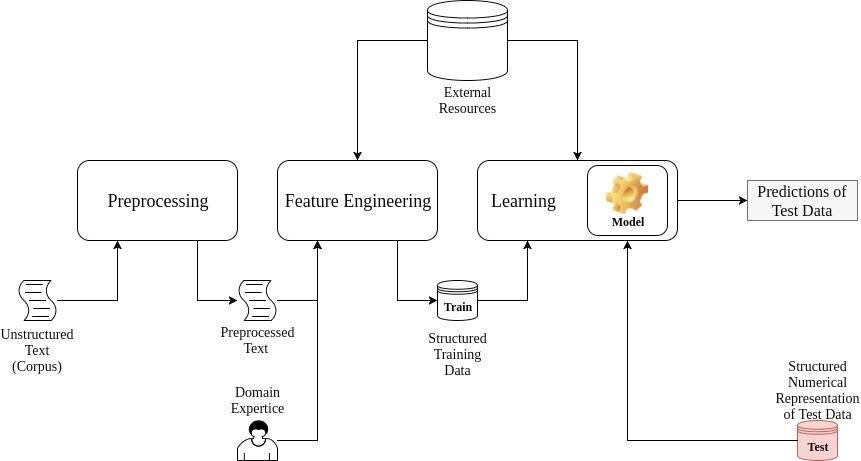
**The conventional approach to Natural Language Processing**

The traditional or classical approach to solving NLP is a sequential flow of several key steps, and it is a statistical approach. When we take a closer look at a traditional NLP learning model, we will be able to see a set of distinct tasks taking place, such as preprocessing data by removing unwanted data, feature engineering to get good numerical representations of textual data, learning to use machine learning algorithms with the aid of training data, and predicting outputs for novel unfamiliar data. Of these, feature engineering proves out to be the most time-consuming and crucial step for obtaining good performance on a given NLP task.

**Understanding the traditional approach**

The traditional approach to solving NLP tasks involves a collection of distinct subtasks. First, the text corpora need to be preprocessed focusing on reducing the vocabulary and *distractions*. By *distractions*, I refer to the things that distract the algorithm (for example, punctuation marks and stop word removal) from capturing the vital linguistic information required for the task.

Furthermore, several feature engineering steps. The main objective of feature engineering is to make learning easier for the algorithms. Often the features are hand-engineered and biased toward the human understanding of a language. Feature engineering was of utter importance for classical NLP algorithms, and consequently, the best performing systems often had the best engineered features. For example, for a sentiment classification task, you can represent a sentence with a parse tree and assign positive, negative, or neutral labels to each node/subtree in the tree to classify that sentence as positive or negative. Additionally, the feature engineering phase can use external resources such as WordNet (a lexical database) to develop better features. We will soon look at a simple feature engineering technique known as *bag-of-words*.



**Figure 3: The general approach of classical NLP**

Understanding a simple deep model – a Fully-Connected Neural Network

Now let's dig a little deeper in the deep neural network in order to gain a better understanding. Although there are numerous different variants of deep models, let's look at one of the earliest models (dating back to 1950-60), known as a **Fully-Connected Neural Network** (**FCNN**), or sometimes called a multilayer perceptron.

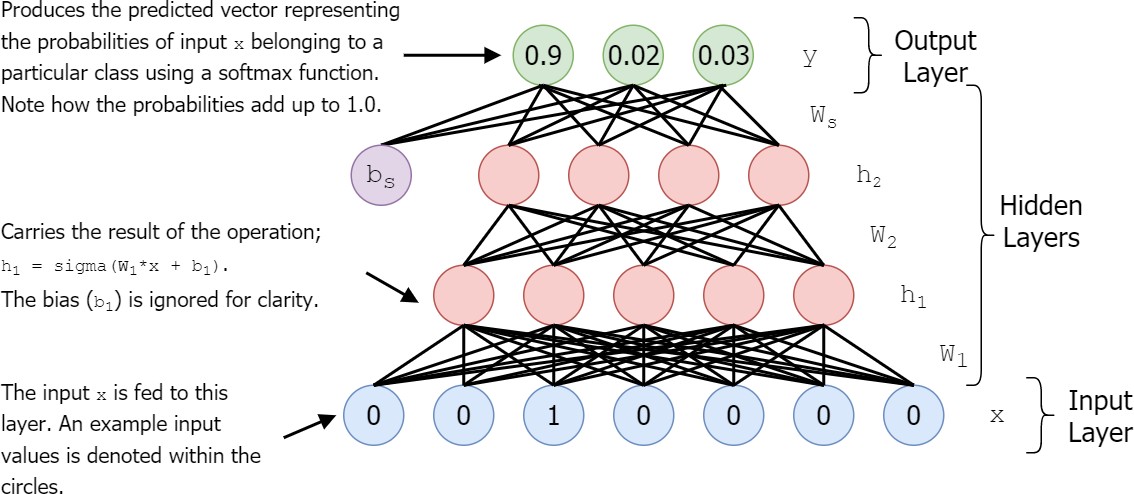
The goal of a FCNN is to map an input (for example, an image or a sentence) to a certain label or annotation (for example, the object category for images). This is achieved by using an input *x* to compute *h*—a hidden representation of *x*—using a transformation such as *h = sigma (W \* x + b)*; here, *W* and *b* are the weights and bias of the FCNN, respectively, and *sigma* is the sigmoid activation function. Next, a classifier (for example, a Softmax classifier) is placed on top of the FCNN that gives the ability to leverage the learned features in hidden layers to classify inputs. Classifier, essentially is a part of the FCNN and yet another hidden layer with some weights, *Ws* and a

bias, *bs*. Also, we can calculate the final output of the FCNN as,

***output = softmax (Ws\*h + bs)*.**

For example, a Softmax classifier provides a normalized representation of the scores output by the classifier layer; the label is considered to be the output node withthe highest softmax value. Then, with this, we can define a classification loss that is calculated as the difference between the predicted output label and the actual output label. An example of such a loss function is the mean squared loss. You don't have to worry if you don't understand the actual intricacies of the loss function. We will discuss quite a few of them in later chapters. Next, the neural network parameters, *W*, *b*, *Ws*, and *bs*, are optimized using a standard stochastic optimizer (for example, the stochastic gradient descent) to reduce the classification loss all the inputs.

It depicts the process explained in this paragraph for a three-layer FCNN. We will walk-through the details on how to use such a model for **NLP tasks**, ***Word2vec – Learning Word Embeddings***.



**Figure 4: A structure of the Fully Connected Neural Network (FCNN)**

Installing Python and scikit-learn

Python is hassle-free to install in any of the commonly used operating systems such as Windows, macOS, or Linux. We will use Anaconda to set up Python, as it does all the laborious work for setting up Python as well as the essential libraries.

To install Anaconda, follow these steps:

1. Download Anaconda from <https://www.continuum.io/downloads>
2. Select the appropriate OS and download Python 3.5
3. Install Anaconda by following the instructions at [https://docs. continuum.io/anaconda/install/](https://docs.continuum.io/anaconda/install/)

To check whether Anaconda was properly installed, follow these steps:

1. Open a Terminal window (Command Prompt in Windows)
2. Now, run the following command:

**conda --version**

If installed properly, the version of the current Anaconda distribution should be shown in Terminal.

[Next, install scikit-learn by following](http://scikit-learn.org/stable/install.html) the instructions at [**http://scikit-learn.**](http://scikit-learn.org/stable/install.html)[**org/stable/install.html**](http://scikit-learn.org/stable/install.html),

NLTK from[**https://www.nltk.org/install.html**](https://www.nltk.org/install.html)and

Matplotlib from [**https://matplotlib.org/users/installing.html**](https://matplotlib.org/users/installing.html).

Installing Jupyter Notebook

You can install Jupyter Notebook by following the instruction at [http://jupyter.](http://jupyter.readthedocs.io/en/latest/install.html) [readthedocs.io/en/latest/install.html](http://jupyter.readthedocs.io/en/latest/install.html).

To check whether Jupyter Notebook is properly installed, follow these steps:

1. Open a Terminal window
2. Run this command:

**jupyter notebook**



**Figure 6: An Insights on the successful installation of the Jupyter Notebook**

Installing TensorFlow

Follow the instructions given at [**https://www.tensorflow.org/install/**](http://www.tensorflow.org/install/) under the *Installing with Anaconda* subsection to install TensorFlow. We will use TensorFlow

1.8.x throughout all the exercises.

When providing the tfBinaryURL as asked in the instruction, make sure that you provide a TensorFlow 1.8.x version. We stress this as the API has undergone many changes compared to the previous TensorFlow versions.

To check whether TensorFlow installed properly, follow these steps:

1. Open Command Prompt in Windows or Terminal in Linux or macOS.
2. Type python to enter the Python environment. You should now see the Python version right below. Make sure that you are using Python 3.
3. Next, enter the following commands:

**import tensorflow as tf**

**print(tf.version)**

What is TensorFlow?

TensorFlow is an open source distributed numerical computation framework released by Google that is mainly intended to alleviate the painful details of implementing a neural network (for example, computing derivatives of the weights of the neural network). TensorFlow takes this even a step further by providing efficient implementations of such numerical computations using **Compute Unified Device Architecture** (**CUDA**), which is a parallel computational platform introduced by NVIDIA. The **Application Programming Interface** (**API**) of TensorFlow [at https://www.tensorflow.org/](https://www.tensorflow.org/api_docs/python/) [api\_docs/python/](https://www.tensorflow.org/api_docs/python/) shows that TensorFlow provides thousands of operations that make our lives easier.

TensorFlow was not developed overnight. This is a result of the persistence of talented, good-hearted individuals who wanted to make a difference by bringing deep learning to a wider audience. If you are interested, you can take a look at the TensorFlow code [at https://github.com/tensorflow/tensorflow](https://github.com/tensorflow/tensorflow). Currently, TensorFlow has around 1,000 contributors, and it sits on top of more than 25,000 commits, evolving to be better and better every day.

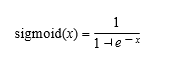
Getting started with TensorFlow

Now let's learn about a few essential components in the TensorFlow framework by working through a code example. Let's write an example to perform the following computation, which is very common for neural networks:

**h = sigmoid (W \* x + b)**

Here W and x are matrices and b is a vector. Then, \* denotes the dot product.

Sigmoid is a non-linear transformation given by the following equation:



import tensorflow as tf

print(tf.version)

Defining TensorFlow outputs

TensorFlow outputs are usually tensors and a result of a transformation to either an input or a variable or both. In our example, h is an output were,

**h = tf.nn. sigmoid(tf.matmul(x,W) + b)**

It is also possible to give such outputs to others

operations, forming a chained set of operations. Furthermore, it does not necessarily have to be TensorFlow operations. You also can use standard Python arithmetic with TensorFlow. Here is an example:

x = tf.matmul(w,A) y = x + B

z = tf.add(y,C)

Implementing our first neural network

Great! Now that you've learned the architecture, basics, and scoping mechanism of TensorFlow, it's high time that we move on and implement something moderately complex. Let's implement a neural network. To be precise, we will implement a fully connected neural network model. One of the stepping stones to the introduction of neural networks is to implement a neural network that is able to classify digits. For this task, we will be using the famous MNIST dataset [made available at http://yann.lecun.com/exdb/mnist/.](http://yann.lecun.com/exdb/mnist/)

You might feel a bit skeptical regarding our using a computer vision task rather than a NLP task. However, vision tasks can be implemented with less preprocessing and are easy to understand.

As this is our first encounter with neural networks, we will walk through the main parts of the example. However, note that I will only walk through the crucial bits of the exercise.

Preparing the data

First, we need to download the dataset with the maybe\_download(...) function and preprocess it with the read\_mnist(...) function. These two functions are defined in the exercise file. The read\_mnist(...) function performs two main steps:

* Reading the byte stream of the dataset and forming it into a proper

numpy.ndarray object

* Standardizing the images to have a zero-mean and unit-variance (also known as **whitening**)

The following code shows the read\_mnist(...) function. The read\_mnist(...) function takes the filename of the file containing images and the filename of the file containing labels, as input. Then the read\_mnist(...) function produces two NumPy matrices containing all the images and their corresponding labels:

def read\_mnist(fname\_img, fname\_lbl):

print('\nReading files %s and %s'%(fname\_img, fname\_lbl))

with gzip.open(fname\_img) as fimg:

magic, num, rows, cols = struct.unpack(">IIII", fimg.read(16)) print(num,rows,cols)

img = (np.frombuffer(fimg.read(num\*rows\*cols), dtype=np.uint8). reshape(num, rows \* cols)).astype(np.float32)

print('(Images) Returned a tensor of shape ',img.shape) # Standardizing the images

img = (img - np.mean(img))/np.std(img)

with gzip.open(fname\_lbl) as flbl: # flbl.read(8) reads upto 8 bytes

magic, num = struct.unpack(">II", flbl.read(8)) lbl = np.frombuffer(flbl.read(num), dtype=np.int8)

print('(Labels) Returned a tensor of shape: %s'%lbl.shape)

print('Sample labels: ',lbl[:10])

return img, lbl

Defining the TensorFlow graph

To define the TensorFlow graph, we'll first define placeholders for the input images (tf\_inputs) and the corresponding labels (tf\_labels):

# Defining inputs and outputs

tf\_inputs = tf.placeholder(shape=[batch\_size, input\_size], dtype=tf. float32, name = 'inputs')

tf\_labels = tf.placeholder(shape=[batch\_size, num\_labels], dtype=tf. float32, name = 'labels')

Next, we'll write a Python function that will create the variables for the first time. Note that we are using scoping to ensure the reusability, and make sure that our variables are named properly:

# Defining the TensorFlow variables

def define\_net\_parameters():

with tf.variable\_scope('layer1'):

tf.get\_variable(WEIGHTS\_STRING,shape=[input\_size,500],

initializer=tf.random\_normal\_initializer(0,0.02))

tf.get\_variable(BIAS\_STRING, shape=[500],

initializer=tf.random\_uniform\_initializer(0,0.01))

with tf.variable\_scope('layer2'):

tf.get\_variable(WEIGHTS\_STRING,shape=[500,250],

initializer=tf.random\_normal\_initializer(0,0.02))

tf.get\_variable(BIAS\_STRING, shape=[250],

initializer=tf.random\_uniform\_initializer(0,0.01))

with tf.variable\_scope('output'):

tf.get\_variable(WEIGHTS\_STRING,shape=[250,10],

initializer=tf. random\_normal\_initializer(0,0.02))

tf.get\_variable(BIAS\_STRING, shape=[10], initializer=tf.random\_ uniform\_initializer(0,0.01))

Next, we'll define the inference process for the neural network. Note how the scoping has given a very intuitive flow to the code in the function, compared with using variables without scoping. So, in this network we have three layers:

* A fully-connected layer with ReLU activation (layer1)
* A fully-connected layer with ReLU activation (layer2)
* A fully-connected softmax layer (output)

By means of scoping, we name variables (weights and biases) for each layer as, layer1/weights, layer1/bias, layer2/weights, layer2/bias, output/weights, and output/bias. Note that in the code, all of them have the same name, but different scopes:

# Defining calcutations in the neural network

# Starting from inputs to logits

# Logits are the values before applying softmax to the final output

def inference(x):

# calculations for layer 1

with tf.variable\_scope('layer1',reuse=True):

w,b = tf.get\_variable(WEIGHTS\_STRING),tf.get\_variable(BIAS\_STRING)

tf\_h1 = tf.nn.relu(tf.matmul(x,w) + b, name = 'hidden1')

# calculations for layer 2

with tf.variable\_scope('layer2',reuse=True):

w,b = tf.get\_variable(WEIGHTS\_STRING),tf.get\_variable(BIAS\_STRING)

tf\_h2 = tf.nn.relu(tf.matmul(tf\_h1,w) + b, name = 'hidden1')

# calculations for output layer

with tf.variable\_scope('output',reuse=True):

w,b = tf.get\_variable(WEIGHTS\_STRING),tf.get\_variable(BIAS\_STRING)

tf\_logits = tf.nn.bias\_add(tf.matmul(tf\_h2,w), b, name = 'logits')

return tf\_logits

Now we'll define a loss function and then a loss minimize operation. The loss minimizes the loss by nudging the network parameters in the direction that minimizes the loss. There is a diverse collection of optimizers available in TensorFlow. Here, we will be using **MomentumOptimizer**, which gives better final accuracy and convergence than **GradientDescentOptimizer**:

# defining the loss

tf\_loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_ v2(logits=inference(tf\_inputs), . labels=tf\_labels))

# defining the optimize function

tf\_loss\_minimize = tf.train.MomentumOptimizer(momentum=0.9,

learning\_rate=0.01).minimize(tf\_loss)

Finally, we'll define an operation to retrieve the predicted softmax probabilities for a given batch of inputs. This in turn will be used to calculate the accuracy of your neural network:

# defining predictions

tf\_predictions = tf.nn.softmax(inference(tf\_inputs))

Running the neural network

Now we have all the essential operations required to run the neural network and examine whether it's capable of learning to successfully classify digits:

for epoch in range(NUM\_EPOCHS): train\_loss = []

**# Training Phase**

for step in range(train\_inputs.shape[0]//batch\_size):

# Creating one-hot encoded labels with labels

# One-hot encoding digit 3 for 10-class MNIST dataset # will result in

# [0,0,0,1,0,0,0,0,0,0]

labels\_one\_hot = np.zeros((batch\_size, num\_labels),

dtype=np.float32)

labels\_one\_hot[np.arange(batch\_size),

train\_labels[ step\*batch\_size:(step+1)\*batch\_size]] = 1.0

# Running the optimization process

loss, \_ = session.run([tf\_loss,tf\_loss\_minimize],feed\_dict={ tf\_inputs: train\_inputs[step\*batch\_size: (step+1)\*batch\_size,:],

tf\_labels: labels\_one\_hot})

train\_loss.append(loss)

# Used to average the loss for a single epoch

test\_accuracy = [] # Testing Phase

for step in range(test\_inputs.shape[0]//batch\_size):

test\_predictions = session.run(tf\_predictions,feed\_dict={tf\_

inputs: test\_inputs[step\*batch\_size: (step+1)\*batch\_size,:]})

batch\_test\_accuracy = accuracy(test\_predictions,

test\_ labels[step\*batch\_size: (step+1)\*batch\_size])

test\_accuracy.append(batch\_test\_accuracy)

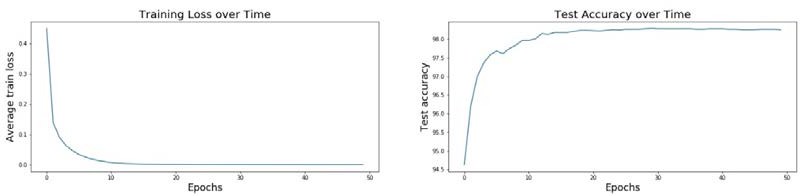
print('Average train loss for the %d epoch: %.3f\n'%(epoch+1,np. mean(train\_loss)))

print('\tAverage test accuracy for the %d epoch:

%.2f\n'%(epoch+1,np.mean(test\_accuracy)\*100.0))

In this code, accuracy(test\_predictions,test\_labels) is a function that takes some predictions and labels as inputs and provides the accuracy (how many predictions matched the actual label). If successful, you should be able to see a behavior similar to the ones shown in figure below.

**Figure 7: Training loss and test accuracy for the MNIST digit classification task**



After 50 epochs, the test accuracy should reach approximately 98%.