Chapter 4: Introduction to Neural Network using PyTorch

**Introduction**

Deep Neural Network based models gradually becoming backbone for artificial intelligence and machine learning implementations. The future of data mining will be governed by usage of artificial neural network based advanced modeling techniques. One obvious question; why recently neural network is gaining so much importance, though it was invented in 1950’s. Borrowed from the computer science domain, neural networks can be defined as a parallel information processing system where the inputs relate to each other as the neurons in human brain to transmit information so that activities like face recognition, image recognition etc. can be performed. In this chapter we are going to learn about application of neural network-based methods on various data mining tasks like classification, regression, forecasting and feature reduction. Artificial Neural Network (ANN) functions in a way that is exactly similar to the way the human brain is functioning, where a billion of neurons that link to each other for information processing and insight generation.

# Introduction to neural network

Brains biological network provides basis for connecting elements in a real-life scenario for information processing and insight generation. A hierarchy of neurons connected through layers, where the output of one layer becomes the input for another layers, the information passes from one layer to another layer as weights. The weights associated with each neuron contain insights so that the recognition and reasoning become easier for the next level. Artificial neural network is a very popular and effective method that consists of layers associated with weights. The association between different layers is governed by mathematical equation that passes information from one layer to the other. In fact, a bunch of mathematical equations are at work inside one artificial neural network model. The following graph shows the general architecture for a neural network-based model.

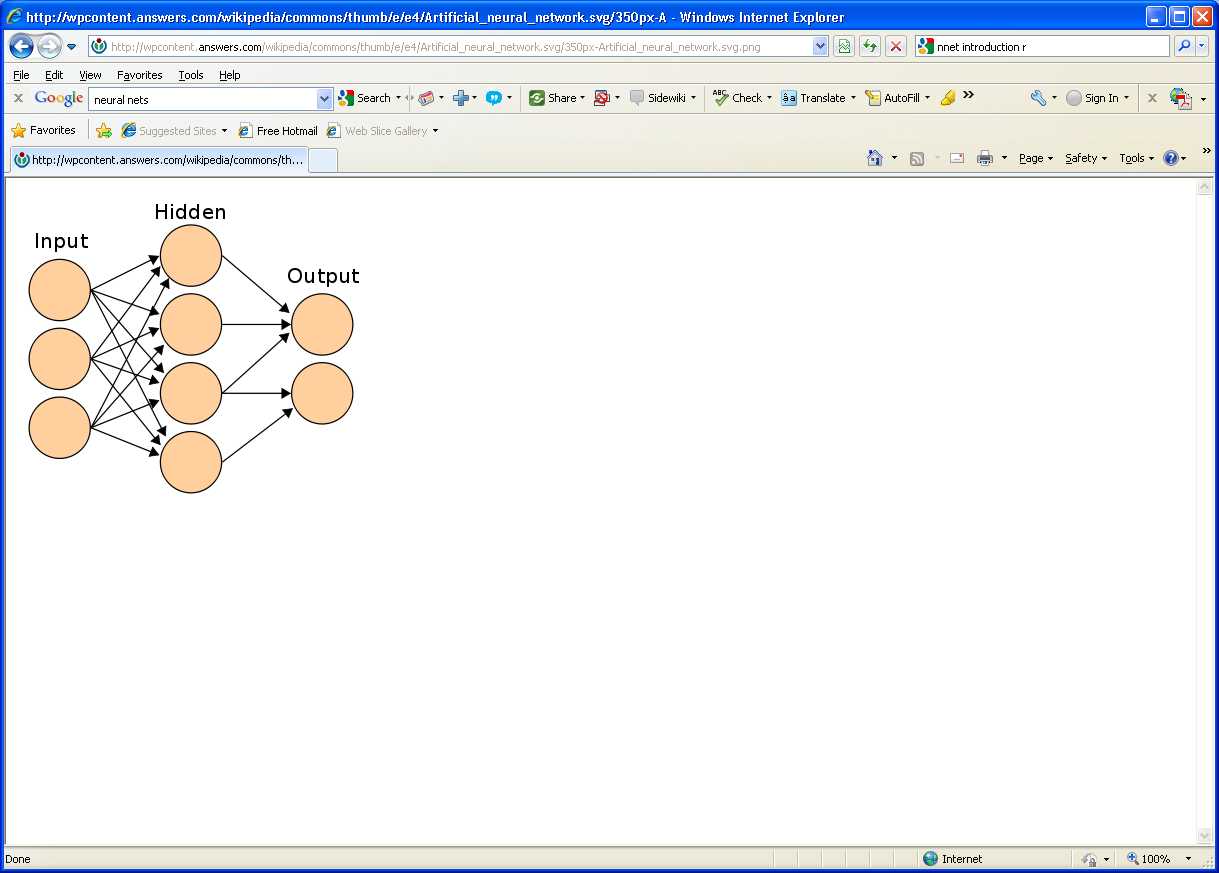


Figure 1: sample neural network structure

In the above graph there are three layers, input, hidden and output layer, which is the core of any neural network-based architecture. Artificial Neural networks (ANN) are a powerful technique to solve many real-world problems such as classification, regression and feature selection. ANNs can learn from new experiences in the form of new input data to improve the performance of classification or regression-based task and to adapt themselves to changes in the input environment. Each circle in the above figure represents a neuron.

There are different variants of neural networks that are used in multiple different scenarios, we are going to explain a few of them conceptually below so that we can understand their usage in practical applications.

• Single hidden layer neural network: this is the simplest form of neural network as it is show in the figure 1 above. In this there is only one hidden layer.

• Multiple hidden layer neural networks: in this form more than one hidden layer will connect the input data with the output data. The complexity of calculation increases in this form as it requires more computational power to the system to process information

• Feed forward neural networks: in this form of neural network architecture, the information is passed one directionally from one layer to another layer; there is no iteration from the first level of learning.

• Back propagation neural networks: in this form of neural network there are two important steps, feed forward works in passing information from input to the hidden and from hidden to output layer and secondly it calculates error and propagate it back to the previous layers.

The feed forward neural network model architecture is given below in figure 2 and back propagation method is explained below in figure3:

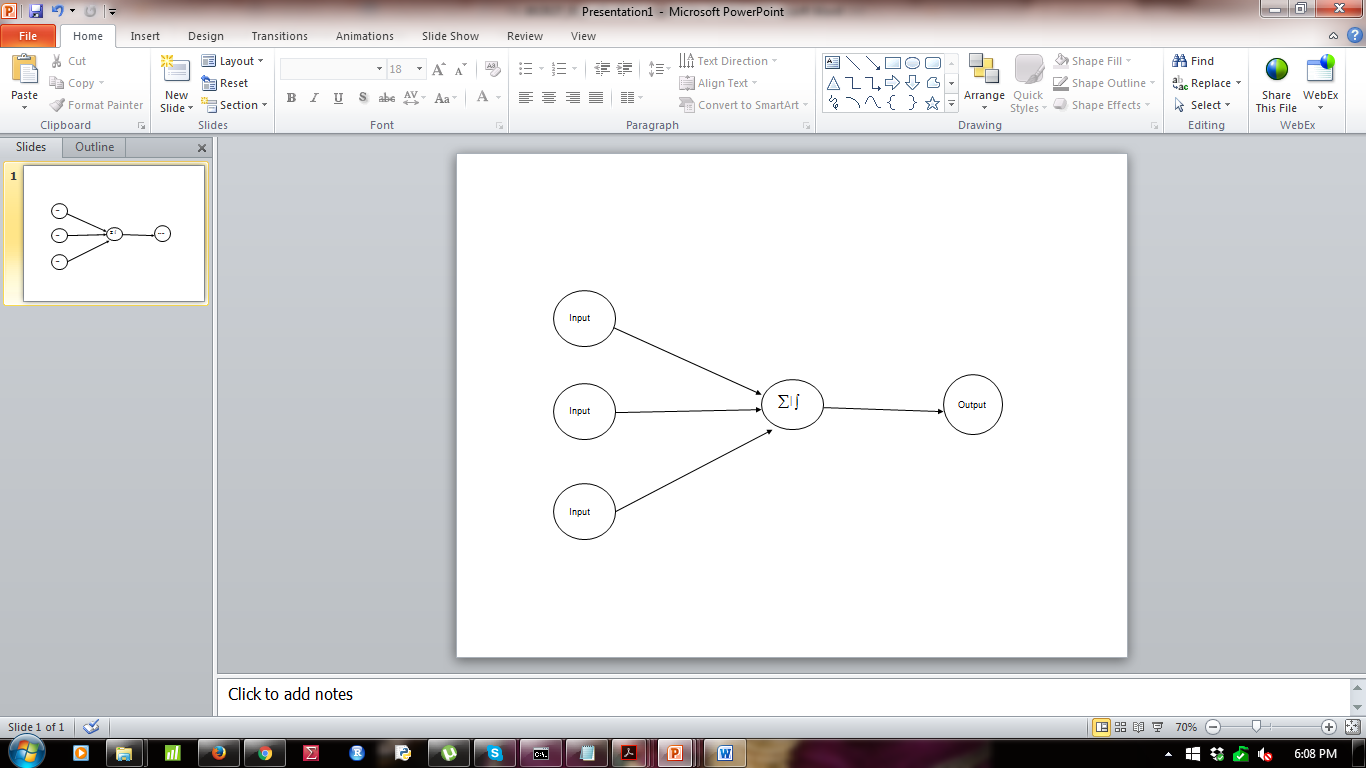


Figure 2: a sample feed forward neural network

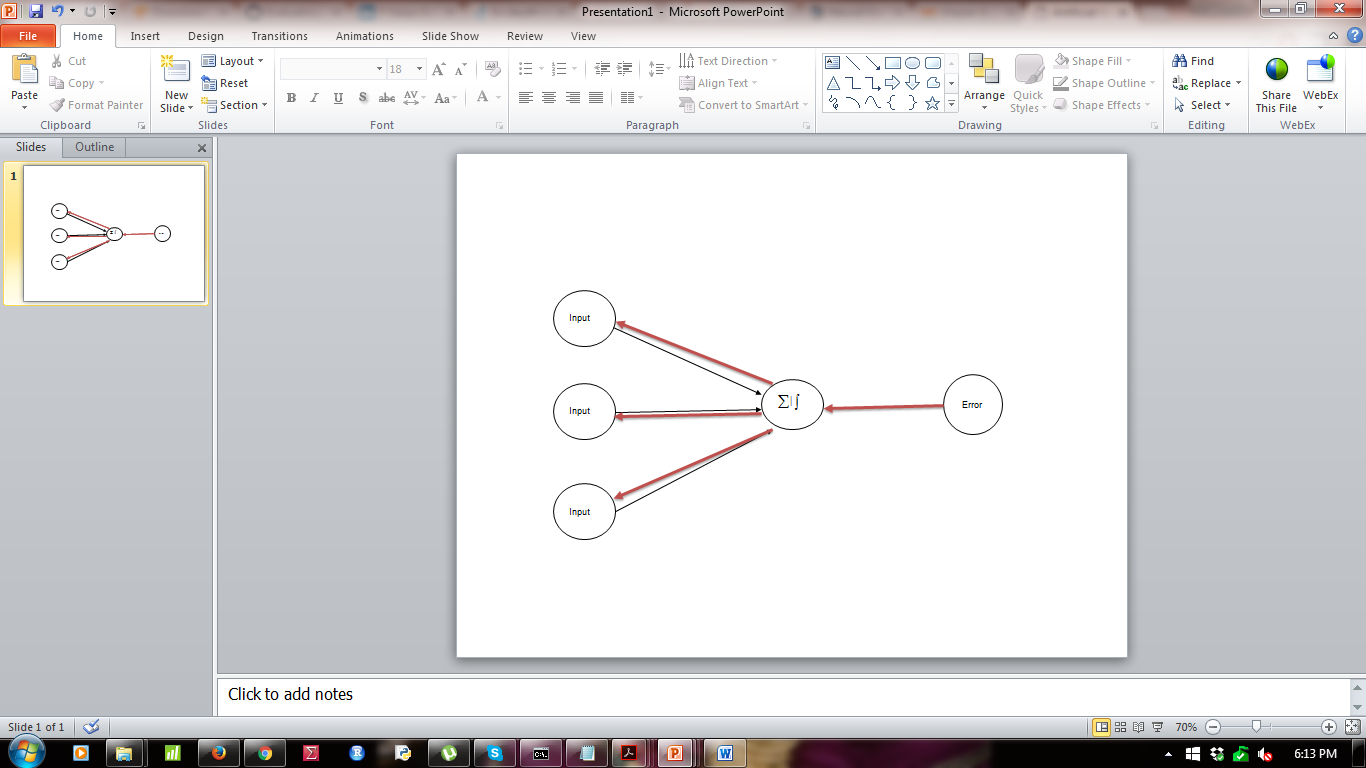


Figure 3: back propagation explained.

In the above figure the red colored arrows indicate the information that has not passed through the output layer are again fed back to the input layer in terms of errors. Having discussed the general architecture for different types of neural networks, let’s visit the underlying math behind the neural networks.

## Problem:

What are the activation functions and how do they work in real projects? How do you implement an activation function using PyTorch?

## Solution:

The neurons present in different layers; input, hidden and output are interconnected through a mathematical function called activation function. There are different variants of activation function which are explained below. Understanding the activation function would help in implementation of neural network model for better accuracy.

## How It works:

All the activation functions that are part of a neural network model can be broadly classified as linear functions and non-linear functions. In PyTorch torch.nn module is used to create any type neural network model. Let’s have some examples of deployment of activation functions using PyTorch and torch.nn module.

**Linear function:**

this is one of the simple functions typically used to transfer information from de-mapping layer to the output layer. The formula is given below.

**Bi-Linear function:**

this is one of the simple functions typically used to transfer information, it Applies a bilinear transformation to the incoming data:

A screenshot of a cell phone

Description generated with very high confidence

A screenshot of a cell phone

Description generated with very high confidence

**Sigmoid function:**

it is frequently used by professionals in data mining and analytics as it is easier to explain and implement too. The equation is mentioned below.



A screenshot of a cell phone

Description generated with high confidence

**Hyperbolic tangent function:**

this is another variant of transformation function which is used to transform information from mapping layer to the hidden layer.



A screenshot of a cell phone

Description generated with very high confidence

A close up of text on a white background

Description generated with very high confidence

**Log sigmoid transfer function:**

the following formula explains the log sigmoid transfer function to be used in mapping input layer to the hidden layer.



A screenshot of a cell phone

Description generated with very high confidence

A screenshot of text

Description generated with very high confidence

**ReLU Function:**

Rectified linear unit is another activation function which is used in transferring information from input layer to the output layer. This is mostly used between different hidden layers in a neural network model.

A screenshot of a cell phone

Description generated with very high confidence

A screenshot of text

Description generated with very high confidence

Different types of transfer functions as discussed above can be interchangeable in neural network architecture. They can be used in different stages like input to hidden layer, hidden layer to output layer etc. to improve the model accuracy.

**Leaky ReLU:**

In a standard neural network model dying gradient problem is a common thing, to avoid this issue, leaky ReLU is applied. Leaky ReLU allow a small and non-zero gradient when the unit is not active.

A screenshot of a cell phone

Description generated with very high confidence

A screenshot of text

Description generated with very high confidence

## Problem:

How do you visualize the activation functions? Visualization of activation functions are really important to correctly build a neural network model.

## Solution:

The activation functions translate the data, information from one layer into another layer, the transformed data again can be plotted against the actual tensor to visualize the function. We have taken a sample tensor, converted to a PyTorch variable, applied the function and store it as another tensor. Represent the actual tensor and the transformed tensor using matplotlib.

## How It works:

The right choice of activation function will not only provide better accuracy but also it will help is extracting meaningful information.

A screenshot of a cell phone

Description generated with very high confidence

In the above script we have taken an array in the linear space in between plus and minus 10 and we have taken 1500 sample points. Converted the vector to a torch variable, then taken a copy as numpy variable for plotting the graph. Nect we have calculated the activation functions. The following images will show the activation functions.

A screenshot of a cell phone

Description generated with very high confidence

A screenshot of a cell phone

Description generated with high confidence

A screenshot of a cell phone

Description generated with high confidence

A screenshot of a cell phone

Description generated with high confidence

## Problem:

How to build a basic neural network model using PyTorch.

## Solution:

Basic Neural Network Model in PyTorch requires six different steps such as preparation of training data, initialization of weights, creating a basic network model, calculating loss function, selecting the learning rate and finally optimizing the loss function with respect to the parameters of the model.

## How It works:

Let’s follow a step by step approach to create a basic neural network model as stated above.

A screenshot of a cell phone

Description generated with very high confidence

A screenshot of a cell phone

Description generated with high confidence

A screenshot of a cell phone

Description generated with high confidence

A screenshot of a cell phone

Description generated with high confidence

A close up of a map

Description generated with very high confidence

Median, mode and standard deviation Computation also can be written in the sa

Standard deviation shows the deviation from the measures of central tendency, which is an indication of consistency of data/variable. It shows whether there is enough fluctuation in data or not.

## Problem:

What is tensor differentiation and how it is relevant in case of a computational graph execution using PyTorch framework.

## Solution:

The computational graph network is represented by nodes and connected through functions. There are two different kind of nodes, dependent and independent. Dependent nodes are waiting for results from other nodes to process the input. Independent nodes are connected and are either constants or the results. Tensor differentiation is an efficient method to perform computation in a computational graph environment.

## How It works:

In a computational graph the tensor differentiation is very much effective because, the tensors can be computed as parallel nodes, as multi-process or as multi-threading. The major deep learning and neural computation frameworks include this tensor differentiation. Autograd is the function that helps in performing tensor differentiation, which is calculating the gradients or slope of the error function, back-propagate the error through the neural network to finetune the weights and biases and through learning rate and iteration, it tries to reduce the error value or loss function.

To apply tensor differentiation nn.backward() method need to be applied. Let’s take an example and see how the error gradients are back propagated.

A screenshot of a cell phone

Description generated with high confidence

In the above script, the x is a sample tensor, for which automatic gradient calculation need to happen. The fn is a linear function that is created using the x variable. Using the backward function, we can perform back propagation calculation. The .grad() function will hold the final output from the tensor differentiation.

**Conclusion:**