S21CSE471: Statistical Methods in AI **Spring 2021**

Introduction To Neural Networks

Prepared by: Vivek Talwar , Vinaya Bachu , Yagyesh Purohit

1 Introduction

Also referred to as **artificial neural networks**, or just **neural nets**, the neural networks are just one of many tools and approaches used in machine learning algorithms, and are being applied to many real-life problems today, including speech and image recognition, intelligent searching, financial forecasting, and medical diagnosis, to name a few. Neural networks form an integral part of deep learning.
A neural network is a computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form.
The concept of the artificial neural network was inspired by human biology and the way neurons of the human brain function together to understand inputs from human senses. In our brain, there are billions of cells called neurons, which processes information in the form of electric signals. External information/stimuli is received by the dendrites of the neuron, processed in the neuron cell body, converted to an output and passed through the Axon to the next neuron. The next neuron can choose to either accept it or reject it depending on the strength of the signal. The following image shows the anatomy of a biological neuron.

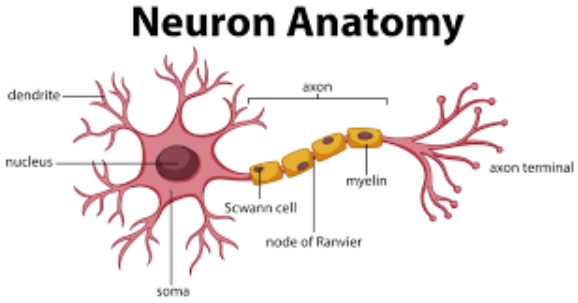


Figure 1: Anatomy of a biological neuron

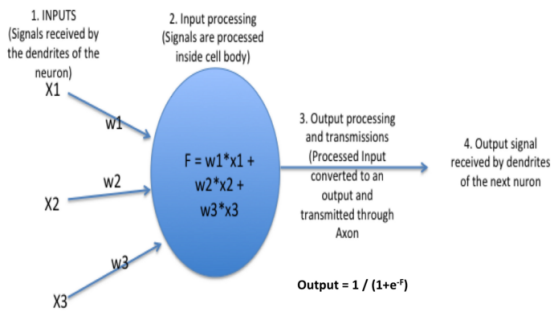


Figure 2: Artificial neuron analogy with biological neuron

1.1 Neural Networks Timeline

1. **1943**: Warren McCulloch and Walter Pitts wrote a paper on how neurons might work. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.

2. **1958**: Rosenblatt submitted a paper entitled "The Perceptron: A Perceiving and Recognizing Automaton", in which he showed that his perceptron had true learning capabilities to do linear binary classification on its own.

3. **1959**: Bernard Widrow and Marcian Hoff of Stanford developed models called "ADALINE" and "MADALINE", the first neural network model to successfully solve a real world problem.

4. **1960**: First ever version of backpropagation model given by Henry J. Kelley in his paper, "Gradient Theory of Optimal Flight Paths".

5. **1965**: Alexey Grigoryevich Ivakhnenko along with Valentin Grigorevich Lapa, creates hierarchical representation of neural network that uses polynomial activation function.

6. **1969**: Marvin Minsky and Seymour Papert published the book "Perceptrons" in which they show that Rosenblatt's perceptron cannot solve complicated functions like XOR. For such function perceptrons should be placed in multiple hidden layers which compromises the perceptron learning algorithm. This setback triggers the winter of neural network research.

7. **1982**: John Hopfield of Caltech creates Hopfield Network, a recurrent neural network. This would turn out to be instrumental for further RNN models of the modern deep learning era. In the same year, Paul Werbos, based on his 1974 Ph.D. thesis, publicly proposed the use of Backpropagation for propagating errors during the training of Neural Networks. This eventually led to the practical adoption of backpropagation by the neural network community in the future.

8. **1985**: Boltzmann machine created by David H. Ackley, Geoffrey Hinton and Terrence Sejnowski. It was a stochastic recurrent neural network.

9. **1989**: Yann LeCun used backpropagation to train convolutional neural networks to recognize handwritten digits. This was a breakthrough moment as it laid the foundation of modern computer vision using deep learning.

10. **2008**: Andrew NG's group in Stanford starts advocating for the use of GPUs for training Deep Neural Networks to speed up the training time by many folds. This could bring practicality in the field of Deep Learning for training on huge volumes of data efficiently.

11. **2009**: Fei-Fei Li, a professor at Stanford, launched ImageNet which is a database of 14 million labeled images. It would serve as a benchmark for the deep learning researchers.

12. **2012**: AlexNet, a GPU implemented CNN model designed by Alex Krizhevsky, won Imagenet's image classification contest with accuracy of 84

2 How Neural Networks Work

2.1 Simple Neural Network Architecture

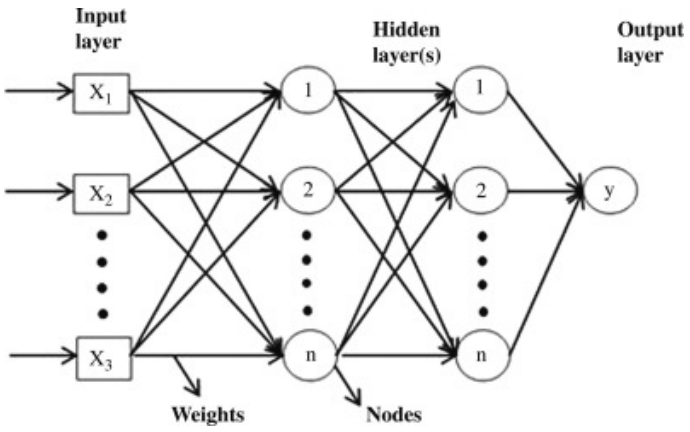


Figure 3: Simple neural-net architecture

- In the diagram given above, the outermost layer is the **input layer**. A neuron is the basic unit of a neural network. They receive input from an external source or other nodes. Each node is connected with another node from the next layer, and each such connection has a particular **weight**. Weights are assigned to a neuron based on its relative importance against other inputs.
- The layer or layers hidden between the input and output layer is known as the **hidden layer**. It is called the hidden layer since it is always hidden from the external world.
- When all the node values from the input layer are multiplied (along with their weight) and summarized, it generates a value for the first hidden layer. Based on the summarized value, this layer has a predefined **activation function** that determines whether or not this node will be "activated" and how "active" it will be.
- So, the hidden layer takes all the inputs from the input layer and performs the necessary calculation to generate a result, which is then forwarded to the **output layer**.

2.2 Forward and Backward Propagation

2.2.1 Forward Propagation

Forward propagation (or forward pass) refers to the calculation and storage of intermediate variables (including outputs) for a neural network in order from the input layer to the output layer. Forward propagation sequentially calculates and stores intermediate variables within the computational graph defined by the neural network. It proceeds from the input to the output layer.
Let us assume that the input example is $\mathbf{x} \in \mathbb{R}^d$ and that our hidden layer does not include a bias term. Here the intermediate variable is:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x},$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ is the weight parameter of the hidden layer. After running the intermediate variable $\mathbf{z} \in \mathbb{R}^h$ through the activation function ϕ, we obtain our hidden activation vector of length h,

$$\mathbf{h} = \phi(\mathbf{z}).$$

The hidden variable h is also an intermediate variable. Assuming that the parameters of the output layer only possess a weight of $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$, we can obtain an output layer variable with a vector of length q :

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}.$$

Assuming that the loss function is l and the example label is y , we can then calculate the loss term L for a single data example,

$$L = l(\mathbf{o}, y).$$

Finally, we have our model's loss or cost function J as:

$$J = L = l(\mathbf{o}, y).$$

2.2.2 Backward Propagation

Backward propagation(or backpropagation) refers to the method of calculating the gradient of neural network parameters. The method traverses the network in reverse order, from the output to the input layer, according to the chain rule from calculus. The algorithm stores any intermediate variables (partial derivatives) required while calculating the gradient with respect to some parameters.
Assume that we have functions $\mathsf{Y} = f(\mathsf{X})$ and $\mathsf{Z} = g(\mathsf{Y})$. By using the chain rule, we can compute the derivative of Z with respect to X as:

$$\frac{\partial \mathsf{Z}}{\partial \mathsf{X}} = \text{prod}\left(\frac{\partial \mathsf{Z}}{\partial \mathsf{Y}}, \frac{\partial \mathsf{Y}}{\partial \mathsf{X}}\right).$$

Here we use the prod operator to multiply its arguments after the necessary operations, such as transposition and swapping input positions, have been carried out. For vectors, this is straightforward: it is simply matrix-matrix multiplication. For higher dimensional tensors, we use the appropriate counterpart.

For a simple network with one hidden layer, the objective of backpropagation is to calculate the gradients $\partial J/\partial \mathbf{W}^{(1)}$ and $\partial J/\partial \mathbf{W}^{(2)}$, where $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ are parameters of this network. To accomplish this, we apply the chain rule and calculate, in turn, the gradient of each intermediate variable and parameter. The order of calculations are reversed relative to those performed in forward propagation. The first step is to calculate the gradients of the cost function J with respect to the loss term L

$$\frac{\partial J}{\partial L} = 1$$

Next, we compute the gradient of the cost function with respect to variable of the output layer o according to the chain rule:

$$\frac{\partial J}{\partial \mathbf{o}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}\right) = \frac{\partial L}{\partial \mathbf{o}} \in \mathbb{R}^q.$$

Now we are able to calculate the gradient $\partial J/\partial \mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$ of the model parameters closest to the output layer. Using the chain rule yields:

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{W}^{(2)}}\right) = \frac{\partial J}{\partial \mathbf{o}} \mathbf{h}^\top$$

To obtain the gradient with respect to $\mathbf{W}^{(1)}$ we need to continue backpropagation along the output layer to the hidden layer. The gradient with respect to the hidden layer's outputs $\partial J/\partial \mathbf{h} \in \mathbb{R}^h$ is given by:

$$\frac{\partial J}{\partial \mathbf{h}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{h}}\right) = {\mathbf{W}^{(2)}}^\top \frac{\partial J}{\partial \mathbf{o}}.$$

Since the activation function ϕ applies element-wise, calculating the gradient $\partial J/\partial \mathbf{z} \in \mathbb{R}^h$ of the intermediate variable z requires that we use element-wise multiplication operator, which we denote by $\odot$:

$$\frac{\partial J}{\partial \mathbf{z}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{h}}, \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right) = \frac{\partial J}{\partial \mathbf{h}} \odot \phi'(\mathbf{z}).$$

Finally, we can obtain the gradient $\partial J/\partial \mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ of the model parameters closest to the input layer. According to the chain rule, we get:

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{z}}, \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}}\right) = \frac{\partial J}{\partial \mathbf{z}} \mathbf{x}^\top$$

3 Why Neural Networks

Logistic Regression can be a good choice while dealing with linear data. It can also perform well on non linear data with limited features. However, logistic regression turns out to be a computationally expensive algorithm when we are dealing with nonlinear data consisting of quite a large number of features (millions, many times). And today, many real world machine learning problems have a very large feature-set.
For example, an image recognition dataset might consist of large resolution images, with each pixel being a feature.
Hence, for complex nonlinear dataset having many features, simple logistic regression is not a good practice of classification. This paves a way for artificial neural networks, as these algorithms provide a much better way to learn complex nonlinear hypotheses, even when the number of features is very large.

Neural networks require for their training, in comparison with other machine learning algorithms, significantly large datasets. They also need significant computational power to be trained. However, because of the excessive amount of data and computational power available today, neural networks have become increasingly common.

Another advantage of neural networks relates to their capacity to approximate unknown functions. The foundational theorem for neural networks states that a sufficiently large neural network with one hidden layer can approximate any continuously differentiable functions.

4 Activation Functions

4.1 Definition

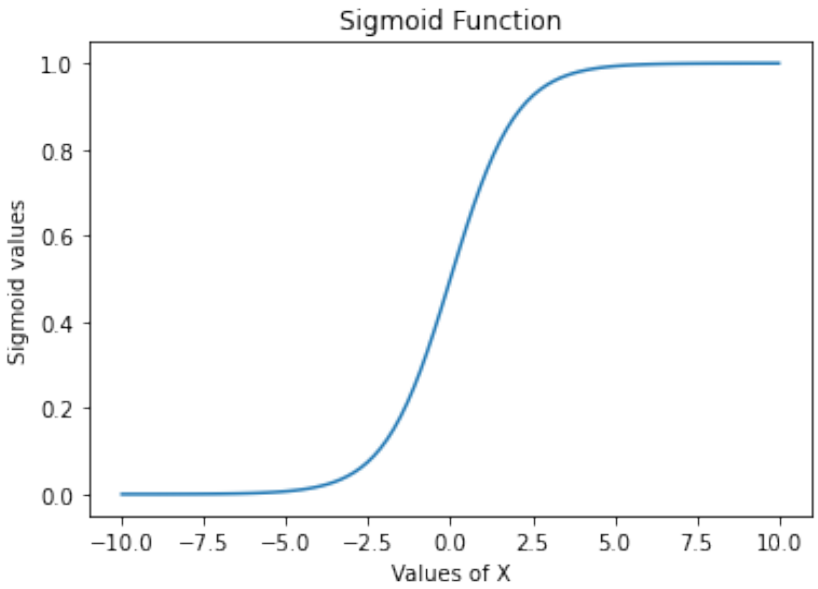
Definition 4.1. *The purpose of activation function is to decide or calculate whether to activate the Neuron by calculating the weights and adding bias given as input to the Neuron. It also adds non linearity into the output of Neuron.*

Let's discuss some functions popularly used in Neural Networks :-

4.2 Sigmoid

Definition 4.2. *Sigmoid function transforms values between 0 and 1. This is very commonly used for binary classification.*

$$h_\theta(x) = \frac{1}{1 + e^{-}\theta^T x}$$

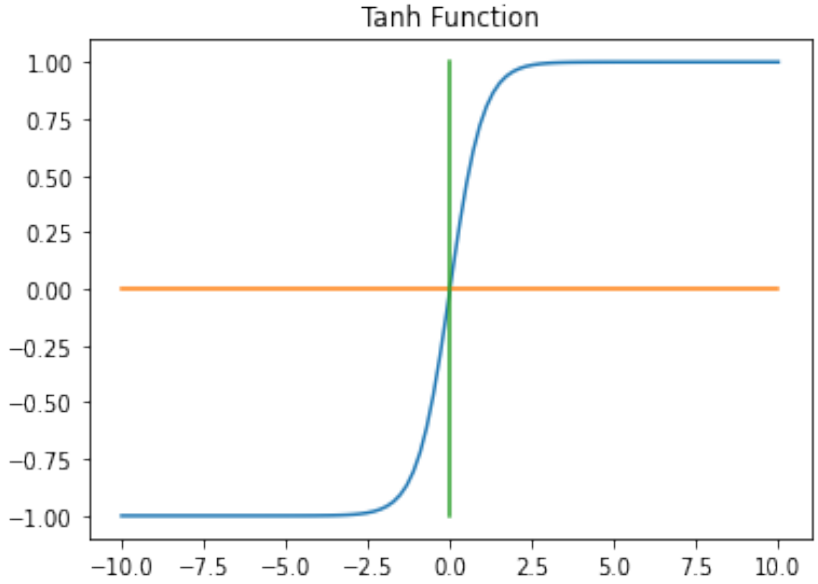


(a) One of the common problem using this activation function is "Vanishing Gradient" . When the activation reaches near the horizontal part any side of the curve , the gradient is small and the Neural Network refuses to learn. But this is still one of the most commonly used activation function.

4.3 Tanh

Definition 4.3. *The Tanh function is very similar to the sigmoid function but this is very symmetric around the origin . The values here range between -1 to 1.*

*The formula is tanh(x) = 2*sigmoid(2x) - 1*

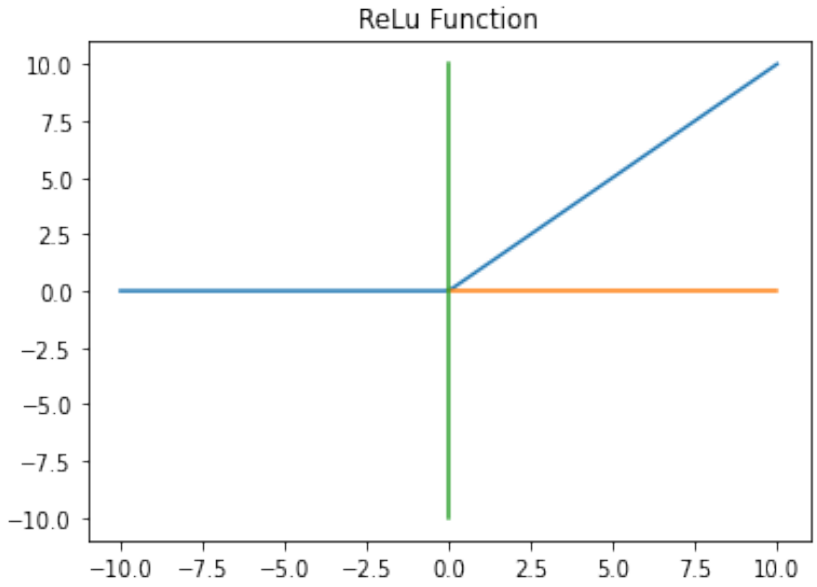


(a) Like sigmoid function this also suffers from Vanishing Gradient Descent , but the gradient is better than sigmoid.

4.4 ReLu

Definition 4.4. *The ReLu function returns the value if it is positive or returns 0 if the value is negative.*

Function is :- max(0,x)



(a) Unlike Sigmoid and Tanh it is not bounded and it ranges from 0 to infinity and it can be computed very fastly.

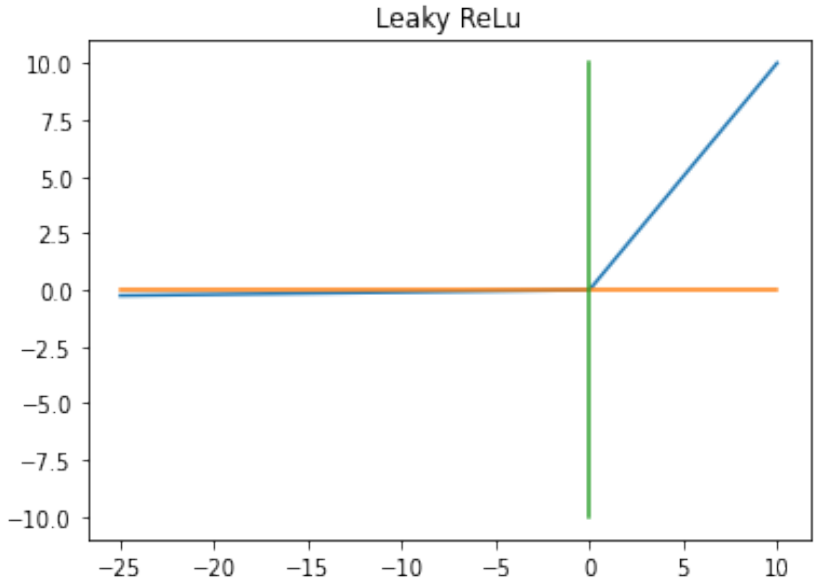
4.4.1 Dying ReLu Problem

(a) The gradient descent works pretty fine in the positive x axis but the gradient descent will be 0 and it will not learn or adjust when it is in the negative x axis.

4.5 LeakyReLu

Definition 4.5. *It is an attempt to fix the Dying ReLu problem. If the value is negative instead of returning 0 we multiply it with constant.*

(a) If we multiply it with 0.01 it is called LeakyReLu and if a is not 0.01 then it is RandomizedReLu.



4.6 Softmax

Definition 4.6. *Sigmoid function is generally used for binary classifications. When it comes to Multi class classification softmax is used . The input to the function is the output of the sigmoid function and softmax function returns the possibility of the data belonging to each individual class.*

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

5 Types Of Neural Networks

5.1 Perceptron

Definition 5.1. *A Perceptron is the simplest and the oldest model of Neural Network. It has only two layers input layer and the output layer.*

A Perceptron divides the data into two classes and is used for binary classification.

5.1.1 Advantages

(a) Easy to Train and used for binary classification

5.1.2 Disadvantages

(a) Perceptrons cannot work with Non Linear Problems

5.2 Feed-forward Neural Network

Definition 5.2. *This is one of the most naive models where the data travels from input layer to output node . There is no back propagation in this type of Neural Network.*

5.2.1 Advantages

(a) Easy to design and less complex.

(b) Works very well for noisy data.

5.2.2 Disadvantages

(a) No Back Propagation.

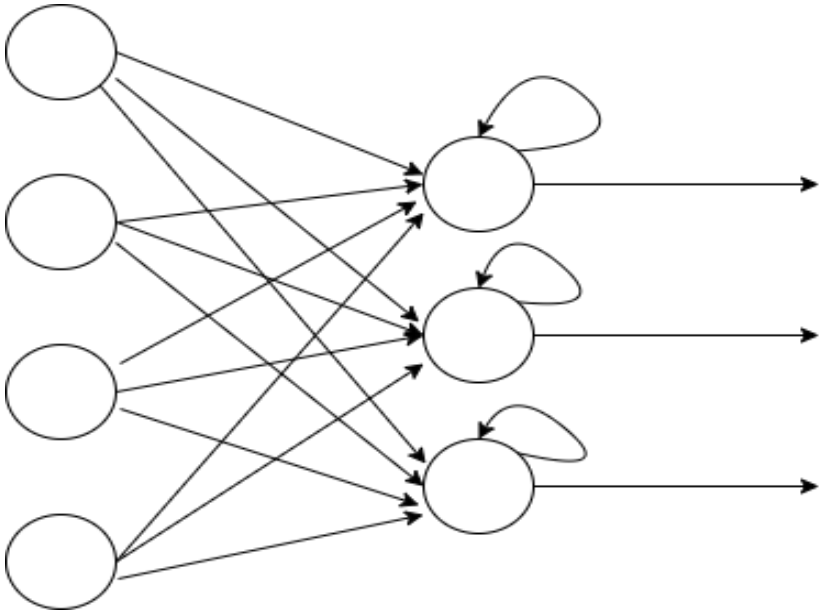
5.3 Radial Basis Neural Network

Definition 5.3. *This is similar to KMeans Classification. Here the classification is performed by measuring the input vectors similarity to the training data points which is used for deciding the class. This similarity measure returns value between 0 and 1 , where 1 means more similar and 0 means not similar.*

5.4 Recurrent Neural Network

Definition 5.4. *The Recurrent Neural Network works on saving the output of a layer and feeding it back to the input in predicting the input. In this type of Neural Network each Neuron acts like a memory cell.*

The first layer would be feed forward model with no memory , but all the remaining layers will store some information which it has computed previously . In case of a wrong prediction the RNN uses this saved information along with the learning rate and corrects itself and tries to predict things correctly.



Representation Of RNN

5.4.1 Applications of RNN

(a) Text Applications like auto suggest.

(b) Text to Speech processing.

(c) Translation

5.4.2 Advantages

(a) It can be used to train on sequential data where input data can be assumed to be dependent on historic data.

5.4.3 Disadvantages

(a) It also suffers with Vanishing Gradient Descent Problem.

(b) Training RNN requires a lot of time and computation.

5.5 Convolution Neural Network

Definition 5.5. *Convolution Neural Networks are similar to feed forward neural networks where the neurons have learnable weights and biases. The building blocks of CNN's are filters or Kernels. These Kernels are used to extract the relevant features from the input using the convolution operation.*

Although CNN were introduced to solve problems related to image data , these also perform very well on sequential inputs.

5.6 Modular Neural Network

Definition 5.6. *Modular Neural Networks have a collection of different networks working independently towards solving the problem . These networks do not interact with each other and each network is fed with unique inputs.*

The main advantage with MNN is that the computation speed decreases along with the complexity.

5.7 Sequence To Sequence Models

Definition 5.7. *A Sequence to Sequence Model consists of two RNN , one for encoding the data(encoder) and other for decoding the output (decoder). The encoder and decoder can use the same parameters or different and they work simultaneously.*

This model is used when the size of input data is equal to size of the output.

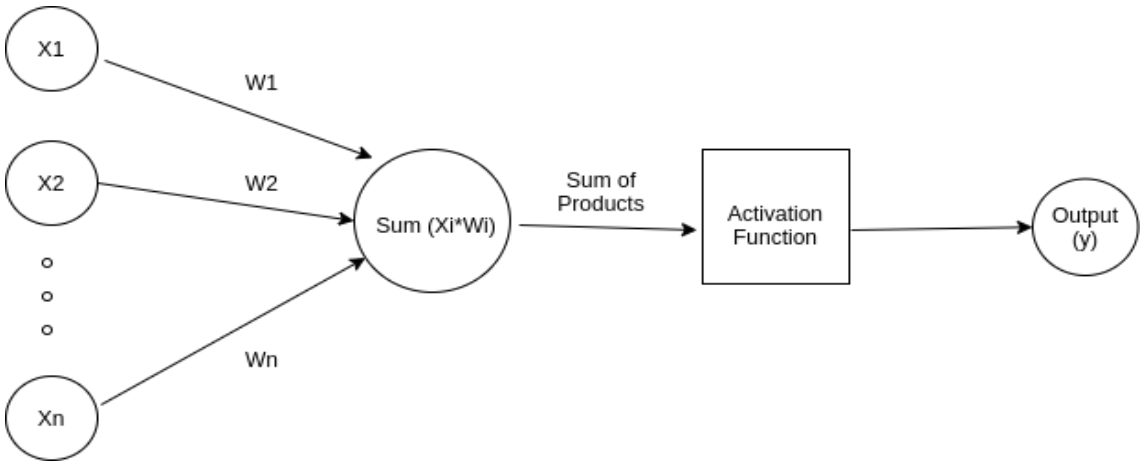
5.7.1 Applications

(a) Chatbots

(b) Machine Translations

6 Single Layer Perceptron

As mentioned above , Single Layer Perceptron is the first ever model created in Neural Networks. The content of the neuron is the weights vector. These weights are multiplied with the input data and this sum is given to the activation function for output.



6.1 Logistic Regression Using Single Layer Perceptron

Steps to train the SLP are :-

1. First initialize the weights randomly.
2. Now predict the output using the weights.
3. Calculate the loss and adjust the weights accordingly.
4. Repeat the above process until we reach the maximium iterations or if the SLP has converged.

6.2 Multi Layer Perceptron

In Multi Layer Perceptron(MLP), we connect or concatenate multiple perceptrons and model a complex function class. We can connect many single layer neural networks to for a multi layer neural network. A typical MLP has

1. Nodes and Edges : Network has nodes (where simple computations take place and edges where information flow happen.
2. Layered Architecture : Network is understood and represented in layers.
3. Dense Connections : Successive layers are often fully connected.

One way to appreciate a MLP/Neural network as a learnable function from X to Y. Eventually this network models the transformation of the input xi to yi. From this point of view, MLPs can be used for either regression or classification. In other words, y could be a single integer, real number or multiple integers/real numbers.

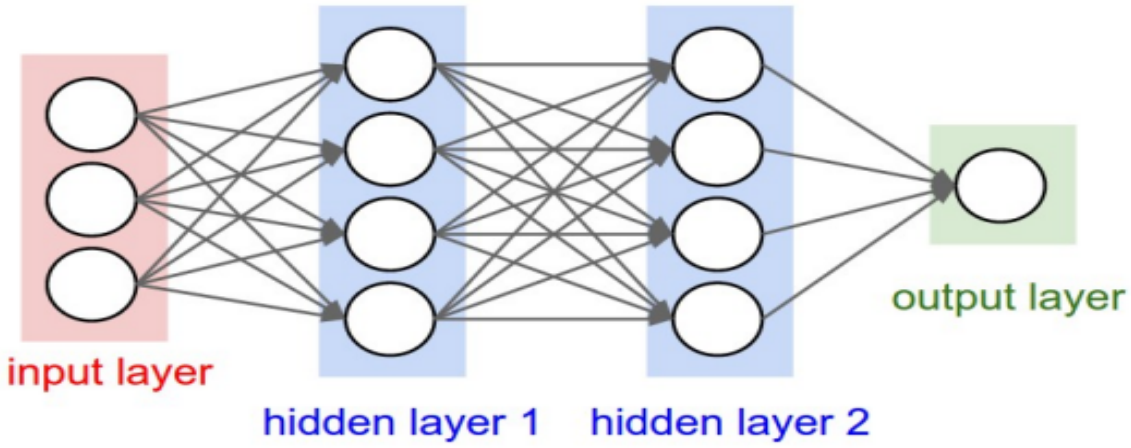
A typical MLP what is given to us in the input, output pairs (x_i, y_i)i = *1*,.....*N*. Typically $\mathbf{x}$ and $\mathbf{y}$ are vectors. The number of neurons in the first/input layer is the dimensionality of x. Numbers of neurons in the output layer is the dimensionaliity of y. We have two things in our control (i) Number of hidden layers (ii) number of neurons in each hidden layer. Typically, we go for 2 or 3 hidden layer. With number of layers increasing, the network becomes deep and the learning problem becomes difficult due to issues like vanishing gradient.

The number of neurons in each layer is typically larger than the number of neurons that are required at the input or output layers.Note that typical neural networks are over parameterized,i.e., there are more neurons and weights than what is required for solving the problem.

6.3 MLP for Classification

MLPs are very good for the classification. In the case of binary classification, one can design a network with only one output neuron. This neuron could output the probability being class-1. And a complement of this could be class-2. This can be achieved by keeping a sigmoid or tanh at the output neuron.

What about multiclass classification? Assume there are four classes, how many neurons are required in the output space? One solution could be to use two neurons and encode the classes as 00,01,10 and 11. However, this assumes a structure at the output spaces(eg. a specific class is larger than the other). This is not preferred. The preferred solution is to have four output neuron and each outputting the probability(or confidence) for certain class. What if two classes are fired at a time?. To avoid this, one can use softmax at the output layer.



In other words, in the case of classification, it is the practice to represent the class-ID as a one hot vector. i.e., if there are C classes, there will be C neurons in the output. The desired class is p, then we represent the output as a C dimensional vector with zero every where except at pth location. Output is often a softmax

6.4 MLP for Regression

MLPs are popular for regression as well as classification. In the case of regression, the output neurons predict a real quantify. In some cases, the output range need not be [0,1] or [-1,1]. For example, we want to design a network that will predict the age of a person from a photograph.In such cases, output could be even a simple linear activation also.

References

[1] https://ctan.math.illinois.edu/info/Math_into_LaTeX-4/Short_Course.pdf

[2] https://machinelearningknowledge.ai/brief-history-of-deep-learning/

[3] https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207