Foundational Principles of Deep Learning – my notes:

# Introduction

As I had mentioned some weeks ago, I will be publishing a series of blogs which will go into the finer details of the Transformer architecture. Each part of the series will involve theoretical insight as well as an insight from coding point of view into every unit/ concept involving Transformer architecture as illustrated below. Each respective part will go into the details of the following:

* Sentence Tokenization and Input Embedding
* Positional Encoding
* Layer Normalization
* Self-Attention
* Multi-Head Attention
* Etc

However, before diving into the above ingredients, I felt that it is essential to document the following:

* Firstly – the foundational principles of Deep Learning
* Secondly – the evolution of Language Models

By revisiting the above 2 topics in (some) detail, the content related to the Transformer is better understood. Hence, I felt that it is definitely useful to spend time and effort into the above topics before diving into the working of Transformers.

This article will go into the details of the foundational principles of Deep Learning and will cover the following aspects:

* Section 2 revisits the definitions of Artificial Intelligence, Machine Learning and Deep Learning
* Section 3 describes the effectiveness/importance of Deep Learning
* Section 4 revisits the concept a perceptron leading to the illustration of neural network.
* Section 5 talks of neural network training, cost function/objective function/empirical loss
* Section 6 throws light on optimization algorithms
* Regularization is not discussed in this article as I have already talked on it in detail min my blog post here: <https://www.linkedin.com/pulse/understanding-regularization-ajay-taneja/?trackingId=Adq5c%2B0WTjaFq7jsjA8kAQ%3D%3D>

Needless to mention, all the content discussed below is a collection of my notes from various courses that I’ve taken so far, YouTube Videos that I’ve watched and blog posts by other learners available as open source. This is documented for my own future reference, however, I’m obviously happy if it benefits anyone reading this content.

# Artificial Intelligence, Machine Learning and Deep Learning

**Artificial Intelligence:**

Talking of Intelligence, it means the processing of information such that we can use it to infer some future decision and action that we take. The field of Artificial Intelligence involves building of computer algorithms that do exactly the same thing: process information to infer some future decision.

**Machine Learning:**

Machine Learning is a subset of Artificial Intelligence that focusses specially on making a machine learn to the above based on some human/real experiences. Statistical techniques are used to enable machines to improve with experience.

**Deep Learning:**

Deep Learning is a subset of Machine Learning that uses multi-layered neural networks to extract patterns that occur within the data so that the network learns to perform the tasks which otherwise will require human intelligence.

This article will go into the foundational aspects of Deep Learning which focusses on making the computer learn different tasks directly from the raw data.

Diagram

Description automatically generated

Figure: Illustration of Artificial Intelligence, Machine Learning and Deep Learning

# What does Deep Learning have to offer?

Traditional Machine Learning algorithms typically define features (patterns) in the data and typically a human with expert domain knowledge will uncover these features – the key idea of Deep Learning is that instead of a human defining these features, the machine extracts patterns in the data so that it can use those to make some decisions.

For example, for a face detection algorithm, a deep neural network will learn that in order to detect the face, it first detects the line, edges which can be combined to get mid-level features like corners and curves which in turn can be combined to in deeper layers of the neural network to form high level features like eyes, ears, node, etc. and then all these together will be able to detect the face.

All the learning is hierarchical starting from lower layers of the network as illustrated in the figure below.

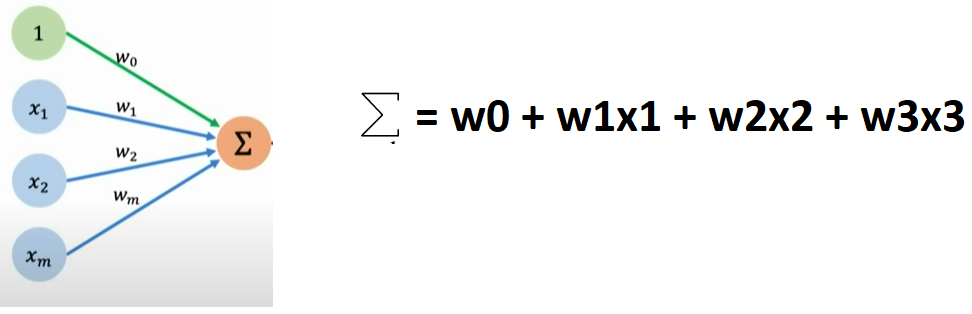
A picture containing text

Description automatically generated

**Figure: Hierarchical learning of the features in deep neural networks**

# The Building Block of Deep Learning: The Perceptron

Let us now start with the fundamental building block of every single neural network that one may develop – which is a single neuron. In the deep learning language, a single neuron is called a perceptron. The perceptron is a single neuron, and its internal state is represented by a set of inputs x1 to xn which are multiplied by the corresponding weights and added together – we also add a bias term indicated as w0 as shown in the figure below.



Then, we take the single number after the addition and pass it through a non-linear activation function and that produces the final output of the perceptron which may be termed as y\_bar as shown below.

Diagram

Description automatically generated

The process is mathematically represented by the equation below:

A picture containing diagram

Description automatically generated

A picture containing text, device, gauge

Description automatically generated

**Purpose of activation function:**

The point of non-linear activation function is to introduce non-linearities to the data/ Almost all real-world data linear in nature, thus, if we want to deal with those datasets, we need models which are also non-linear so that the models can capture the kind of patterns in the data.

To understand this better, let us say we have a dataset as shown in the figure below:

Chart, scatter chart

Description automatically generated

**Figure: Non-Linear dataset**

Suppose given this dataset, we have to construct a decision boundary i.e., a boundary separating the red and the green dots in the figure above. Now, if were to use only a straight line to separate the green and the red points, the best we could do is to separate as shown in the figure below:

Chart, scatter chart

Description automatically generated

**Figure: Straight line (linear approach) to**

**construct the decision boundary.**

Thus, the problem cannot be solved effectively using a linear approach and we will have to resort to non-linearity which helps to deal with such types of problems. The non-linear activation functions allow us to deal with non-linear data which makes the neural networks very powerful.

Qr code

Description automatically generated

**Figure: Decision boundary after using on-linear activation functions**

Further, it may be underscored that since we’re just multiplying the inputs with the corresponding weights and adding them together, the problem remains a linear problem until we introduce non-linearities using non-linear activation functions.

**Types of non-linear activation functions:**

Some types of activation functions include,

* Sigmoid Activation Function
* Tan hyperbolic/ Hyperbolic Tangent activation function
* ReLu activation function

These are illustrated in the figure below:

Chart, line chart

Description automatically generated

**Figure: Activation Functions**

# Perceptron To Neural Network

Continuing with the above discussion: Now, let’s take the (single) perceptron and build something more striking!

Now, suppose we want 2 outputs from the function. We simply add another perceptron – this added / second perceptron will have its own sets of weights. Each perceptron controls the output of its associated piece.

A picture containing circle, screenshot, line, diagram

Description automatically generated

**Figure: Single layered neural network with 2 perceptron’s**

Further, such perceptron’s can be stacked to form a single layered neural network as shown below:

A diagram of a network

Description automatically generated with low confidence

**Figure: Single layered neural network**

A deep neural network can be built by stacking more such sequential layers as shown in the figures below:

A picture containing pattern

Description automatically generated

**Figure: Deep Neural network with 3 hidden layers**

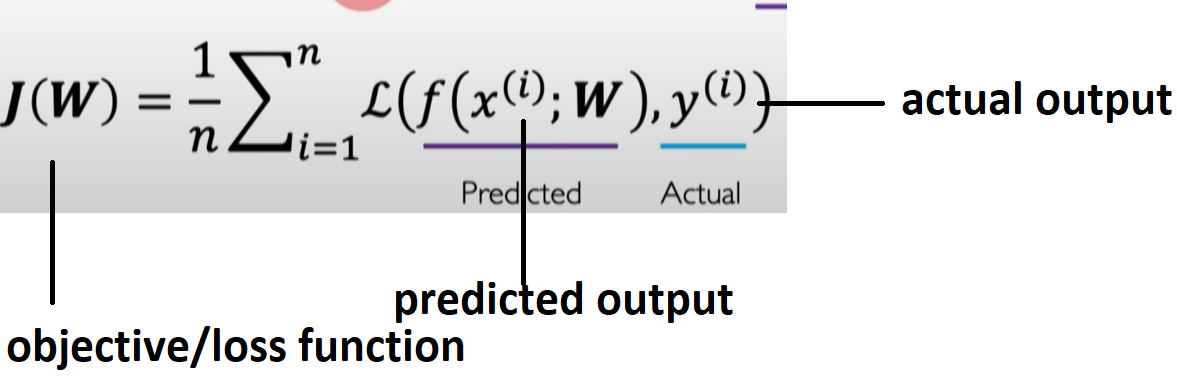
With this illustration, we can imagine/interpret that the given inputs (at the beginning) are transformed into a new dimensional space with the values closer to what we want (i.e., closer to the output we want) and this transformation has to be learnt and this is described in the next section which pertains to the loss function (or objective function).

# Training a neural network

**Loss Function:**

Having constructed the neural network (single/multi layered) and if we just start utilizing the network – with random values of weights – to predict the output, the network will not predict correctly because it’s not yet been trained. The network does not have the information of the world concerning the problem!

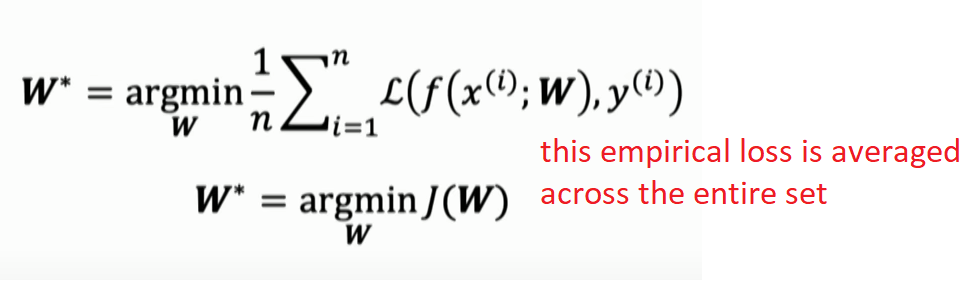
To train the network, we will have to construct the **loss function** which will tell us how far the predicted output is from the actual output. The loss of the network measures the cost incurred from incorrect prediction. The loss function is also termed as the objective function or the cost function or empirical loss and is a measure of the total loss over the entire dataset. Mathematically, the loss function is expressed as follows:



As it may be noticed from the equation above, the loss function is a function of the inputs and the weights – i.e., the predicted output and the actual output.

**Minimizing the Loss:**

Training the neural network will not only involve determining how far the predicted output is from the actual output but also **minimizing the loss**. Thus, mathematically we want to find the network weights that will result in the smallest loss as possible over the **entire dataset**. The mathematical equation is represented as follows:



A picture containing font, calligraphy, typography, white

Description automatically generated

**Cross entropy loss**

For a binary classification problem, the loss function employed is cross entropy loss denoted as below:

A picture containing text, font, screenshot

Description automatically generated

**Mean squared error loss:**

Mean squared error loss can be used for regression models that can output continuous real numbers denoted as below:

A picture containing text, font, screenshot

Description automatically generated

**Loss optimization: How to minimize the loss?**

The loss function is going to be the function of the weights – for a 2-dimensional problem, this loss function can be visualized as follows:

A picture containing diagram, screenshot

Description automatically generated

**Figure: Variation of loss function for different values of the weights**

In the above landscape, we want to find the least loss which will correspond to the lowest point.

This is done mathematically through the following steps:

1. Firstly, we start at a random space and compute the loss at the specific location.
2. We then calculate how the loss is changing – i.e., we compute the gradient of the loss. The process of computing the gradient is known as “**backpropagation**”.
3. The gradient tells us how the loss is changing as a function of the weights.
4. We update the weights in a direction opposite to that of the gradient.
5. We continue the above process until we get to the lowest point.

The above algorithm is formally termed as gradient descent. Formally, the steps in the Gradient Descent algorithm may be highlighted as follows:

* Initialize the weights of the network randomly.
* Loop until convergence
* Compute gradient – the gradient tells us how the loss is changing in the direction of the??
* Update weights

A picture containing text, font, white

Description automatically generated

The weights are updated in the direction opposite to that of the gradient. The parameter η is a small step that we take in the direction opposite to that of the gradient and is commonly termed as the “learning rate”.

* Return the weights.

# Backpropagation:

The process of computing the gradient is termed as Backpropagation.

Mathematically, for a single layered neural network with two neurons as shown below, the gradient is computed using the chain rule of differentiation – backwards from the loss function throughout the output – as follows:

A red line with black text

Description automatically generated with low confidence

A picture containing text, font, number, screenshot

Description automatically generated

A picture containing font, text, number, screenshot

Description automatically generated

A picture containing text, font, number, screenshot

Description automatically generated

The algorithm of backpropagation is decades old, and the paper is attached below!

It must be underscored that the landscape of the cost function involved in a deep neural network is highly complicated than the one shown in the Figure:

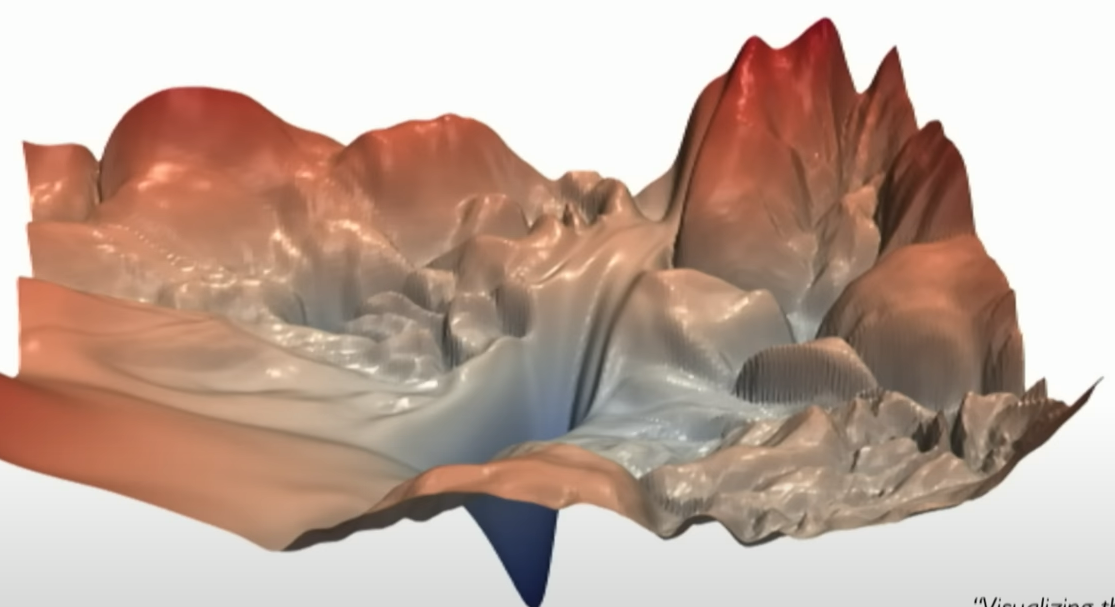


Figure: Loss function in a deep neural network

**Setting the learning rate: η**

Setting the learning rate can have very large consequences while building the neural network, Having the learning rate too small will make the travel to the lowest point in the landscape too slow (convergence is slow) whereas if the learning rate is high the calculation might bypass the point of global minima as intuitively illustrated below.

A picture containing text, line, diagram, screenshot

Description automatically generated

Figure: Learning rates overshooting

In practise an adaptive process is followed wherein the “learning rate” “adapts” to the landscape. When we say “adapt”, it means that the learning rate can be made smaller or larger depending on:

* How large the gradient is.
* How fast the learning is happening.
* Size of weights
* Etc.

More details can be found here: <https://www.ruder.io/optimizing-gradient-descent/>

# Regularization:

𝗗𝗿𝗼𝗽𝗼𝘂𝘁𝘀  
  
In the case of Neural Networks, regularization is typically done using "Dropouts". In Dropouts, during training, we essentially select randomly some subset of neurons in the neural network and prune these neurons with some probability. We randomly turn these neurons on and off at different iterations during training.  
  
This essentially forces the neural network to learn an "ensemble" of different models. 𝐼𝑡 𝑐𝑎𝑛 𝑏𝑒 𝑖𝑛𝑡𝑒𝑟𝑝𝑟𝑒𝑡𝑒𝑑 𝑠𝑜 𝑏𝑒𝑐𝑎𝑢𝑠𝑒 𝑎𝑡 𝑒𝑣𝑒𝑟𝑦 𝑖𝑡𝑒𝑟𝑎𝑡𝑖𝑜𝑛 𝑡ℎ𝑒 𝑛𝑒𝑡𝑤𝑜𝑟𝑘 𝑤𝑖𝑙𝑙 𝑏𝑒 𝑒𝑥𝑝𝑜𝑠𝑒𝑑 𝑡𝑜 𝑑𝑖𝑓𝑓𝑒𝑟𝑒𝑛𝑡 𝑚𝑜𝑑𝑒𝑙𝑠 𝑖𝑛𝑡𝑒𝑟𝑛𝑎𝑙𝑙𝑦 𝑡ℎ𝑎𝑛 𝑡ℎ𝑒 𝑜𝑛𝑒 𝑖𝑡 ℎ𝑎𝑑 𝑜𝑛 𝑡ℎ𝑒 𝑝𝑟𝑒𝑣𝑖𝑜𝑢𝑠 𝑖𝑡𝑒𝑟𝑎𝑡𝑖𝑜𝑛𝑠 𝑏𝑒𝑐𝑎𝑢𝑠𝑒 𝑎 𝑑𝑖𝑓𝑓𝑒𝑟𝑒𝑛𝑡 𝑠𝑒𝑡 𝑜𝑓 𝑛𝑒𝑢𝑟𝑜𝑛𝑠 𝑎𝑟𝑒 𝑡𝑢𝑟𝑛𝑒𝑑 𝑜𝑛 𝑎𝑛𝑑 𝑜𝑓𝑓. This results in being a very powerful technique and helps in generalizing better.

A picture containing circle, line, screenshot, diagram

Description automatically generated

**Figure: Dropouts – Pruning of neurons during iterations of training**

𝗘𝗮𝗿𝗹𝘆 𝗦𝘁𝗼𝗽𝗽𝗶𝗻𝗴

The next regularization technique often practised for Neural Networks in "Early Stopping". Here the Data Scientist will normally plot the performance of the network on the training and the test data. As the network is trained, one would notice both the training and the test set loss decrease but a stage is reached where the training error continues to decrease but the test set error begins to increase. IT is at this point essentially that the model is beginning to overfit. And it is this point one would want to stop the training process as otherwise the model will learn the training data very precisely but not perform well on unseen data (overfitting).

A picture containing line, diagram, plot, parallel

Description automatically generated

**Figure: Early stopping**

I have discussed in (some) detail on Regularization in an earlier blog here:

<https://www.linkedin.com/pulse/understanding-regularization-ajay-taneja/?utm_source=share&utm_medium=member_android&utm_campaign=share_via>