Report 1

Understanding Deep Learning:

Breaking Down the Black Box and Shedding an Intuitive Light

Author: Faizan Khalid Mohsin

Course: CHL5250HY - Special Topics in Biostatistics

Professor: Wendy Lou.

March 24, 2021

Table of Contents

[Abstract 3](#_Toc68953424)

[Introduction 4](#_Toc68953425)

[Methods 8](#_Toc68953426)

[Theoretical Framework of Neural Networks 8](#_Toc68953427)

[Simplest Neural Network: Simple Feedforward Shallow Neural Network 8](#_Toc68953428)

[Statistics, Probability, and Information Theory in Neural Networks. 10](#_Toc68953429)

[Neural Networks Structure, Terminology, and Mathematical Representation. 10](#_Toc68953430)

[Regression losses 15](#_Toc68953431)

[### Metrics to evaluate the performance of an algorithm 16](#_Toc68953432)

[### Optimizer 17](#_Toc68953433)

[Activation Functions 17](#_Toc68953434)

[Gradient Descent & Stochastic Gradient Descent 20](#_Toc68953435)

[Back Propagation 22](#_Toc68953436)

[Hyperparameters of neural networks 22](#_Toc68953437)

[Regularization: Methods to Improve Neural Network Performance 24](#_Toc68953438)

[The Idea of Regularization: 24](#_Toc68953439)

[Parameter Norm Penalties 24](#_Toc68953440)

[Ensemble Methods 25](#_Toc68953441)

[Data Augmentation 26](#_Toc68953442)

[Semi-Supervised Learning: 26](#_Toc68953443)

[Dropout 27](#_Toc68953444)

[# Application of Neural Networks with Applications 28](#_Toc68953445)

[## Setting up the software. 28](#_Toc68953446)

[### Installing Python 28](#_Toc68953447)

[### Installing Tensorflow 28](#_Toc68953448)

[### Installing Keras 28](#_Toc68953449)

[## Neural Networks to Predict Popularity of the Ted Talk. 29](#_Toc68953450)

[### Data 29](#_Toc68953451)

[### Univariable Analysis 29](#_Toc68953452)

[### Multivariable Analysis 29](#_Toc68953453)

[### List of issues in Neural Network Application (also applicable to machine learning). 29](#_Toc68953454)

[Creating Images to Represent Neural Network Architectures. 30](#_Toc68953455)

[# Results and Insights: Deep Neural Network 31](#_Toc68953456)

[Discussion and Conclusion 34](#_Toc68953457)

[References 34](#_Toc68953458)

[Appendix 35](#_Toc68953459)

[Definitions and Further Explanations. 35](#_Toc68953460)

[Tensor: 35](#_Toc68953461)

[Derivations: 35](#_Toc68953462)

[Deriving deep learning loss functions using Maximum Likelihood method. 35](#_Toc68953463)

[SDG Expectation 35](#_Toc68953464)

[Derivation showing that the L2 penalty causes the weights to shrink. 36](#_Toc68953465)

[Potential Research Idea 37](#_Toc68953466)

[List of pandas dataframe methods. 37](#_Toc68953467)

[## Questions 39](#_Toc68953468)

# Abstract

**Goal:** We aim to answer why deep learning is one of the most successful and popular AI algorithms. We present in depth exactly what deep learning is by introducing the mathematical foundation and framework on which it is built, as well as provide an intuitive understanding of how deep learning and its different components work. Finally, we demonstrate and explain the several advantages of deep learning. **Overview:** Deep learning is a type of neural network; to be precise, it is a deep neural network. Most components of neural networks can be represented as vectors and matrices, and many calculations can be represented as matrix multiplications or by taking the expectation of different mathematical objects and operators. **Methods:** We discuss the different important components of neural networks: artificial neurons, layers, perceptrons, activation functions, cost functions, backpropagation, (stochastic) gradient descent, and minibatches. **Advantages:** (1) The performance of deep neural networks consistently continues to improve as more and more data is provided for it to train on. This is in contrast with many statistical and other machine learning models whose performance plateaus after a particular data (sample) size is reached and the performance is, in a sense, capped by some upper theoretical bound. (2) Further, it is one of the most versatile algorithms capable of tackling many different problems, as well as a very wide variety of data types such as sound data, image data, natural language, and structured data. (3) Lastly, one of the most amazing properties of deep neural networks is the ability to self-select features - and to do so directly from the raw data it is provided. **Conclusion:** These collective advantages found in deep learning is what overwhelmingly makes it the most successful and powerful artificial intelligence algorithm.

# Introduction

What is deep learning?

The short answer is that it is a type of neural network, more specifically a deep neural network. Neural networks (NN) are a type of Machine Learning algorithm and machine learning (ML) itself is a subfield of artificial intelligence. Machine learning is divided into three main branches: supervised learning, unsupervised learning, and reinforcement learning. Deep learning (DL) is a special approach in machine learning which covers all three branches and seeks also to expand them to address other problems in artificial intelligence which are not usually included in machine learning such as knowledge representation, reasoning, and planning among others (Skansi, 2018, pg 51). In this paper, we will only cover the supervised learning branch of deep learning.

Deep learning employs four main fields: Mathematics, Probability and Statistics, Information Theory, and Computer Science (Programming) (Skansi, 2018, pg 41). The idea of a neural network originates from biology and neuroscience mimicking how computation and thought processes of the brain work. The biological neural network consists of a population of interconnected neurons.

The neuron is a cell that can be electrically excitable and is capable of communicating with other cells and neurons. The neuron consists of the cell-body (soma), - which contains the nucellus; dendrites, - which receive signals (neurotransmissions and electrochemical stimulation) from other cells; and lastly, a single axon, - which transmits signals to other cells and can extend as far as 1 meter in humans. See Fig. 1 (Top) illustrating the different components of a biological neural in detail. The specialized connections through which the neurons are connected, and signals are transmitted are called synapses.

This idea of receiving signals, processing them, and further transmitting them is adopted into the concept of artificial neurons through inputs and outputs as illustrated in Fig. 1 (Bottom) which also shows the parallels between a biological and artificial neuron.



![ Fig.1: Comparing a biological neuron (Top) with artificial neuron (Bottom) and their components.](images\_graphresults/biological\_and\_ai\_neuron\_image.png)

This establishes the concept of an artificial neuron.

These artificial neurons are connected to form an artificial neural network. Unlike the biological neural network, which is very complex with many intricate connections, in artificial neural networks, the neurons are connected in a set pattern.

In an artificial neural network, the neurons are organized in layers. Each layer has several neurons. The neurons within each layer are not connect to one another. They are only connected to neurons in the layer before or after it.

The first layer in the artificial neural network serves a very special function: it takes in inputs and is hence called the input layer. There are as many neurons as there are different types of inputs. The last layer of the NN is called the output layer as it produces the final desired outcomes of interest. The number of neurons in the output layer depends on the types and number of outcomes desired. This can be very simple such as a binary outcome indicating if the input image contains a cat or not, thus will only have one neuron in the output layer which will produce either a 0 or a 1.

The layers in between the input and output layers are called hidden layers. They are the layers that perform the main calculations including feature selection of the data and patterns recognition to best predict the outcomes, or best match the pattern of the data provided to the NN. There can be any number of hidden layers in a NN.

To summarize, a population of artificial neurons connected to one another through three main types of layer:s the input layer, the hidden layers, and the output layer is what forms the basic structure of an artificial neural network and is inspired from the biological NN structure of the brain. See Fig. 2 below.



![ Fig. 2: Artificial Neural Network.](images\_graphresults/Artificial\_neural\_network\_image.png)

There are more components of the NN, however, these we will discuss in the Methods section when formalizing the artificial NN mathematically. The simplest artificial neural network will only contain one hidden layer. By increase the number of hidden layers we obtain a deep neural network. There are obviously several different types of neural networks constructed to solve and tackle a wide variety of specialized problems and data types.

The main types of neural networks are:

1. Feedforward Neural Networks

2. Convolutional Neural Networks

3. Recurrent Neural Networks

4. Encoder-Decoder Networks

5. Recursive Neural Networks

Lastly, people familiar with statistics have already encountered a NN known as logistic regression. It is a one-neuron neural network called a perceptron.

One of the things that make deep learning extremely powerful is the ability to do feature selection and pattern recognition directly from the raw data. For other machine learning algorithms, the features are either predetermined or preselected (e.g., choosing which variables to keep and what transformations to do on the variables), however self-selecting features and doing so directly from the raw data is unique to deep learning. Another, extremely amazing property of deep neural networks is that the performance of deep neural networks consistently keeps improving as more and more data is provided for it to train on. This is in contrast with many statistical and even other machine learning models whose performance plateaus after a particular data (sample) size is reached and the performance is, in a sense, capped by some upper theoretical bound. One reason is that many models, specifically statistical models, divide the data into two components: signal and noise – a distribution and an error term. The error is random and hence in the statistical model it is inherent to never be able to achieve 100% accuracy because the highest accuracy it can reach will always be one minus the magnitude of the noise divide by the magnitude of the signal (the proportion of the data affected by the noise versus the signal). Another way to think about this is that the error term will also take away from the accuracy of the model. Now, this approach and framing is in fact very advantageous for understanding and isolating relationships and variables, which is the point of statistical models. Their point is not prediction, and certainly not achieving 100% accuracy under all circumstances. Statistical models are closer to models in physics and are a generalization of physical models that primary model causal relationships. Statical models generalize from not only capturing causal relationships but also associations, correlations, links, and connections – relationship that are much weaker than causality, more general in principle, and more observable in the world and in data. The purpose of statistical models is to understand how certain variables relate to one another, be it causal or less strongly associated. Its purpose is not prediction.

The purpose of prediction is taken up by neural networks.

Now we come to the point why are deep neural networks able to consistently keep improving their performance till they appear to reach near 99.9% accuracy as sample size (the training data) grows whereas statistical and machine learning models’ accuracy plateaus well before reaching 95% accuracy even as the sample size keeps on increasing. It is as if the other models have a theoretical cap for the performance of their accuracy. We believe deep learning is able to achieve this because of the very complex nonlinear patterns it can learn. The last 5% of the accuracy that other models are not able to attain is indeed captured by the neural network because this 5% is not the rule but indeed the exception to the main trend that the other models are able to capture and to be able to capture this 5% the main trend that accounts for 95% of the signal has to be changed and coerced which started decreasing the overall accuracy much more quickly, than the improvement seen from capturing the 5% exception to the rule. What we think neural networks are able to do is in fact hold within them multiple models within different sets neural network connections. Hence, store different patterns of the in different parts of the neural network. Turning off and on these sections of the network to predict the result given the pattern is the main trend or if it is the exception to the main trend. This ability of holding different models, trends, patterns, in different sections of the network is what allows neural networks to reach such high levels of accuracy as more amount of data, and trends within the data, are provided for the it to train on and learn.

Now often it is said that to become a deep learning expert one has to develop this intuition. Basically, do so many applications of deep learning and experiment what produces the best results under what circumstances as to emperically learn

Now, why do we need such an “experimental” approach for understanding machine learning (similar to running experiments in physics)? Why not methodologically and analytically solve and understand the behavior of neural networks. This is because the sheer number of parameters, architectures, and permutations possible, make it infeasible to so do analytically and is easier to do experimentally. This does not mean to say no analytical attempt should be made. Certainly, for smaller networks it is very important to do things analytically to understand the fundaments of neural networks. We will delve into some detail doing that.

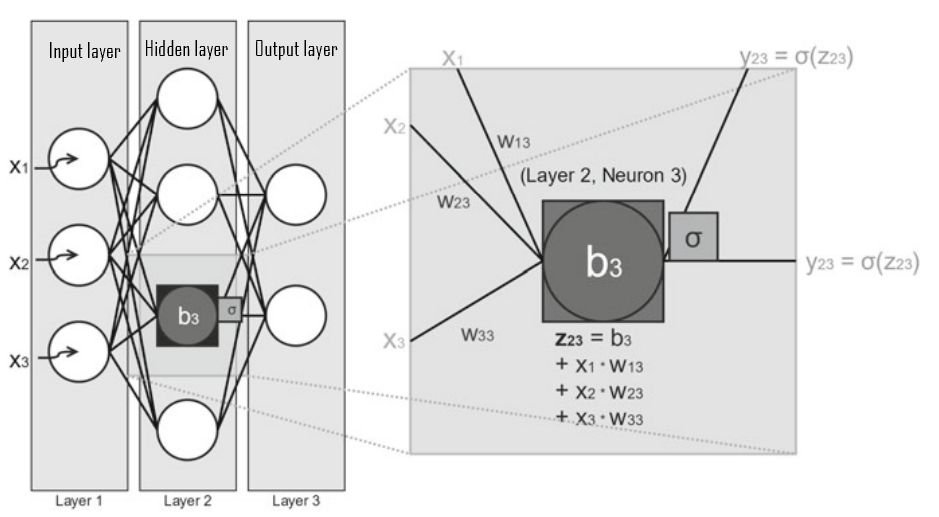
# Methods

In this section we will establish the theoretical and mathematical framework of neural networks, explain the important terminology used, the main algorithms involved, and an intuitive understanding of deep neural networks.

## Theoretical Framework of Neural Networks

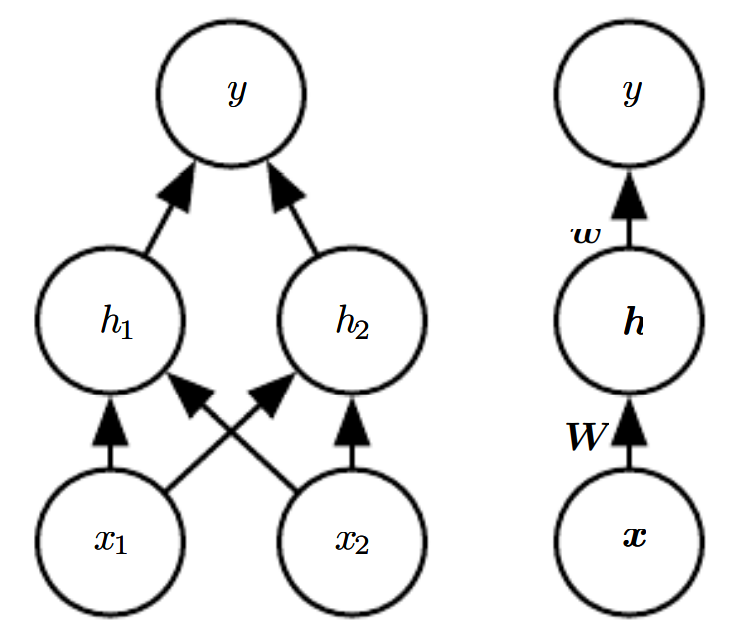
### Simplest Neural Network: Simple Feedforward Shallow Neural Network

As mentioned in the introduction, one of the simplest neural network is one with only three layers: one input layer, one hidden layer, and one output layer; where information is only processed and conveyed in forward direction. This is called a Feedforward Shallow Neural Network (Skansi, 2018; pg 79).



![ Fig. 3: A simple neural network with three layers: Input layer, hidden layer, and output layer (Skansi, 2018, pg 80).](images\_graphresults/SimpleNNGraphV2.png)

*Feed forward neural network* is simply one in which data flows only in the forward direction and has no loops or cycles. From the input layer to the hidden layer(s) to the output layer Zell (1994). This can be seen in Fig. 4 below.



![Fig. 4: A feedforward network, drawn in two different representations. It has a single hidden layer containing two units. (Left) In this presentation, we draw every unit as a node in the graph. (Right) We indicate that a matrix W describes the mapping from x to h, and a vector w describes the mapping from h to y. (Goodfellow, 2016; pg 174)]( images\_graphresults/FeedForward\_network\_twoStylesImage.png)

Now an important question we would like to answer is that which functions can feedforward neural networks approximate? This will help us form an idea what are the “potential” patterns a feedforward NN can mimic in a potential data set.

“The *universal approximation theorem* (Hornik et al., 1989; Cybenko, 1989) states that a feedforward network with a linear output layer and at least one hidden layer with any “squashing” activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units” (Goodfellow, 2016; pg 198).

This means that feedforward networks with hidden layers, provide a universal approximation framework and can approximate virtually any function to any desired accuracy. Hence, we are in good shape to use feedforward neural networks and can use it to solve a wide variety of problems.

### Statistics, Probability, and Information Theory in Neural Networks.

Statistics, probability, and information theory play a very important role in artificial neural networks (and machine learning in general). They are used to deal with uncertainty in three important ways:

1. Firstly, probability and statistics allow us to quantify uncertainty.
2. Secondly, they are used to design how our neural networks should reason and analyse.
3. Thirdly, they allow us to assess and evaluate the accuracy and behaviour of neural networks.

For these three reasons not only is statistics, probability, and information theory very import for artificial neural networks but also for Machine Learning and Artificial Intelligence in general, as they allow us to do the exact same three things for our Machine Learning algorithms and Artificial Intelligence Systems (Goodfellow, 2016; pg 53).

Uncertainty is present in two important ways. The first is in terms of uncertainty in quantities or the data such as missing, or incomplete data, etc. The second is uncertainty due inherent randomness in some processes such as rolling dice, called stochastic processes – in other words random processes (Goodfellow, 2016; pg 54). Statistics and Probability also help us establish robust ways of evaluating the algorithms of our neural networks such as the MSE among others and help improve the optimization algorithms.

Lastly, statistically speaking, whenever we use neural networks, we are implicitly stating that we have a general prior belief that that the function we want to our neural network to learn should involve composition of several simpler functions. This point will be important when talking about the depth of a NN.

### Neural Networks Structure, Terminology, and Mathematical Representation.

#### Overview

At an abstract level, a Neural Network can be thought of as a function.

f x y θ : , → which takes an input x n ∈ and produces an output y m ∈ , and whose behavior is parameterized by θ ∈p. So for instance, f θ could be simply y f = ( ) x x = ⋅ θ θ (Ketkar, 2017; pg 17)

We will first give an overview of the different components of a neural network. As mentioned before, an artificial neural network has three types of layers: the input layer, the hidden, and the output layer. The structure of a simple three-layer neural network is shown in Fig. 3. Each layer can have any number of neurons. A neuron is an element that holds information, or in other words, holds an input at each layer of the neural network. Further, typically each neuron of a layer is connected to all neurons of the previous and following layer. However, neurons in the same layer are not interconnected. The input from one layer is multiplied by a “weight” before the resulting product is sent to a neuron in the next layer.

The value of this “weight” can be any non-negative value (not just between 0 and 1 as is sometimes incorrectly perceived), and hence can be thought as either “diminishing” or “amplifying” the input that is transmitted to the next neuron. The weights between connected neurons can all be of unique values and need not be the same (Skansi, 2018, pg 79).

Vectors and matrices are used to as the main way to represent NN components, and the computation of NN is calculated as matrix multiplications. Now, each layer can be represented as computing a function where would represent the input, hidden, and output layers of the simple neural network. Hence, if is the input received by the neurons in the input layer, then the final resulting output of the neural network can be represented as . Hence, we can represent the neural network as .

The most crucial part in NN is finding the weights that improve the accuracy of the NN. This is done (most commonly) through stochastic gradient descent.

#### Input Layer

The input layer only serves the function of accepting input. Further, each neuron in the input layer can only hold one input value. In Fig. 3 the neural network’s input layer has three neurons and can hence only take three inputs. These inputs can be represented by variables x1, x2, and x3 which in turn can be mathematically represented as a row (or column) vector of three elements called the input vector. For n neurons in the input layer, the input vector can be represented as (Skansi, 2018, pg 79).

Now, taking it a bit further, the concept and the word “Tensor” is very important in deep learning. This is because all inputs of artificial neural networks can be represented as tensors. If there is only one input neuron then the tensor is simply a scalar. We describe in detail the concept of a tensor in the Appendix section for the reader.

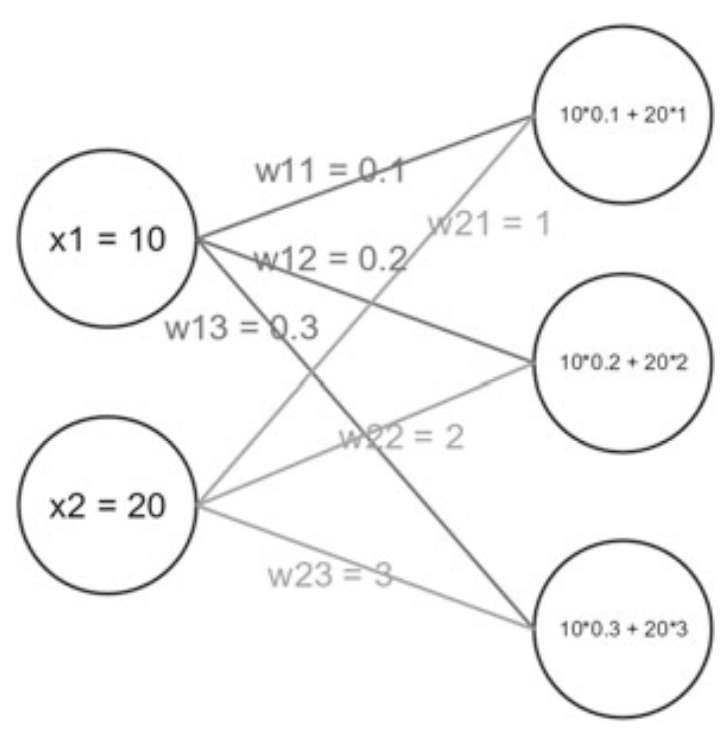
The concept of tensor becomes more and more important as the data becomes more complex, for example image data. The input of an image is converted into a 3-dimensional array, a tensor, to be able to be passed into the neural network for processing. Therefore, for an artificial neural network, as they are currently developed and structured, do take in image data as input and must have the image represented as a 3 meta-dimensional object (in other terms a 3-dimensional array).

#### Hidden Layers, Weights, and Biases

The connection between every neuron has a weight associated with it. When information or data is transmitted from one neuron of a particular layer to a neuron of the next layer it is multiplied by that weight. The weight regulates how much of the initial value will be forwarded to a given neuron, so if the input is 8 and the weight to the destination neuron is 0.5, the destination will receive the value 4. Weights can have any non-negative value; hence not only can they decrease the value being transmitted but can also amplify it if the weight is greater than 1. Fig. 5 below shows the input values, the weights of each connection, and final value received by the neurons in the second layer.

The inputs are multiplied with the corresponding neuron connection weight in the next layer. This value represented by z is then used as the input to the activation function of the neuron it is transmitted to, the result of the activation function again being used as input and multiplied by again the corresponding neuron connection before being transmitted to the neurons in the next layer. And the cycle repeats for every hidden layer. The sum of the product of the weights and inputs can be represented more generally as a column vector . For example, for Fig 3, it can be represented as: . The first index “i” of zij-element represents which layer the neuron output is from, for Fig 3 it is the second layer in the neural network. The second index represents which neuron’s output in that layer it is. Hence, since there are four neurons, the vector has four elements. More generally, output from any hidden layer in a neural network can be notated as .

The step of passing from the input layer to the first hidden layer with all the calculation is illustrated in detail in Fig. 5 below.



![ Fig. 5: Illustration of inputs (x1, x2) and weights $\textbf{w} = \begin{bmatrix} w\_{11} \left( =0.1\right) & w\_{12} \left( =0.2\right) & w\_{13} \left( =0.3\right) \\ w\_{21} \left( =1\right) & w\_{22} \left( =2\right) & w\_{23} \left( =3\right) \end{bmatrix}$ in a neural network explicitly written out.](images\_graphresults/NNWeightsInputsFig.png)

These calculations are represented as follows for the summands:

Now, in terms of mathematical representation, the weights of the neural networks can be represented as a matrix whose dimensions will be determined by the number of neurons in the input layer and the neurons in the hidden layer. So, if there are n input neurons and m neurons in the following layer then the dimensions of the weight matrix will be n x m. Hence, for the neural net in Fig. 5 will have dimensions 2 x 3, two rows and three columns.

The computation of the neural network can be represented as matrix multiplication: (Skansi, 2018, pg 83). Further, each neuron has a modifiable value in it, called the bias, which is represented here by b3, and this bias is added to the previous sum. The result of this is called the logit (denoted by z) which in our case is z23 (Skansi, 2018, pg 79).

#### Output Layer

Now, if is the input received by the neurons in the input layer, then the final resulting output of the neural network can be represented as , where if the neural network as ‘m’ output neurons.

#### Loss Function and Cost Function

Here statistics, probability, and information theory play an important role. The cost function is a very import part of the design of the deep neural network. It measures how close our model’s predictions or estimates are to the actual data. Fortunately, this is not an entirely new concept and already existed in statistics, hence we already have a lot of established framework and theory to work with. In statistics, for a parametric model such as linear models, the cost function is simply found through the principle of maximum likelihood, hence, usually the cross-entropy between the training data and the model’s predictions is used as the cost function. We use the same cost function for neural networks (Goodfellow, 2016; pg 178).

The cost function should not to be confused with the loss function **J** (or equivalently, the error function **E**), though the two are very closely related. For the cost function the performance is measure or evaluated over all the training data. On the other hand, when it is evaluated for a particular data point or for a particular instance it is called either the loss function denoted by **J** or the error function denoted by **E** (the two are one and the same - one arises from computer science terminology, whereas the other comes from statistics). Further, the cost function is calculated as the average of the loss functions error. Hence, the relationship between the cost function and loss function can be represented as follows:

Now, since the difference between the cost function and the loss function is just that the former is evaluated over the entire training set, whereas, the latter is evaluated over a subset of the training data , and the functional form of the cost function and the loss function is the same. Therefore, many people and many books use the term cost function and loss function interchangeably and it is left to the reader to understand from the context what is actually being talked about. We may also be prone to doing this going forward (sometime deliberately in the interest of brevity).

We mentioned probability and statistics is very important in deep learning and machine learning in general. The Maximum Likelihood method in statistics can be used to derive the loss functions. Many commonly used loss functions in deep learning like binary cross entropy, cross entropy, and squared error can be derived using the Maximum Likelihood principle (Ketkar, 2017; pg 24). These derivations are shown in the Appendix Section.

We give examples of a few loss functions (or cost functions if evaluated over all the data).

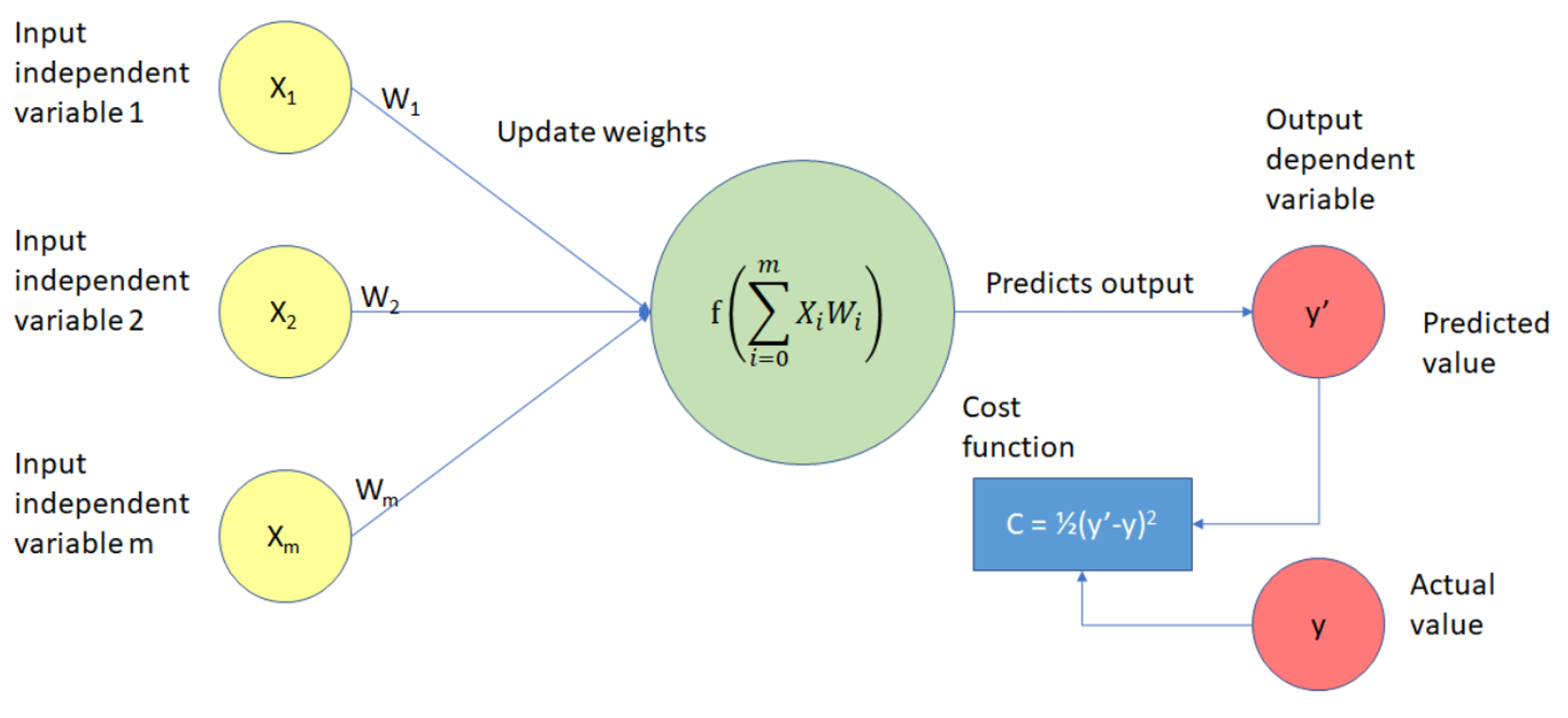
There are three very common loss functions used for regression. Where is the actual value and is the estimated value from the model:

1. Mean Absolute Error (MAE): loss = this is an regularization or constraint.
2. Mean Squared Error (MSE): loss = This is a regularization.
3. Root Mean Square Error (RMSE): loss =

Below is an example of a loss function for binary classification:

1. Binary Cross-Entropy Loss Function: loss = . Where is the actual value and is the estimated value from the model.

Below is a figure that shows how the cost function fits in the neural network scheme.



![Fig. 6: The schema illustrating how the cost function, in the blue box, is present in the neural network.](images\_graphresults/cost\_function\_schema.png)

There are several desirable and advantageous properties that the loss functions or cost functions should have. We will only mention one and that is that they be (everywhere) differentiable. A cost function that has this property will be very crucial since the gradient of the cost function has to be calculated to update the weights, though there are ways around this as well.

We list several more loss functions for regression:

### ### List of Regression Loss Functions.

* [MeanSquaredError class](https://keras.io/api/losses/regression_losses/#meansquarederror-class)
* [MeanAbsoluteError class](https://keras.io/api/losses/regression_losses/#meanabsoluteerror-class)
* [MeanAbsolutePercentageError class](https://keras.io/api/losses/regression_losses/#meanabsolutepercentageerror-class)
* [MeanSquaredLogarithmicError class](https://keras.io/api/losses/regression_losses/#meansquaredlogarithmicerror-class)
* [CosineSimilarity class](https://keras.io/api/losses/regression_losses/#cosinesimilarity-class)
* [mean\_squared\_error function](https://keras.io/api/losses/regression_losses/#mean_squared_error-function)
* [mean\_absolute\_error function](https://keras.io/api/losses/regression_losses/#mean_absolute_error-function)
* [mean\_absolute\_percentage\_error function](https://keras.io/api/losses/regression_losses/#mean_absolute_percentage_error-function)
* [mean\_squared\_logarithmic\_error function](https://keras.io/api/losses/regression_losses/#mean_squared_logarithmic_error-function)
* [cosine\_similarity function](https://keras.io/api/losses/regression_losses/#cosine_similarity-function)
* [Huber class](https://keras.io/api/losses/regression_losses/#huber-class)
* [huber function](https://keras.io/api/losses/regression_losses/#huber-function)
* [LogCosh class](https://keras.io/api/losses/regression_losses/#logcosh-class)
* [log\_cosh function](https://keras.io/api/losses/regression_losses/#log_cosh-function)

Source: Keras online Api documentation: <https://keras.io/api/losses/>

When trying different hyperparameters, I used several of these always in the search. The four I used along with my own RAM metric (regression accuracy metric) are mse, mae, mape, cosine\_similarity, msle.

Further, these loss functions were also included in the metrics for evaluating the performance of the neural network.

All the loss function choices can also be chosen as metrics to evaluate the performance of the neural network.

## ## Metrics to evaluate the performance of an algorithm

### ### New Metric to Evaluate the Performance of Regression Problems.

#### #### Problems with existing Metrics for Regression.

The typical metric to evaluate regression problems is the mean squared error (MSE) which is also the most common loss function for optimizing a regression output neural network. The problem with MSE is that it is just a number and there is little interpretability associated with it in terms of understanding how the algorithm is performing on an absolute scale. For instance, in the classification problem with the metric of accuracy, it is easily interpretable what any value of the accuracy means and easily understandable how good the algorithm is performing on a absolute scale. For example, 90% accuracy means the algorithm is able to correctly predict 90% off the test set data exactly accurately. There is no such ease of translation for the MSE. To give interpretability to your algorithm’s test MSE, it is typically evaluated against the test MSE of a baseline model and always evaluated compared to the improvement made over the baseline model. That being said, there is a very strong intrinsic relative interpretability of the raw MSE value of the algorithm which is the distance in unit measurement of the original scale of how far on average our predicted values are from the actual values. Here when I propose a new metric it will try to keep the advantage of the MSE but will also try to capture the ease of interpret ability that is found in the classification performance metric of accuracy. We can simply call this metric regression accuracy metric RAM, mean regression accuracy MRA, mean accuracy error MACE. (MSE)

This regression accuracy metric is very simple in idea. It is to first simply choose any value $\alpha$ between zero and one hence a fraction and then multiply that with one standard deviation of the error of the MSE ($\sigma\_e$). Then for each data point the predicted value y\_hat of the algorithm is subtracted from the actual value y\_actual. It is then counted how many of these differences fall within $\alpha \* \sigma\_e$ and divided by the total number of data points. What is also calculated is the ratio of the mean (sum) of the difference between the points that did fall within the fraction of the standard deviation and the mean (sum) of the difference between the points that fell outside. This value is reported with the percentage of points that fall within the fraction of the standard deviation $\alpha \* \sigma\_e$.

##### \*\*Advantages and disadvantages of this evaluation metric.\*\*

#### #### Metrics for Classification Problems

### ### Optimizer

### Activation Functions

In artificial neural networks, the activation function is a very important component of a neural network and causes the neuron to activate or not. It determines the output of a neuron.

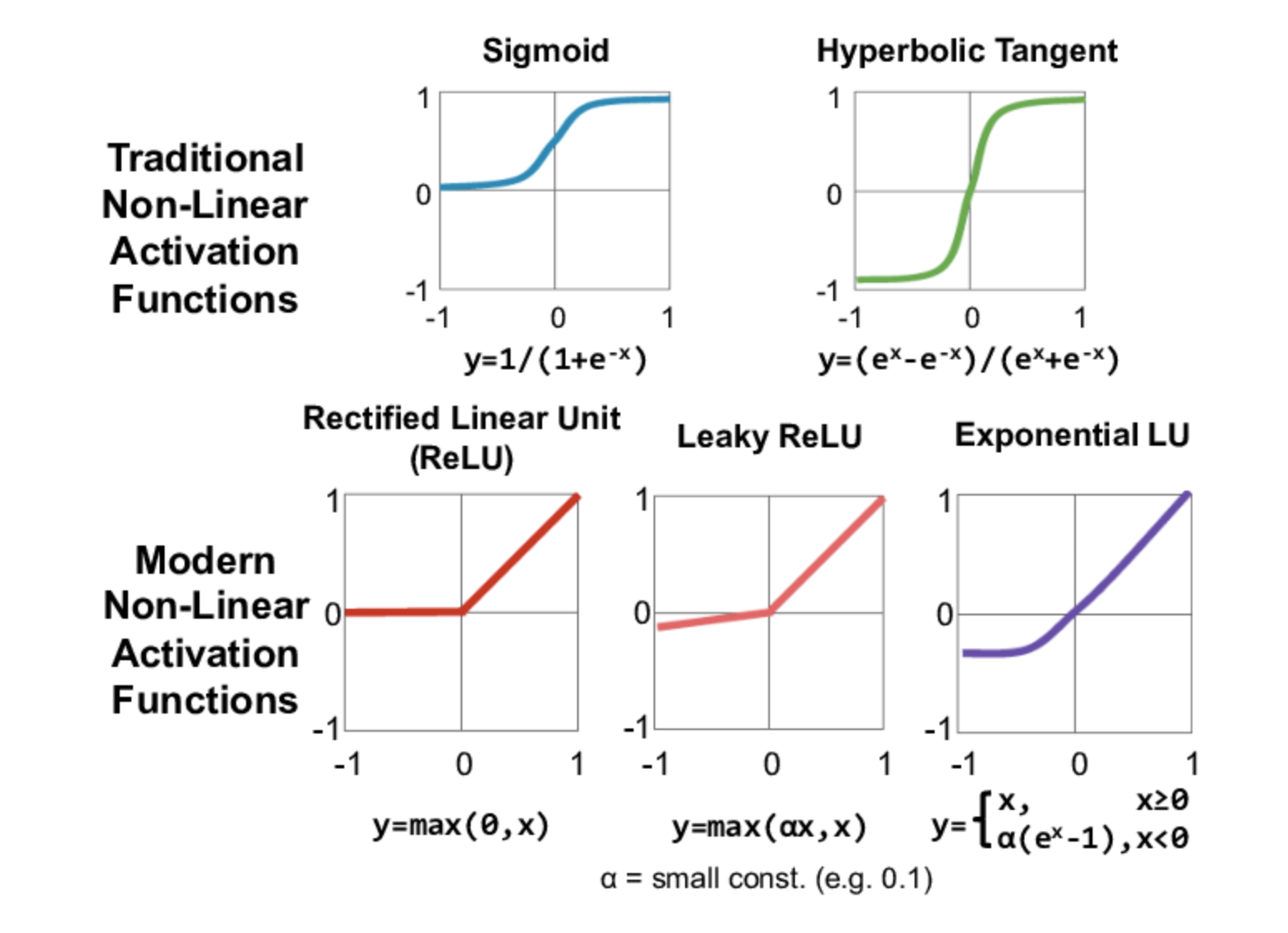
The reader may recall us mentioning that a neural network with sufficient width (number of neurons) can approximate any function. This result is only true when the activation function of the neurons is nonlinear: “In theory, when an activation function is non-linear, a two-layer Neural Network can approximate any function (given a sufficient number of units in the hidden layer).” Ketkar 2017, page 27. Hopefully this underlines the importance of the activation function to the reader as well as the advantage of having a nonlinear function.

An activation function transforms the shape/representation of the data going into it. These representations are essential for making high-dimensional data linearly separable, which is one of the many uses of a neural network.

Three important properties that are very advantageous for activation functions to have for the algorithm are:

1. Differential function: the derivative exists everywhere. This is important for performing backpropagation because it allows the gradient to be taken at any point.
2. Monotonic function: function that is either non-decreasing or non-increasing. Now, although, properties 1 and 2 are good to have, they are not necessary and there are techniques such as stochastic gradient descent that works for functions that are not everywhere differentiable for a finite set of points, as long as the limit exist from a single direction for the non-differential points ( ).
3. “A function whose range is finite (as against infinite) leads to a more stable performance with respect to gradient-based methods” (Ketkar, 2017; pg 27).

Examples of activation functions can be seen in Fig. 7 below.



![Fig. 7: Traditional and more modern activation functions.]( images\_graphresults/activation\_functions\_image.png)

It is important to note that all activations functions are nonlinear. Even piece-wise linear activation functions are completely acceptable as long as we do not have actual completely linear functions. This is very important because most trends and patterns in the data, especially raw data, are nonlinear. To give the reader an intuitive idea, if we want to draw a circle, even a piece-wise set of linear lines will be able to do the job to some level of satisfaction, however, if we have a single straight line, drawing a circle becomes completely impossible. Hence, it is important for the activation functions to be nonlinear.

For the benefit of the reader, we will mention by name one other very recent activation that has shown to perform extremely well. It is called the swish function and was discovered by Google in 2017 Ramachandran, Zoph, and Le (2017).

##### #### Activation function choice for regression problems.

There are two points that have to be considered.

1. Take care of the output of your network. If that's a Real number and can take any value, you have to use linear activation as the output.
2. The inner activations highly depend on your task and the size of the network that you use. What I'm going to tell you is based on experience. If you don't have a very deep network, tanhtanh and ReLUReLU may not differ very much in convergence time. If you're using very deep networks, don't use tahntahn at all. ReLUReLU is also not recommended in some [contexts](https://arxiv.org/abs/1502.01852). You can employ PReLUPReLU in very deep networks. It does not add too many parameters to learn. You can also use leaky−ReLUleaky−ReLU in order to avoid dying ReLU problem which may occur.

Finally, about the other nonlinearity that you've referred; try not to use Sigmoid due to that fact that it's expected value is not equal to zero but half. It's a bit statistical stuff, but you can consider it's roughly hard for a network to learn shifted weights.

Source Post: <https://datascience.stackexchange.com/questions/47751/what-activation-function-should-i-use-for-a-specific-regression-problem>

Stackexchange User: Media - <https://datascience.stackexchange.com/users/28175/media>

#### #### Activation function choice for classification problems.

### Gradient Descent & Stochastic Gradient Descent

Nearly, all of deep learning is powered by one very important algorithm: stochastic gradient descent (SGD). Stochastic gradient descent is an extension of the gradient descent algorithm. It is a method to update the weights of the neural network such as to minimize the cost function.

We first explain the gradient descent algorithm. Mathematically speaking gradient descent is simply:

Where is the weight that is updated, is the learning rate, and is the cost function evaluating overall performance. This can also be written in equivalently in computer science notation:

The gradient descent algorithm is the method used to find the minima of the cost function as a function of the weights. We define the algorithm more formally:

Gradient Descent Algorithm:-

Step 1. initialize waits randomly

Step 2. loop until convergence:

Step 3. compute gradient $\dfrac{J(\text{W})}{\textbf{W})$

Step 4. Update weights

Step 5. Return weights

The most important step in this algorithm is step 3, taking the gradient if the cost function with respect to all the weights. This is done through backpropagation and is discussed in detail below in the next section.

However, even though gradient descent allows us to find the weights to minimize the loss function, it unfortunately has a few drawbacks. One of the main drawbacks is that in this algorithm the gradient in Step 3 and updating of the weights in Step 4 are done for all the data set in the training set and the computational cost is O(n), hence as the training size increases too hundreds of thousands or even millions it actually becomes computationally infeasible to implement this algorithm (Goodfellow, 2016; 152).

**Stochastic Gradient Descent**

Stochastic gradient descent improves upon this algorithm and addresses this main drawback. Stochastic gradient descent simply takes a subset of the training set and trains the data on that. This is even possible from the insight from statistics that the gradient is an expectation.

.

Now using the definition of the expectation, we have the following:

Further, the expectation can be approximated using a smaller subset of the training set. Specifically, on each step of the algorithm, we can sample a minibatch of examples B’ = {x(1), … , x (m’)} drawn uniformly from the training set. The minibatch size m’ is typically chosen to be a relatively small number of examples, ranging to a few hundred. Crucially, m’ is usually held fixed as the training set size m grows even up to billions using updates computed on only a hundred examples. As m approaches infinity, the model will eventually converge to its best possible test error before SGD has sampled every example in the training set. Increasing m further will not extend the amount of training time needed to reach the model’s best possible test error (Goodfellow, 2016; 152-153).

Finally, stochastic gradient descent is a very strong, versatile, and powerful optimization algorithm and in fact has many advantageous properties. Two very important properties of stochastic gradient descent are that it can converge even when the loss function is nowhere differentiable. Secondly, it is able to escape local minimums to reach an even better result something that is not possible in regular gradient descent.

### Back Propagation

We now come to the important Step 3 and describe how the gradient of the cost function is in fact taken with respect to the weights. This method is called back propagation.

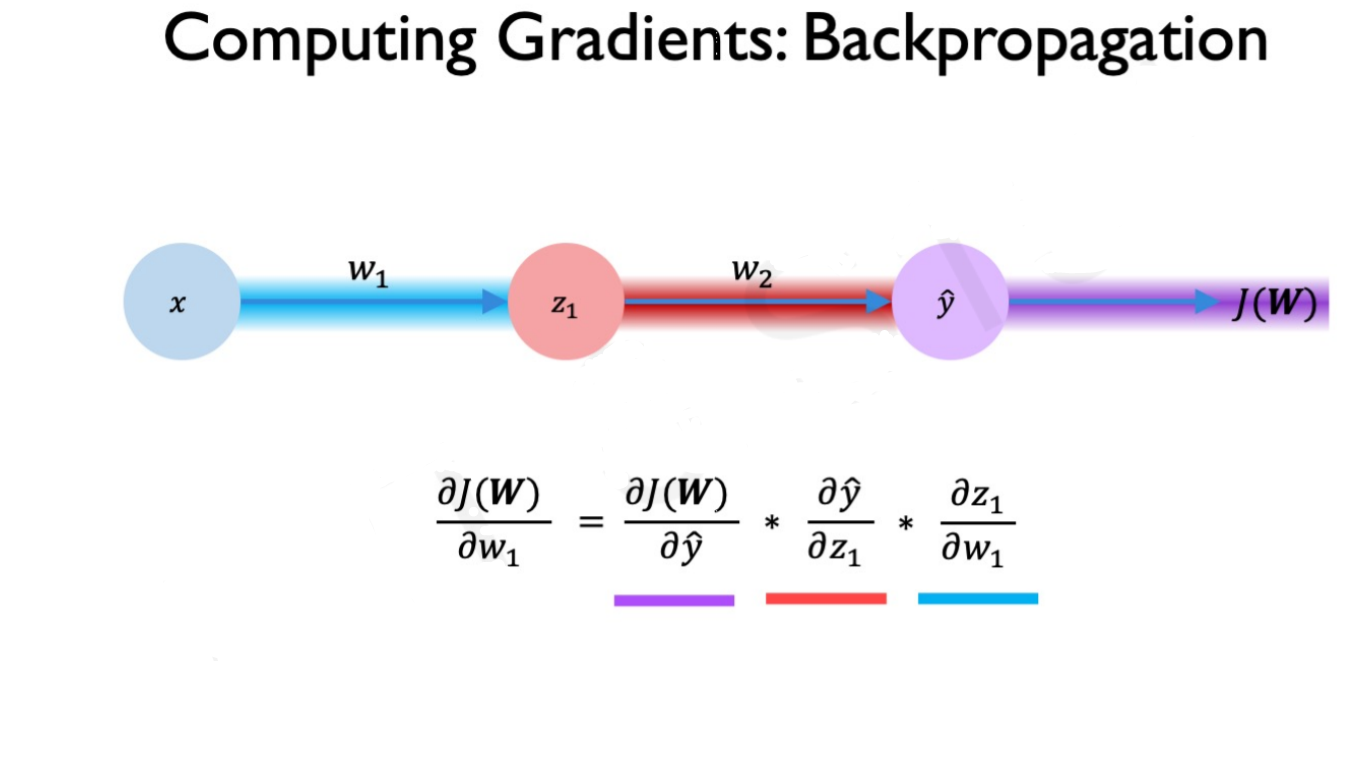
Back-propagation is often misunderstood as meaning the whole learning algorithm. As mentioned previously, back-propagation refers only to the method for computing the gradient, while stochastic gradient descent is used to perform learning using this gradient.

Back propagation essentially can just be thought of as the chain rule. We introduce the chain run in the multidimensional case.

Let . Let g map from , and f maps from . Further, let and .

(Goodfellow, 2016; pg 204).

We illustrate below in Fig. 8 a very simple backpropagation example which is taken from the MIT course on deep learning MIT 6S191 lecture 1 slides. Link to the source material is: <http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf>



![Fig. 8: Applying Chain rule to show backpropagation. Picture obtained from MIT course on deep learning MIT 6S191.](images\_graphresults/backpropagation\_image.png)

### Hyperparameters of neural networks

Hyperparameters of an algorithm, simply put, I consider are all the parameters that the algorithm inherently does not directly learn while training. For example, number of hidden layers in the neural network, the number of neurons (units) in each layer (this can be thought as a hyperparameter for each layer, the activation function for each neuron, and even the optimization algorithm that is used: SGD, adam, nadam, etc. I would consider a hyperparameter. Hence, all parameters that are learned by the algorithm automatically during training are not hyperparameters. For example, weights of a neural network are always learned by the algorithm while training hence they are parameters and not hyperparameters.

Hyperparameters are parameters that are not learned by the algorithm directly and need to be prespecified externally before running the algorithm. Hyperparameters are typically found by trial and error, though there are automated methods like creating an outer loop which compares the performance of different set of hyperparameters (e.g. grid search) to find computationally. We, however, will focus on only giving some insight and general principles for important neural network hyperparameters.

Parameters in a machine learning and AI algorithms are used to tune the algorithm so as to improve the performance determined by some metric. Hyperparameters are used for controlling the general behavior off the machine learning algorithm. One major class of hyperparameters are the parameters that control the model capacity, in other words, the overall number of parameters in the model which we have also sometimes referred to as the model size.

Here is the list of the main hyperparameters:

Algorithmic Hyperparameters

1. Activation function:
2. Loss function:
3. Optimizer:
4. Metric: (the metric of evaluation is in fact is strictly not a hyperparameter, but we include it here for the sake of completeness.)

Regression Metrics

* 1. MeanSquaredError
  2. RootMeanSquaredError
  3. MeanAbsoluteError
  4. MeanAbsolutePercentageError
  5. MeanSquaredLogarithmicError
  6. CosineSimilarity
  7. LogCoshError

NN Structural Hyperparameters

1. Number of neurons in the input layer
2. Number of hidden layers
3. Number of neurons per hidden layer

Source of list of regression metrics (https://keras.io/api/metrics/)

Hyperparameters typically control the model size. If they are allowed to be learned, typically they will always the learned such that the model size is maximized. This will always in turn lead to overfitting. Hence, hyperparameters control the size of the model and are typically determined before fitting the model parameters (Goodfellow, 2016; pg 120) Goodfellow et al. (2016) pg 120. Examples would be the learning rate or the number of neurons in the hidden layer and they have to be adjusted manually. Here machine learning leans heavily towards art Skansi (2018); pg 125 (Skansi, 2018; pg 125).

Another very import hyperparameter is the depth of the neural network, i.e. the number of hidden layers. We will talk in much more detail about the depth later.

## Regularization: Methods to Improve Neural Network Performance

In this section we will discuss several methods used to improve the performance of a neural network, but all of these can also be applied to improve any other machine learning algorithm. We will discuss the five methods: Parameter norm penalties, ensemble method, data augmentation, semi-supervised method, and dropout.

Deep neural networks can have hundreds of thousands, even millions of parameters, hence it is a very flexible model. The problem with its hyperflexibility is the very detrimental disadvantage of overfitting the data. In other words, modeling not only the signal in the data but also the noise. Hence, regularization methods are especially important in deep neural networks for their consistent and high performance.

### The Idea of Regularization:

Regularization is any method that helps with generalizing an algorithm beyond the training set. It is to help improve generalization while keeping the training error low or unchanged (Goodfellow, 2016; ). Regularization methods have existed long before (decades before) deep learning and have been developed and used very successfully in statistics such as linear regression regularization techniques (Goodfellow, 2016; 230).

### Parameter Norm Penalties

The first regularization technique we will discuss is a very broad and successful method, pre-existing before neural networks. It is to add a penalty term to the cost function so as to reduce the overall capacity of the model. This helps reduce overfitting and the generalization error since flexible algorithms such as neural networks tend to also start modeling the noise and not only the underlying trend. If **J** denotes the cost function and is the penalty term, then the adjusted new function to minimize can be represented mathematically in general as below:

Where is a hyperparameter term that weights the relative contribution of the norm penalty term, , relative to the standard objective function **J** (Goodfellow, 2016; 230).

\*\* Parameter Regularization\*\*

The L2 norm penalty is called the weight decay. L2 regularization is also knows as Ridge regression. The idea is to incentivize the weights to be close to 0 simultaneously.

The regularization term or the penalty term for the L2 regularization is as follows:

(Goodfellow, 2016; pg 231)

In the Appendix section we show through mathematical derivation how adding L2 penalty causes the weights to shrink or in other words be closer to 0 for regression.

In essence, adding the penalty term causes X who have higher variance which in turn causes the wait we shrink record the covariates to adjust for the higher variance. (Goodfellow, 2016; pg 234)

This L2 penalty term causes the estimates of the parameters or coefficients (in statistical linear regression terminology) to simultaneously asymptotically approach zero, but they are never equal to zero.

**L1 regularization**

L1 regularization or L1 penalty is also known as lasso. In this case the penalty term is replaced with the following.

This particular penalty causes the coefficients to become zero one by one sequentially with the least useful coefficient becoming zero first followed by the second most important one as the penalty is hyperparameter is progressively increased.

### Ensemble Methods

We will now discuss ensemble methods. People familiar with random forest will know that it is simply an ensemble of trees. The performance of single tree model algorithms do not perform very well, however, when the results are aggregated of many trees are trained on randomly selected subsets of variables and data points their performance outperforms by a wide margin any individual tree. This is the random forest algorithm put concisely, and the crux of how any ensemble method works.

This concept of training many models on randomly selected subsets of the data and features whose results are at the end combined in some fashion, typically through aggregation, can be applied to deep neural networks as well. Several deep neural networks can be trained and their results aggregated to give the final estimates. This is a common approach for improve the performance and typically between 5 to 10 neural networks are used. In fact, Szegedy et al. (2014a) used six neural networks to win the ILSVRC (Goodfellow, 2016; 258).

### Data Augmentation

One of the best ways to improve a machine learning algorithm is to use more data to train it. However, it is not easy to obtain new data and can be very expensive. Hence, one method of overcoming this is called data augmentation. It is the process through which new generated data is created. This can be thought of new fake generated data. This is easier to do with some types of data in machine learning problems more so than others. For example, for classification problem for classifying objects one can generate data that is nuffic generated data by simply performing transformations to the data that one already has and to train this new data and train the model with this new data so it can also beware invariant under transformations. A simple example of this is object classification where gnu fake data can be generated by translating zooming in zooming out and rotating the pictures objects in the pictures hopefully training the data to become to correctly identify the object under different transformations. This is a very powerful technique that can not only increase the data but also make the algorithm more robust. However, data augmentation is not always possible and sometimes very hard to achieve.; for example when predicting house prices based on house features it is very difficult to create new data since it would be akin to randomly creating new data or false data. (Goodfellow, 2016; 240).

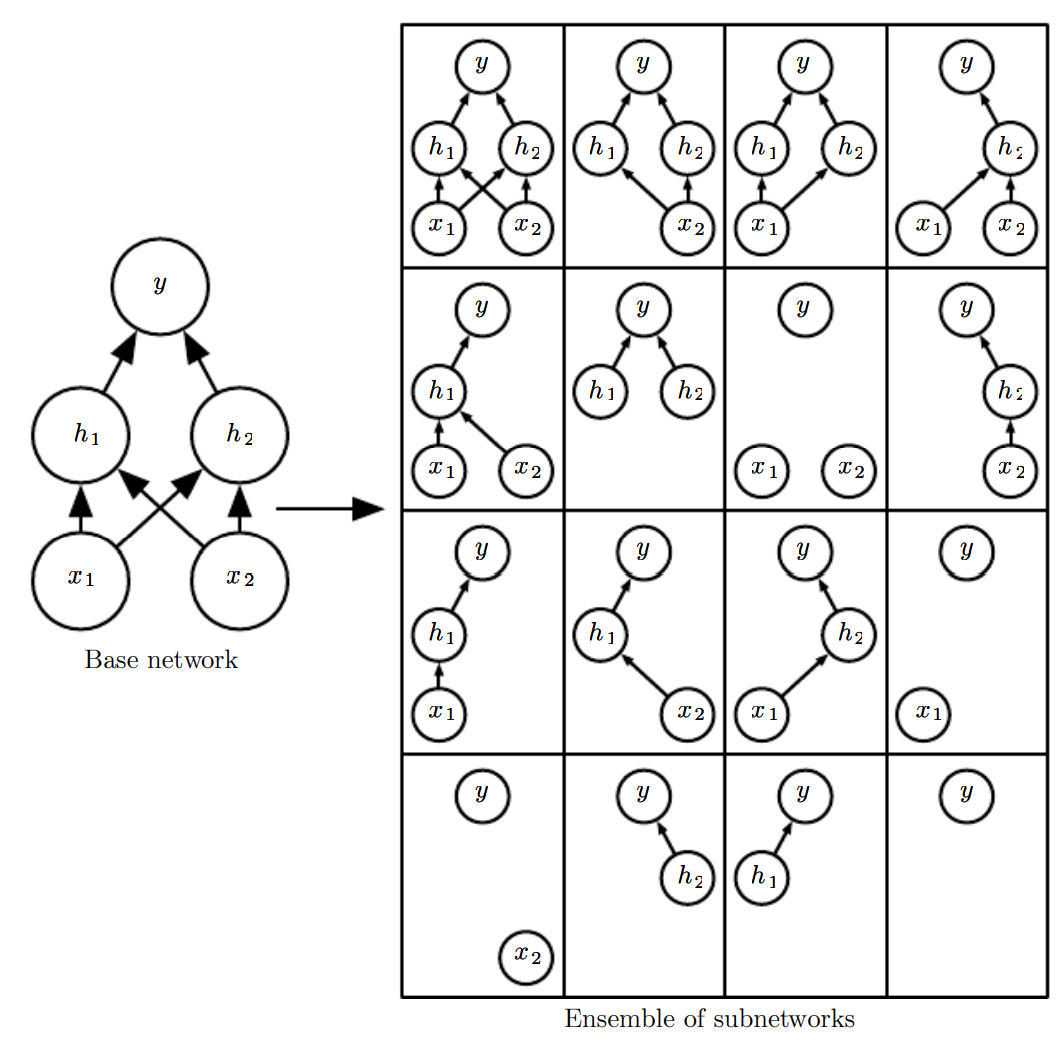
### Semi-Supervised Learning:

Another method for improving performance of deep learning models or any machine learning algorithm for that matter is to also perform unsupervised learning along with supervised learning. One very popular example which many people are aware of is performing PCA analysis and then on the transform data running some machine learning algorithm or deep learning algorithm for classification problem. Another method which would combine supervised and unsupervised learning another method another way of including unsupervised learning in a deep neural network would be to perform some form of cluster analysis such as K means clustering. Once the data is grouped into categories then a new variable can we create it new variable can we create, and this can be passed into the neural net as a additional input variable (Goodfellow, 2016; 243).

### Dropout

In neural networks, dropout regularization technique consists training the data on all subnetworks that can be formed of the neural network by removing non-output units. The main method used in modern neural networks for removing a particular unit is to is to multiply the neuron’s output by zero. Note that the weights are not made zero which is sometimes thought as how dropout is achieved. One simple reason that this is not done is that there would be no learning as the weights through backpropagation and stochastic gradient descent need to be adjusted to make the neural network improve its performance. Hence, in a neural network the weights must not be inherently fixed by design.

In Fig. 9 below, dropout is excellently illustrated as it shows all the subnetworks within the ensemble that are trained. There are some subnetworks that have no path from input to output. Those are simply altogether dropped.



![Fig. 9: Example of how dropout works for a network showing the original or base network and all the subnetworks resulting from dropout regularization (Goodfellow, 2016; 260).( images\_graphresults/Droupout\_subnetworks.png)

One thing what the dropout regularization causes the model to be larger which is a cost that is paid. Another cost is that it takes longer to train any takes more iterations to run and achieve a particular accuracy.

Two very advantageous features of dropout regularization methods are that firstly, they are very computationally inexpensive, and secondly, they can be applied to a wide range of models and algorithms (Goodfellow, 2016; 258).

We would like to mention two scenarios where drop out does not in fact improve the performance substantially. The first is when there is a lot of data. In this case the algorithm is already able to capture the underlying trend and distinguish the noise from the signal simply because there is so overwhelming data available. It is rare to have show my data available there having drop out regularization not improve the generalization error of the algorithm or in our particular case the neural network. However, this is indeed observed. Second scenario is when there is too little labeled Training data available. In this case, there is not sufficient data at all to even identify the underlying trend properly. In such scenarios, unsupervised or data augmentation regularization methods perform better then drop out (Goodfellow, 2016; 265).

# # Application of Neural Networks with Applications

We will be doing all applications in python using the keras package.

## ## Setting up the software.

Sources used in this section are:

### ### Installing Python

Install python through anaconda. I recommend the full anaconda which will be about 4GB and will demonstrate all the steps.

We would recommend installing packages through anaconda navigator.

If there is a package not found in the list, then install it using the pip install command.

#### #### Setting up the Environment

### ### Installing Tensorflow

First install tensorflow. Do not install it using Anaconda. Install it from the method directly on the TensorFlow website.

### ### Installing Keras

## ## Neural Networks to Predict Popularity of the Ted Talk.

Analysis Plan:

Univariable analysis with only 1 hidden layer and without any regularization.

Search from full set the best performing hyperparameters.

Multivariable analysis with only 1 hidden layer and without any regularization.

Search from a subset the best performing hyperparameters.

Multivariable analysis with deep neural network and still without any regularization.

Multivariable analysis with deep neural network with regularization.

### ### Data

The Ted Talk data contains xx variables.

### ### Univariable Analysis

We will predict the number of views of a Ted Talk using the number of comments.

We will implement a very simple neural network with only one hidden layer.

Structure of Neural Network:

List of hyperparameters and the combinations we will implement:

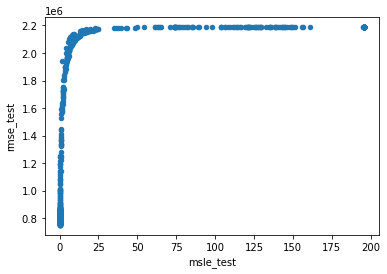
Best performing combination of hyperparameters. Best performance determined by the rmse metric.

### ### Multivariable Analysis

We will predict the number of views using the following variables:

Shallow Multivariable Neural Network (1 hidden layer)

Hyperparameter search results:



![Fig. 10: Plotting raw test mse vs. test msle values for shallow multivariable neural network hyperparameter search.](images\_graphresults/msetest\_vs\_msletest\_msnn\_hyperparasearchresults\_v2.png)

Book Question) What does the shape of this graph mean, what information does it convey, and why is the shape of the graph like this?

## ## Comparing Performance of Univariable with Multivariable Shallow Neural Networks

Chart, bar chart

Description automatically generated

![Fig. 11: Plotting test rmse values for shallow univariable and multivariable neural networks.](images\_graphresults/test\_rmse\_usnn\_vs\_msnn.png)

### ### List of issues in Neural Network Application (also applicable to machine learning).

Most issues you will note will be surrounding the data, particularly the data not fitting in the required mold of the model for the optimal functioning of the model. This issue is strongly felt and extensively delt in statistics. We will be hence borrowing and incorporating the wisdom established in the field of statistics.

Issues surrounding the data:

A common issue in the data is being unbalanced. It can be unbalanced in the explanatory variables (called features by computer science people) or in the variable of interest, for example the number of people who survived being very few where we want to either understand what increases the probability of survival or predicting survival. The unbalanced nature of the variable of interest leads to issues. Why is this so? Why does unbalanced data cause problems? Before we give the mathematical answer, we would like to develop the reads skill of gaining intuition through using the power of analogy. How would this problem of unbalanced binary variable of interest translate to a continuous variable?

The continuous version would be to have a very small range of the continuous variable. In the extreme it would that y is equal to a single value for all data points. Can the reader see why this will be a problem to perform any statistical analysis or mathematical calculation? Mathematically in fact there is not issue here, any quantity can be calculated. Statistically, however, since there is no variation in the data most statistics when calculated will fail. It is like trying to calculate the slope of a line when you are only provided one point on the line. We need at least two points on the line to find the slope. The problem is the lack of degrees of freedom – the lack of variability. The same issue translates in statistics. The same issue permeates to the continuous variable issue where there is not enough variability in the data to get a good a reliable estimate, let’s say, for the coefficients of the variable of a model. The same issue reaches the binary or discrete variable case when the classes are unbalanced.

In deep learning the solution to this is very simple for classification problem. Simply increase the threshold probability of giving it a particular class instead of the typical 50% to 65% or even higher depending on how unbalanced the data is. The higher threshold will cause the algorithm to assign or declare a particular class for a data point only when the algorithm obtains a probability or deems the probability to be 65% or higher.

Now the question for the reader is why does this help mitigate the issue of unbalanced data?

### ### Lessons on Structured data for Deep Learning.

<https://markryan-69718.medium.com/deep-learning-on-structured-data-part-1-7f08584b9883>

<https://markryan-69718.medium.com/deep-learning-on-structured-data-part-2-5d888b966e9>

# # Creating Images to Represent Neural Network Architectures.

Resources to draw neural networks using different representations:

Best tool to draw your model architecture: <https://www.electronjs.org/apps/netron>

If the neural network is given as a Tensorflow graph, then you can [visualize this graph with TensorBoard](https://www.tensorflow.org/guide/graph_viz).

Source: stack overflow.

TensorBoard Instructions for Windows:

First import TensorBoard

Import tensorflow as tf

from ts.kears.callbacks import TensorBoard

First we must save each model as a unique name with the important hyperparameter values in the very name of the model. To this end we use the code below:

NAME = "mdnn\_4hl\_inverted\_triangle\_archi\_2\_dropout\_layers\_defaulthyperpara\_{}".format(int(time.time()))

Now that we have create the name of the model, we need to create an TensorBoard object.

tensorboard = TensorBoard(log\_dir='logs/{}'.format(NAME))

This tensorboard object need to be passed into the model.fit() under the argument ‘callbacks’. So in the model.fit() code we need to add the ‘callbacks’ argument.

This will look like the code below.

model\_fit = model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, verbose=verbose, validation\_split=validation\_split, callbacks = [tensorboard])

Now we need the results of the TensorBoard to be displayed. The below code always to embed tensorboard in jupyter notebook, display it, and interact with it within the notebook.

%tensorboard --logdir logs

possible errors:

'tensorboard' is not recognized as an internal or external command,

operable program or batch file.

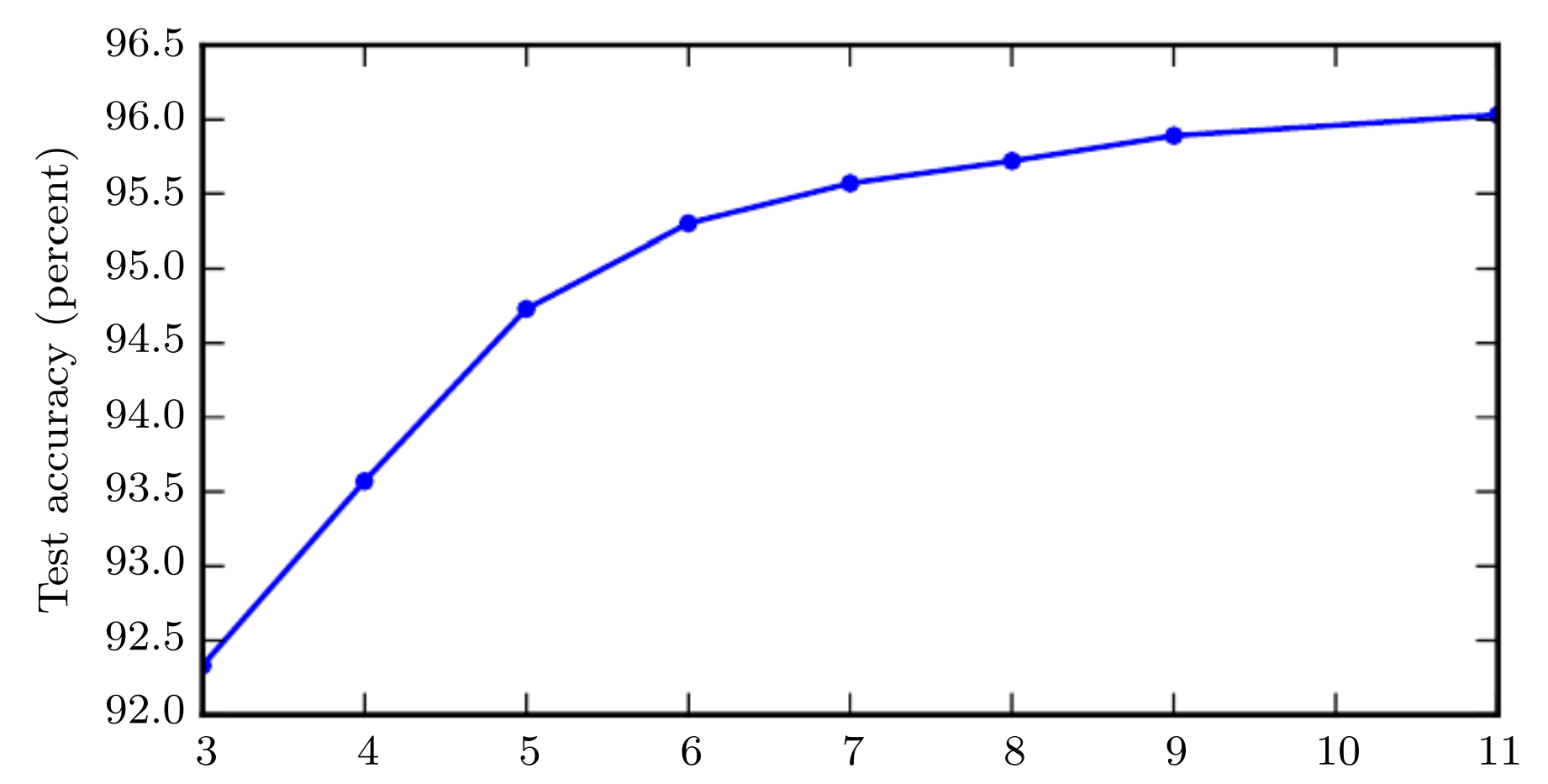
# # Interpretability for TensorFlow 2.0

https://www.sicara.ai/blog/2019-07-31-tf-explain-interpretability-tensorflow?utm\_source=NL+EN+Tech&utm\_campaign=5bed7733ef-pipe\_tech\_email&utm\_medium=email&utm\_term=0\_cf6288210a-5bed7733ef-573417442

# # Results and Insights: Deep Neural Network

First, in Section 1 Feedforward Neural Network, the *universal approximation theorem* says that there exists a neural network large enough to achieve any accuracy. However, it does not say how large this NN would have to be. In the worst case scenario, it could be possible that a feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly. Therefore, using deeper models can reduce the total number of neurons required to represent the desired function and can reduce the amount of generalization error (Goodfellow, 2016; pg 199).

As Goodfellow says in his book “choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions. This can be interpreted from a representation learning point of view as saying that we believe the learning problem consists of discovering a set of underlying factors of variation that can in turn be described in terms of other, simpler underlying factors of variation” (Goodfellow, 2016; pg 199). Further, empirical results show that greater depth does seem to result in better generalization for a wide variety of tasks (Bengio et al., 2007; Erhan et al., 2009; Bengio, 2009; Mesnil et al., 2011; Ciresan et al., 2012; Krizhevsky et al., 2012; Sermanet et al., 2013; Farabet et al., 2013; Couprie et al., 2013; Kahou et al., 2013; Goodfellow et al., 2014d; Szegedy et al., 2014a). See Fig. 10 and Fig. 11 for examples of some of these empirical results. This suggests that using deep architectures does indeed express a useful prior over the space of functions the model learns (Goodfellow, 2016; pg 201).

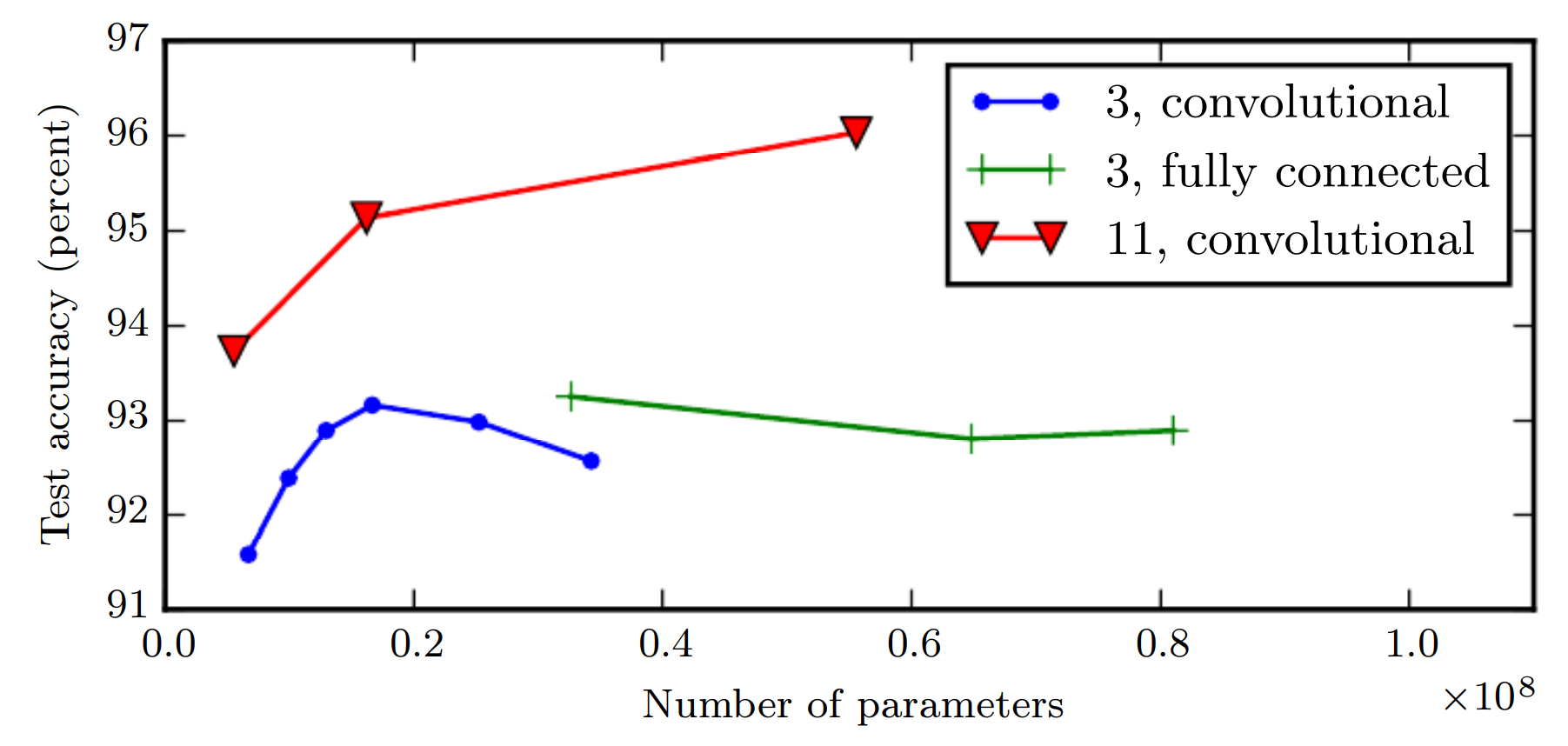


![Fig. 12: Test set accuracy increases with increasing depth. Hence, these empirical results show that deeper neural networks generalize better (Goodfellow, 2016; pg 202).] (images\_graphresults/Control\_test\_accuracy\_increasing\_withNNdepth\_image.png)

However, one could argue that the test accuracy in Fig. 10 increases simply because the model size is bigger. In other words, the increase in number of parameters causes the increase in test accuracy rather than specifically the dept, number of layers in the NN.

A control study done by Goodfellow et all, 2014 shows that it is in fact increasing the dept of the neural net that improves the test accuracy. Fig. 11 below shows the summary of the results, where neural networks of different depths have their number of parameters increase (making the models bigger) while keeping their depth constant.

The deeper neural network always performs better than the shallower neural networks for a given number of parameters (i.e., for a given model size). Further, as the number of parameters is increased, the deep neural network’s test accuracy consistently keeps improving and is able to take advantage of the increased number of parameters, whereas the shallower neural networks’ test accuracy stops improving after a certain number of parameters in the models is reached. In fact, the test accuracy starts decreasing as the number of parameters increase beyond a certain number. Both of these insights are very useful to know and can be clearly seen in Fig. 11 below.



![Fig. 13: Controlled study of neural networks of different depths (3 and 11), to see how the increase in number of parameters (i.e., the model size) and test set accuracy interact with the depth of the neural network. The deeper neural network of depth 11 has higher accuracy than the shallow neural networks of depth 3. Further, the deep neural network’s test accuracy keeps consistently increasing as the number of parameters increases, whereas the shallow neural networks’ test accuracy either plateaus or starts becoming worse (Goodfellow, 2016; pg 203).] (images\_graphresults/Control\_test\_accuracy\_increasing\_withNNdepth\_image.png)

Now, we would like to offer some intuitive insight as to why deep learning performs much better and is such a powerful artificial intelligence tool. One very distinct and unique feature of deep learning is that it's performance continues to consistently improve with larger and larger data which is in start contrast with many though not all other statistical and machine learning methods. Other statistical methods’ performance plateaus after a certain amount of data and adding anymore will not improve it.

Neural networks as we mentioned before, can be thought of as many different small units analysis very well that are very basic models all performing similar tasks and all solving the similar problem all of them working in dependently in parallel because the neurons within one layer are not connected with the other neurons within the same layer hence within a layer the neurons act independently. Hence, at each layer, we can think of the many neurons as many small models performing independent tasks whose results are then given of to the next layer of neurons who again perform the similar tasks again independently. Therefore, a neural network could be thought of as many models performing similar tasks in each layer independently. As these independent processes grow they are able to identify more and more complex features and patterns.

To conclude, feedforward networks are universal approximators of functions. However, a sufficiently deep feedforward network can have an exponential advantage over a network that is too shallow. Further, decrease in model size by making the network deep also leads to improved statistical efficiency.

# # Discussion and Conclusion

To conclude this report, we come back to our two original questions: First, why are deep neural net work networks better then shallow networks when shallow neural networks can also approximate any function to any desired accuracy? The reason is that deep neural networks can represent a function or representation with a much smaller overall model size which is in some cases can be exponentially smaller compared to if a shallow neural network would be used instead; hence, have far fewer total parameters to estimate and compute. Therefore, the advantage is of feasibility and practicality.

Secondly, what are the advantages of deep learning that have made it the most successful artificial intelligence algorithm? It is that the performance of deep learning continues to consistently improve with more data, whereas with other artificial and statistical models their performance plateaus. Deep learning can represent and perform far more complex patterns and functionality not possible by other algorithms. For example, deep generative models can create completely human looking fake knew faces. A task which is extremely complex and not possible by many by other algorithms. And lastly, deep neural networks can perform feature selection themselves and they can do so directly on the raw data provided to them.

**\*\*Disadvantages of deep neural networks.\*\***

One disadvantage is, after the data has been cleaned, the model setup and configuration take a lot more work: how many hidden layers to have, how many neurons per layer, what should be the activation function, what regularizations should be used and what should be their parameters, the batch size, the number of epochs, the choice of loss function, the optimization method (adam, nadam, SGD, etc.) among others. And most of these if not all of them have to be chosen through experimentation since as of now there is no other comprehensive methodological way established to do them. Hence, very deep expertise developed through work experience is required. If this is compared to any other machine learning method, such as XGboost, elastic nets, or random forest regression, after the data cleaning, there are a very small number of hyperparameters that need to be experimentally determined compared to deep neural networks.

The second disadvantage, which is the most well known and most brought up is the computation resources required to not only train the model but also find the optimal hyperparameters for the specific data set in question. Computational resources required, both in terms of compute power and compute time, grow {exponentially? {what is the growth rate? Need citation}}very quickly as the model complexity increases and the amount of data available (though for several optimization methods for the weights of the neural network, such as stochastic gradient descent, compute time stays constant as the sample size of the data increases).

**\*\*Optimization versus Learning.\*\***

We would like to share a interesting insight with the reader: what is the difference between learning, as in machine learning, and what is optimization. The term learning has been used in terms of AI all good algorithms. The reader must be aware of the term learning with respect to many AI algorithms including the cohort of machine learning algorithms.

In machine learning typically the performance of some algorithm is Evaluated through some particular measure. This measure in machine learning is improved only indirectly. What is optimized is always some cost function J which contains the design and information of the problem that is being solved. Only through the optimization of J is the measure of performance improved. If this word this word this work this word this word this word this war this war if this were in optimization. In optimization what is optimized is the measure off performance itself directly. This indirect optimization of performance through the cost function is how learning as in machine learning for any algorithm defers from optimization.

# References

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning*. Vol. 1. 2. MIT press Cambridge.

Ramachandran, Prajit, Barret Zoph, and Quoc V Le. 2017. “Searching for Activation Functions.” *arXiv Preprint arXiv:1710.05941*.

Skansi, Sandro. 2018. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer.

Zell, Andreas. 1994. *Simulation Neuronaler Netze*. Vol. 1. 5.3. Addison-Wesley Bonn.

# Appendix

## Definitions and Further Explanations.

### ###Tensor:

We will explain here what at tensor is. As vector generalizes the idea of a number to more than one dimensions, and matrix generalizes the idea of a vector from only columns or rows to arrays with column and rows, hence, possessing what we call meta-dimensions. We will denote meta-dimensions the letter capital M. A scalar has zero meta-dimensions, a vector only has one meta-dimension, a matrix has two meta-dimensions. Each meta-dimension can have any number of dimensions. Hence, all objects of meta-dimension 1 are vectors, similarly as all vectors with dimension 1 are scalars. All objects with meta-dimension 2 are matrices. Tensors are simply objects that have M meta-dimensions.

### ###Functions:

What are functions?

Functions perform a function. Some form of function.

Functions take in an input and produce an output. They are rules.

Input 🡪 output

Functions have three main representations: description, equation, and graph.

When we talk about functions we denote them by the letter “f” because as it is better than writing the whole work function every single time.

The set of allowable inputs of a function is called domain of the function. This could be restricted naturally or deliberately. The set of outputs of a function is called range of the function.

As we denote a function by the letter “f”, we denote the domain by a fancy capital D and we denote range by capital R.

Now you know different types of numbers. Whole numbers (also called integers), fractions, decimals, positive numbers, negative numbers and many more. Now, if you simply want to refer to all numbers what do you call them? Conveniently, when you simply want to refer to all number, not surprisingly mathematics have also given them a name, and they are called real numbers. Now gain, as you will come to realize, instead of writing complete words we prefer to use letters or symbols to represent them. Hence, real numbers are denoted by a fancy capital |R.

So a function has three important components the domain (D), the range (R), and the rule that takes in the domain and produces the range denoted by the letter “f”. All of this is put very neatly in a very compact and efficient notation below.

f: D –> R and the most common domain and range of a function are real numbers:

f: |R 🡪 |R

You will note, compactness, efficiency, and clarity are highly valued in the language of mathematics – and indeed, mathematics is a language, as English, French, Urdu, Music notation, and is very fun to learn and master.

Another way of thinking about functions is that they link a set of things to another set of things. In math most things can be represented as numbers hence, functions are rules to link set of numbers to set of numbers.

## ##Derivations:

### ###Deriving deep learning loss functions using Maximum Likelihood method.

Binary Cross Entropy Loss

(Ketkar, 2017; pg 25)

(Multiple) Cross Entropy Loss

(Ketkar, 2017; pg 25-26)

### ###SDG Expectation

The key principle in solving neural networks is to solve the optimization problem of the network. This optimization problem can be defined by and expressed in terms of the loss function. This loss function also in kind represents the underlying learning system. The problem of optimizing the loss function can be formulated in a popular statistical formulation that has very successful theoretical results. The optimization of the loss function can be represented as minimizing a cost function which is expressed as the expectation of the loss function as shown below. If J represents the loss function and C represents the cost function, then through the statistical formulation where C can be represented as the expectation of J the loss function. As shown below.

.

Now using the definition of the expectation, we have the following:

Where dP is an unknown probability distribution that characterizes the problem to learn.

The minimization of such a cost may be achieved with a stochastic gradient descent algorithm:

With some restrictions on J and C, this algorithm converges, even if J is non differentiable on a set of measure 0, as mentioned before.

If the training set is finite, then the distribution dP(z) is discreet. Hence equation 2 can be expressed as below.

Using the above 2 formulas equations 3.2 and equation 3.3 the gradient descent algorithm can be expressed explicitly as the following:

Each iteration involves a sum over the N examples zi. The gain t is either a positive scalar or a symmetric positive definite matrix. This algorithm is sometimes called the total gradient algorithm. It is known to converge to a local minimum of the cost.

### Derivation showing that the L2 penalty causes the weights to shrink.

We now show, through mathematical derivation, how adding L2 penalty causes the weights to shrink, or in other words, be closer to 0 for regression. We begin with the general form of the cost function with the regularization term:

Where , hence we have:

Where

The matrix which is proportional to is replaced by the term .

It can be seen from the formula above adding the penalty term causes X who have higher variance which in turn causes the wait we shrink record the covariates to adjust for the higher variance. (Goodfellow, 2016; pg 234)

### Potential Research Idea

Huge advancement has been achieved in artificial intelligence by mimicking the neural network off the brain. One new potential area of research that I feel can lead to a completely new shift in advancement not only in neural network but in artificial intelligence is to imitate another feature of the brain’s neural network. This feature is the true reason the human mind and humanity is one of the most successful species. Based on feedback of the experiences the mind encounters it can grow new neurons. Hence, this could be a potential area of research.

## List of pandas dataframe methods.

Definition of terms: First of all, df will indicate a pandas data frame object.

There are two version of, I would say, a method. A method and a method function.

Methods are those that are of the exact following form: df.method and that is it. Just the name of the method. If you write these with a parenthesis it will fail and give and error. Do not write a parenthesis.

These methods need to be burned into your memory. There is no logic as to why they fail when a parenthesis is placed at the end of them.

This are the important methods to be used for dataframes.

Df.shape # give the number of columns and rows in the data frame.

Df.columns # gives the names of the columns of the data frame as an index type object.

Df.

Method Functions are those that are of the form: df.method(). They take a parenthesis at the end. Further, they also, for certain method functions take arguments in the parenthesis. For example the method function .drop() must be provided with the name of the column and axis choice as follows: df.drop(‘variable\_name’, axis = 1). This will keep all the columns and rows except the one indicated.

df.drop(‘variable\_name’, axis = 1) # This keeps all the variables except the one which is dropped.

Df.to\_csv(‘path/name\_to\_save\_under.csv’) # This writes the pandas data frame as a csv file.

These are very useful ones. However, for the strong will I would recommend them to go through all the methods of a pandas dataframe.

Below is the code that was found on stack exchange which provides you with the list of any object and its methods.

CODE CHUNK

import pandas as pd

df = pd.DataFrame([[10, 20, 30], [100, 200, 300]],

columns=['foo', 'bar', 'baz'])

def get\_methods(object, spacing=20):

methodList = []

for method\_name in dir(object):

try:

if callable(getattr(object, method\_name)):

methodList.append(str(method\_name))

except:

methodList.append(str(method\_name))

processFunc = (lambda s: ' '.join(s.split())) or (lambda s: s)

for method in methodList:

try:

print(str(method.ljust(spacing)) + ' ' +

processFunc(str(getattr(object, method).\_\_doc\_\_)[0:90]))

except:

print(method.ljust(spacing) + ' ' + ' getattr() failed')

get\_methods(df['foo'])

## ## Questions

Questions on Architecture of neural network.

Q1) Why do we need to specify the number of neurons in the input layer when the input dimension is already specified.

Q2) Given the input dimensions of a neural network what should be the number of neurons in the input layer? Are there any domain restrictions (strict restrictions, i.e., cannot be zero or less than the input dimension, the neural network will fail) and are there any restrictions for optimal performance (soft restrictions, i.e. can be more than the number of input dimensions and will not cause the neural network to fail but the results would be inferior to those that were within the restrictions)?

Q3) Prove for each of the loss functions below if min loss value or maximum loss value maximizes the performance. Hence, first determine the direction that maximizes the performance, then, prove that direction: mse, rmse, mae, mape, msle, cosine similarity test, …

Q4) How to compare the different regression metric to one another: rmse, msle, cosine similarity, mae, mape, etc.?