# Report 1: Understanding Deep Learning – Breaking Down the Black Box and Shedding the Light of Intuition.

Table of Contents

[Report 1: Understanding Deep Learning – Breaking Down the Black Box and Shedding the Light of Intuition. 1](#_Toc67474542)

[#Abstract: 2](#_Toc67474543)

[# Introduction 2](#_Toc67474544)

[# Methods 5](#_Toc67474545)

[## Theoretical Framework of Neural Networks 5](#_Toc67474546)

[### Simplest Neural Network: Simple Feedforward Shallow Neural Network 5](#_Toc67474547)

[### Statistics, Probability, and Information Theory in Neural Networks. 7](#_Toc67474548)

[### Neural Networks Structure, Terminology, and Mathematical Representation. 7](#_Toc67474549)

[### Gradient Descent & Stochastic Gradient Descent 13](#_Toc67474550)

[### Back Propagation 15](#_Toc67474551)

[### Hyperparameters of neural networks 16](#_Toc67474552)

[## Regularization: Methods to Improve Neural Network Performance 16](#_Toc67474553)

[### The Idea of Regularization: 17](#_Toc67474554)

[### Parameter Norm Penalties 17](#_Toc67474555)

[### Ensemble Methods 18](#_Toc67474556)

[###Data Augmentation 18](#_Toc67474557)

[###Semi-Supervised Learning: 19](#_Toc67474558)

[### Dropout 19](#_Toc67474559)

[# Results and Insights: Deep Neural Network 21](#_Toc67474560)

[#Discussion 24](#_Toc67474561)

[## Regularization and Features in Neural Networks. 24](#_Toc67474562)

[##Optimization versus learning 25](#_Toc67474563)

[#References 25](#_Toc67474564)

[# Appendix 25](#_Toc67474565)

[## Definitions and Further Explanations. 25](#_Toc67474566)

[### Tensor: 25](#_Toc67474567)

[### SDG Expectation 25](#_Toc67474568)

[### Derivation showing that the L2 penalty causes the weights to shrink. 26](#_Toc67474569)

[### Potential Research Idea 27](#_Toc67474570)

# #Abstract:

\*\*Goal:\*\* We answer why is deep learning one of the most successful and popular AI algorithms, we also answer in depth what is deep learning – introducing the mathematical foundation and framework on which it is built, as well as giving an intuitive understanding how deep learning and its different components works. Finally, we also, show and explain many advantages that deep learning has.

\*\*Overview:\*\* Deep learning is an artificial intelligence algorithm; 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.

\*\*Methods:\*\* We discuss the different important components of neural networks: artificial neurons, layers, activation function, cost function, backpropagation, and stochastic gradient descent.

\*\*Advantages:\*\* 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 models whose performance plateaus after a particular data (sample) size. Further, it is one of the most versatile algorithms which is able to tackle many different problems, but also a very wide variety of data types: such as sound data, image data, natural language, and structured. Lastly, deep neural networks are able to do self-selection of features directly from the raw data.

\*\*Conclusion:\*\* All these advantages simultaneously found in deep learning is what 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: a deep neural network. Neural networks (NN) are a type of Machine Learning algorithm. 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 extend them to address other problems in artificial intelligence which are not usually included in machine learning such as knowledge representation, reasoning, planning, etc. (Skansi, 2018, pg 51). In this paper, we will cover supervised learning in deep learning only.

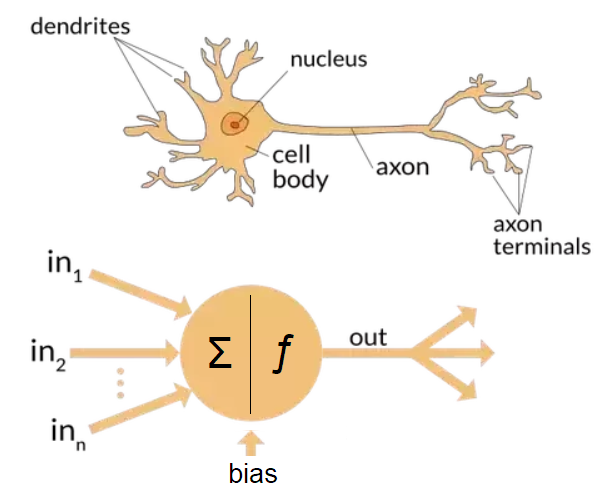
One of the things that make deep learning extremely powerful is the ability to do feature selection and pattern recognition directly from the raw data. For other machine learning algorithms, the features are either predetermined or preselected (e.g. choosing which variables to keep and what transformations to be done), however self-selecting features and directly from the raw data is unique to deep learning.

Deep learning employees three main fields: Mathematics, Probability and Statistics, and Computer Science (Programming) (Skansi, 2018, pg 41).

The idea of 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 neurons interconnect through synapses.

The neuron is a cell that can be electrically excitable and is capable of communicating with other cells. 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, the neuron consists of a single axion which can transmit signals to another cell across synapses and can extend as far as 1 meter in humans. These specialized connections through which the neurons are connected, and signals are transmitted are called synapses. See Fig. 1 below for an illustration.

This idea of receiving signals, processing them, and further transmitting signals is adopted into the concept of artificial neurons. This idea of inputs and outputs is illustrated in Fig. 2 which also shows the parallels between the biological and artificial neuron.



![Comparing a biological neuron with artificial neuron and their components.](biological\_and\_ai\_neuron\_image.png)

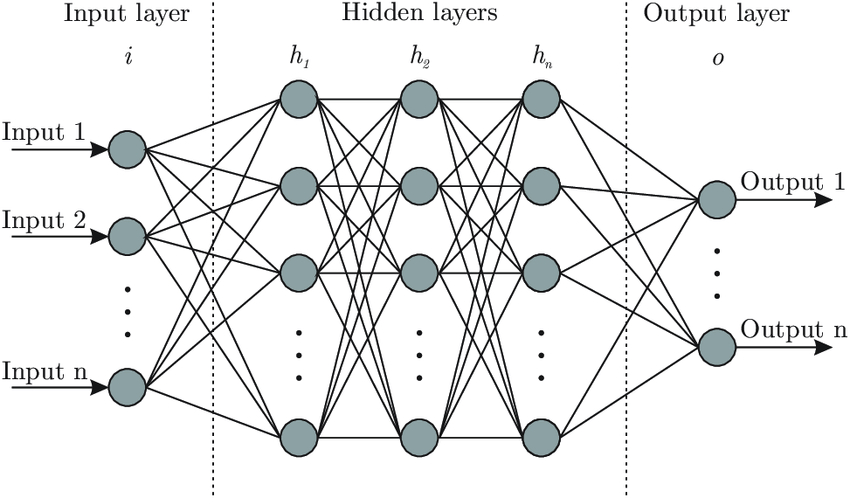
This establishes the concept of an artificial neuron.

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

In the 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 of 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 that are of interest. The number of neurons in the output layer depends on the types of outcomes.

The layers in between the input and output layers are called hidden layers. They are the layers that perform the main calculations and are similar to feature selection of the data in a way, to find patterns that can 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.



![Artificial Neural Network.](Artificial\_neural\_network\_image.png)

To summarize, the input layer, the hidden layers, and the output layer is the basic structure of an artificial neural network and is inspired from the biological structure of the NN in the brain. See Fig. 3 above.

There are more components of the NN, however, these we will discuss in the Methods section when showing formalizing the artificial NN mathematically.

The simplest artificial neural network has three layers, with the input, output, and one hidden layer. 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 as logistic regression is actually a one-neuron neural network.

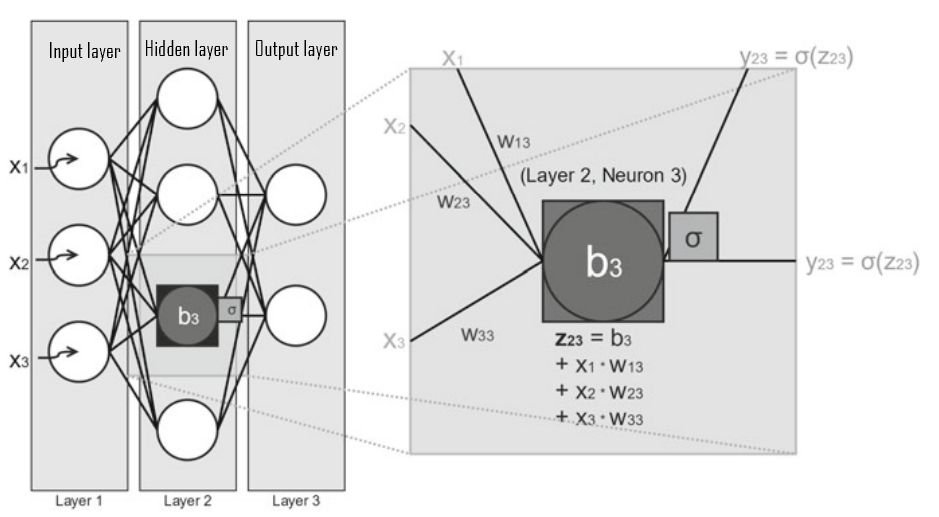
# # Methods

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

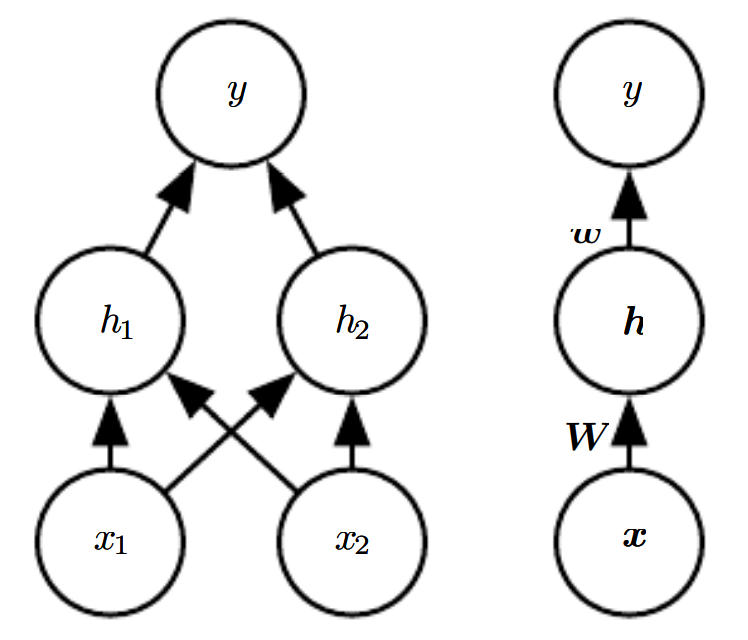
### ### Simplest Neural Network: Simple Feedforward Shallow Neural Network

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).



![A simple neural network with three layers: Input layer, hidden layer, and output layer (Skansi, 2018, pg 80).]( 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 @zell1994simulation. This can be seen in Fig. 5 below.



![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)]( 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

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. 4. 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 $ f\_i, f\_h, \quad and \quad f\_o $ would represent the input, hidden, and output layers of the simple neural network. Hence, if $\textbf{x}$ is the input received by the neurons in the input layer, then the final resulting output of the neural network can be represented as \textbf{y}. Hence, we can represent the neural network as $\textbf{y} = f\_o(f\_h(f\_i(\textbf{x})))$.

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

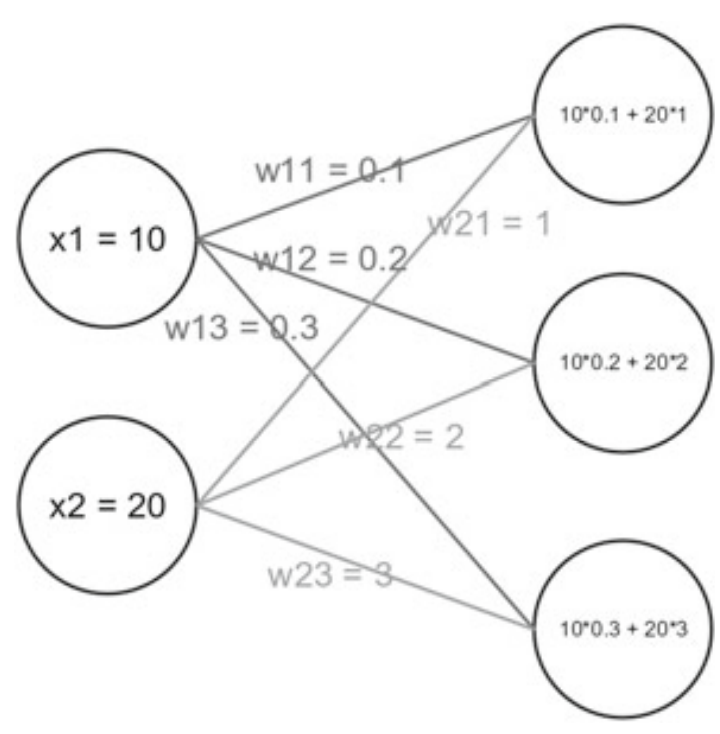
The input layer only serves the function of accepting input. Further, each neuron in the input layer can only hold one input value. In Fig. 4 the neural network’s input layer has three neurons and can hence only take three inputs. These inputs can be represented by variables x1, x2, and x3 which in turn can be mathematically represented as a row (or column) vector of three elements $ textbf{x} = (x1, x2, x3) $ called the input vector. For n neurons in the input layer, the input vector can be represented as $ textbf{x} = (x1, x2, …, xn) $ (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. 6 below shows the input values, the weights of each connection, and final value received by the neurons in the second layer.



![ 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.](NNWeightsInputsFig.png)

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 $textbf{z}$. For example, for Fig 4, it can be represented as: $\textbf{z\_2} = (z21, z22, z23, z24)$. The first index “i” of zij-element represents which layer the neuron output is from, for Fig 4 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 $\textbf{z\_i} = (zi1, zi2, … , zin)$.

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

These calculations are represented as follows for the summands:

$$ z21 = \sum{i = 1}{2} x\_i w\_{i1} , z22 = \sum{i = 1}{2} x\_i w\_{i2}, z23 = \sum{i = 1}{2} x\_i w\_{i3}. $$

Now, in terms of mathematical representation, the weights of the neural networks can be represented as a matrix $\textbf{W}$ 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 $\text{W}$ will be n x m. Hence, for the neural net in Fig. 6 will have dimensions 2 x 3, two rows and three columns.

$$ \textbf{w} = \begin{bmatrix} w\_{11} \left( =0.1\right) & w\_{12} \left( =0.2\right) & w\_{13} \left( =0.3\right) \\ w\_{21} \left( =1\right) & w\_{22} \left( =2\right) & w\_{23} \left( =3\right) \end{bmatrix} $$

The computation of the neural network can be represented as matrix multiplication: $ \textbf{w}^T \textbf{x} = \textbf{z}. $ (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 $\textbf{x}$ is the input received by the neurons in the input layer, then the final resulting output of the neural network can be represented as $\textbf{y}$, where $\textbf{y} = (y\_1, y\_2, …, y\_m)$ 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:

$textbf{Cost function = 1/m ( sum of loss functions error over m samples)}$

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 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 $y\_i$ is the actual value and $\widehat{y}\_{i}$ is the estimated value from the model.

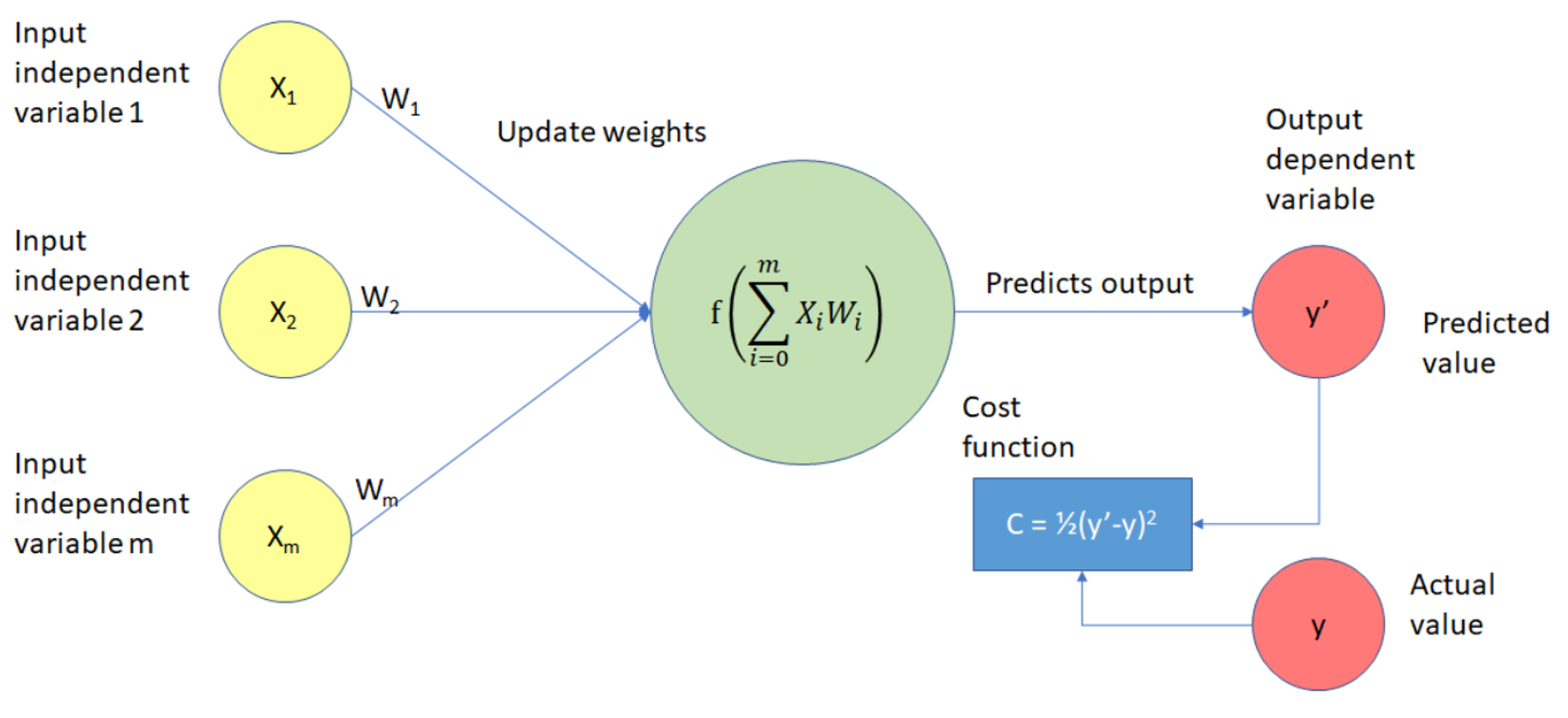
1. Mean Absolute Error (MAE): loss = $ $ this is an $L^1$ regularization or constraint.
2. Mean Squared Error (MSE): loss = $( \sum |(predicted\_i – actual\_i)| )/N$ This is a $L^2$ regularization.
3. Root Mean Square Error (RMSE): loss = $\sqr{ ( \sum (predicted\_i – actual\_i)^2 )/N}$

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

1. Binary Cross-Entropy Loss Function: loss = $\sum^{N}\_{i=1}y\_{i}\log ( \widehat{y}\_{i})$.

Where $y\_i$ is the actual value and $\widehat{y}\_{i}$ is the estimated value from the model.

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



![The schema illustrating how the cost function, in the blue box, is present in the neural network.](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.

#### ### 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 as is illustrated in Fig. 9 below:

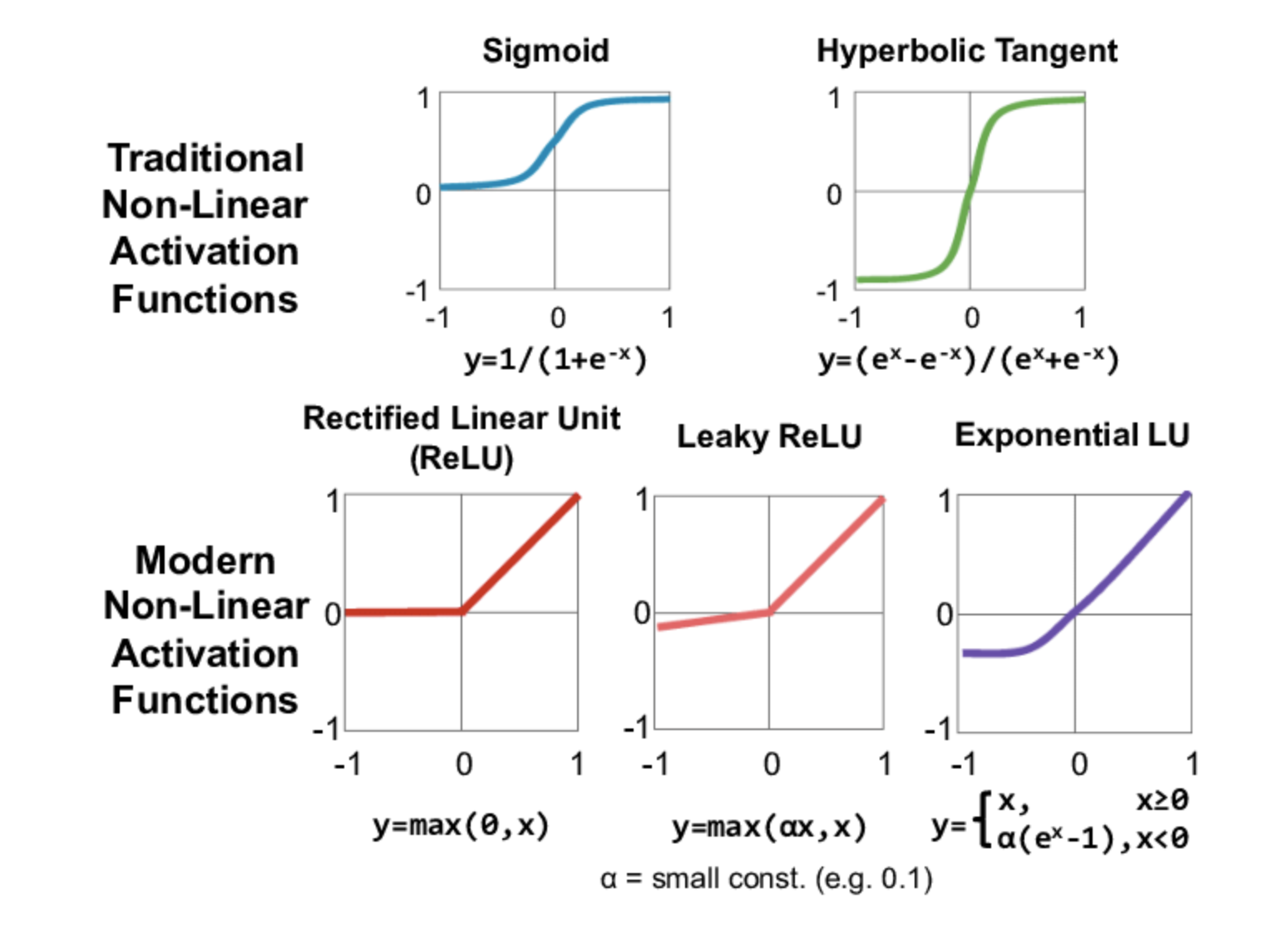
An activation function transforms the shape/representation of the data going into it. These representations are essential for making high-dimensional data linearly separable, which is one of the many uses of a neural network.

Two important properties that are very advantageous for activation functions to have:

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 ( ).

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



![Traditional and more modern activation functions.](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 very recent activation that has shown to perform extremely well. It is called the swish function and was discovered by Google in 2017 @ramachandran2017searching.

### ### 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:

$$ w\_{updated} = w\_{old} - \eta \nabla E $$

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

$$ w \leftarrow w - \eta \nabla E $$

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 $ w\_i ~ N(0, \sigma^2) \forall i$

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.

$$ C(w) = E(J(z,w)) \qquad \qquad \qquad (3.0) $$.

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

$$ C(w) = E(J(z,w)) = \int J(z,w)dP(z) \qquad \qquad \qquad (3.1) $$

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).

Further, 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.

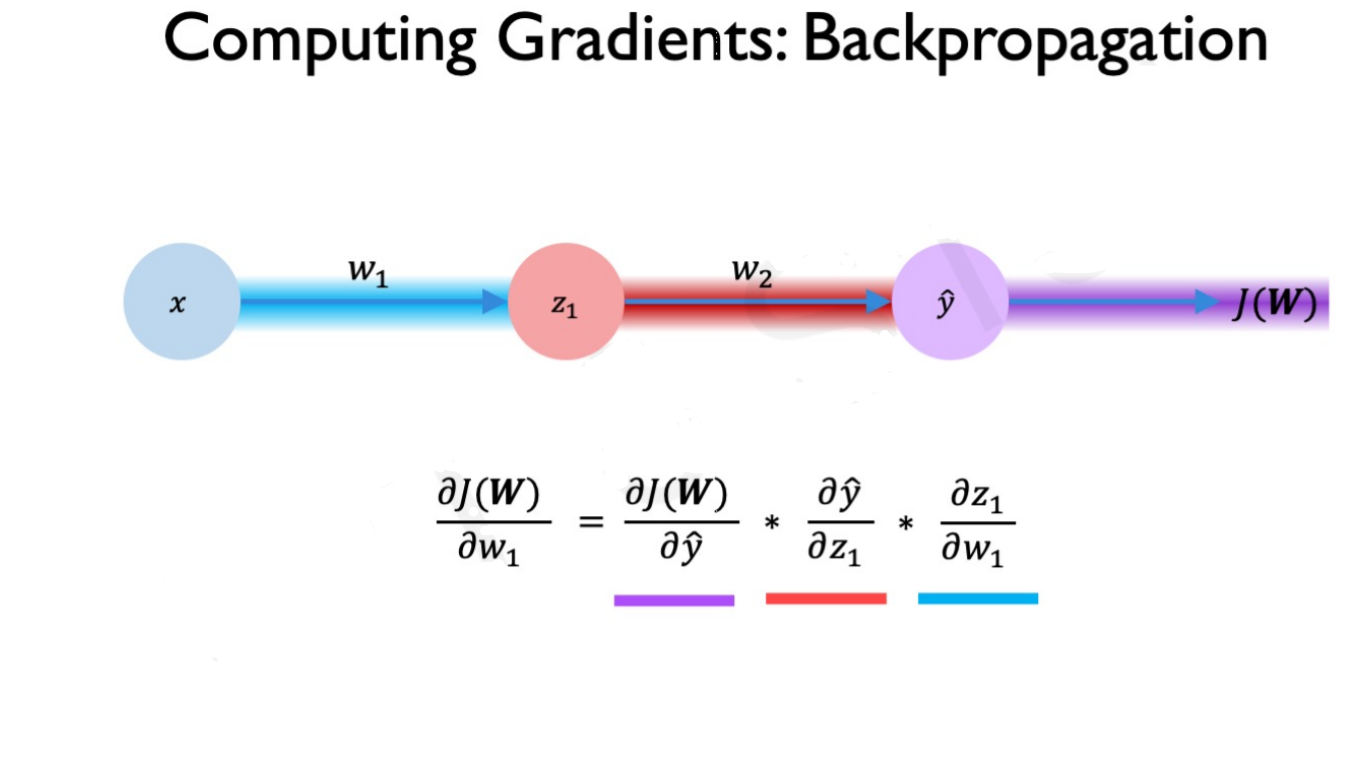
Back-propagation is often misunderstood as meaning the whole learning algorithm. As mentioned previously, back-propagation refers only to the method for computing the gradient, while stochastic gradient descent is used to perform learning using this gradient.

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

Let $ \textbf{x} \in \R^m, \textbf{y} \in \R^n $. Let g map from $|R^m to |R^n$, and f maps from $\R^n to \R$. Further, let $\textbf{y} = g(\textbf{x})$ and $z = f(\textbf{y})$.

(Goodfellow, 2016; pg 204).

We illustrate 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>



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

### ### Hyperparameters of neural networks

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.

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) @goodfellow2016deep 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 @skansi2018introduction; 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 $\Omega\$ is the penalty term, then the adjusted new function to minimize $\overline{J}$ can be represented mathematically in general as below:

$$\overline{J}( \theta ; x,y) = J( \theta ; x,y) + \alpha \Omega( \theta)$$

Where $\alpha \in [0, \inf)$ is a hyperparameter term that weights the relative contribution of the norm penalty term, $\Omega$, relative to the standard objective function \*\*J\*\* (Goodfellow, 2016; 230).

\*\* $L^2$ 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 $\Omega$ for the L2 regularization is as follows call in

$$ \Omega( \theta) = (1/2) \* ||\textbf{w}||\_2^2 $$

(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 $\Omega\$ is replaced with the following.

$$ \Omega ( \theta ) = \| \textbf{w}\| \_{1} = \sum \_{i}w\_{i} $$

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 $\alpha$ 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).

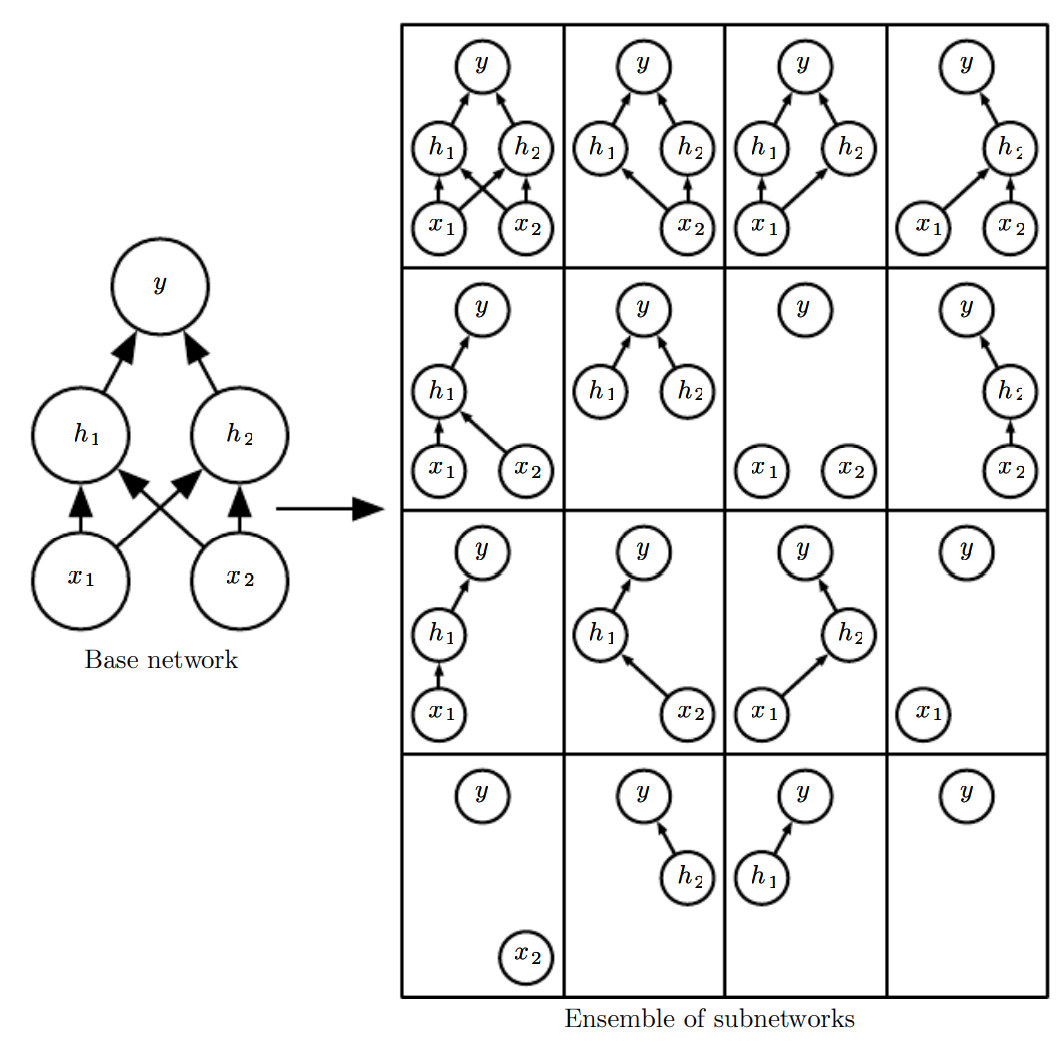
### ###Semi-Supervised Learning:

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. XXX 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.



![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).(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).

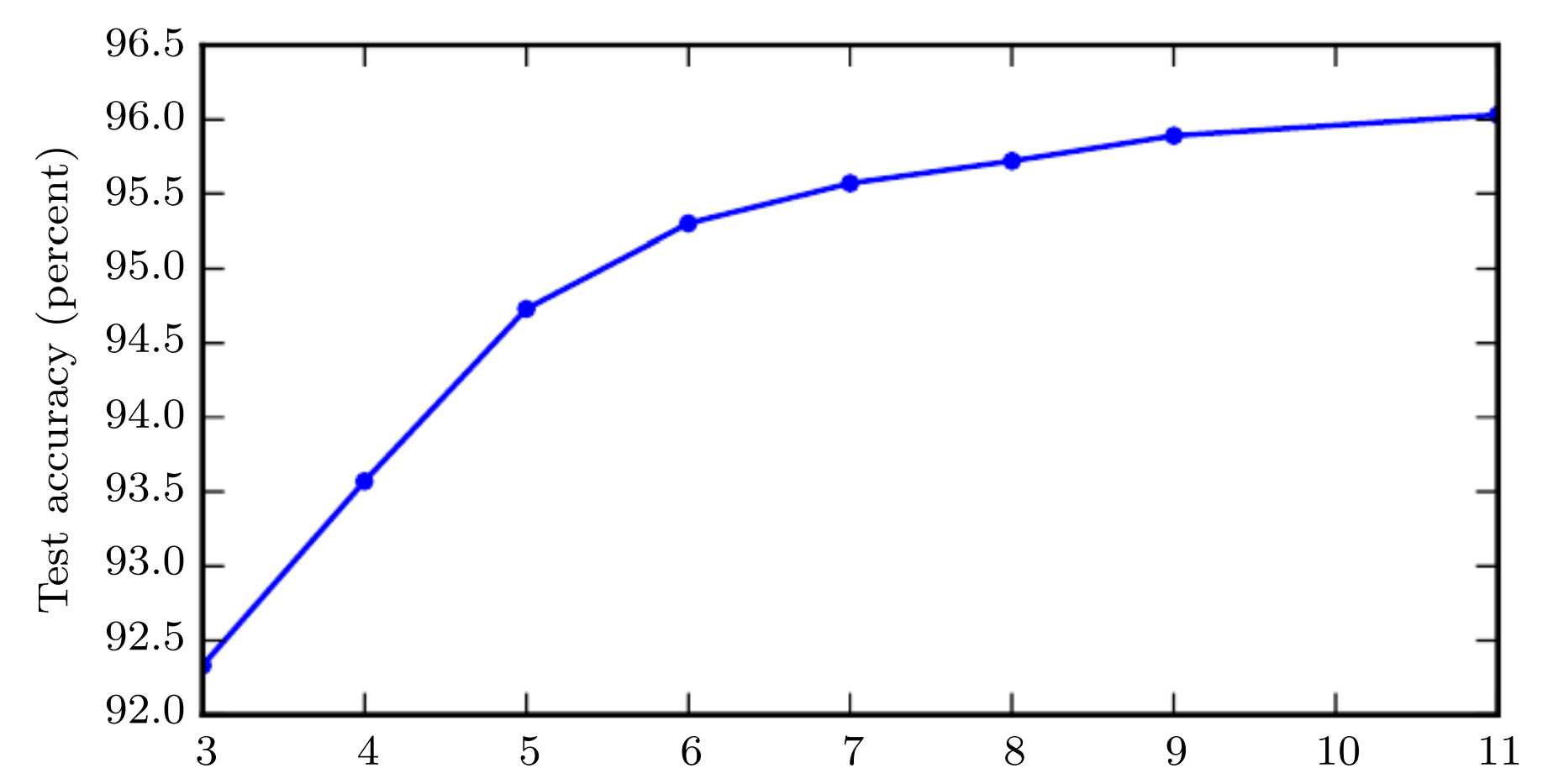
We would like to mention two scenarios where drop out does not in fact improve the performance substantially. The first is when there is a lot of data. In this case the algorithm is already able to capture the underlying trend and distinguish the noise from the signal simply because there is so overwhelming 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).

# # 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).

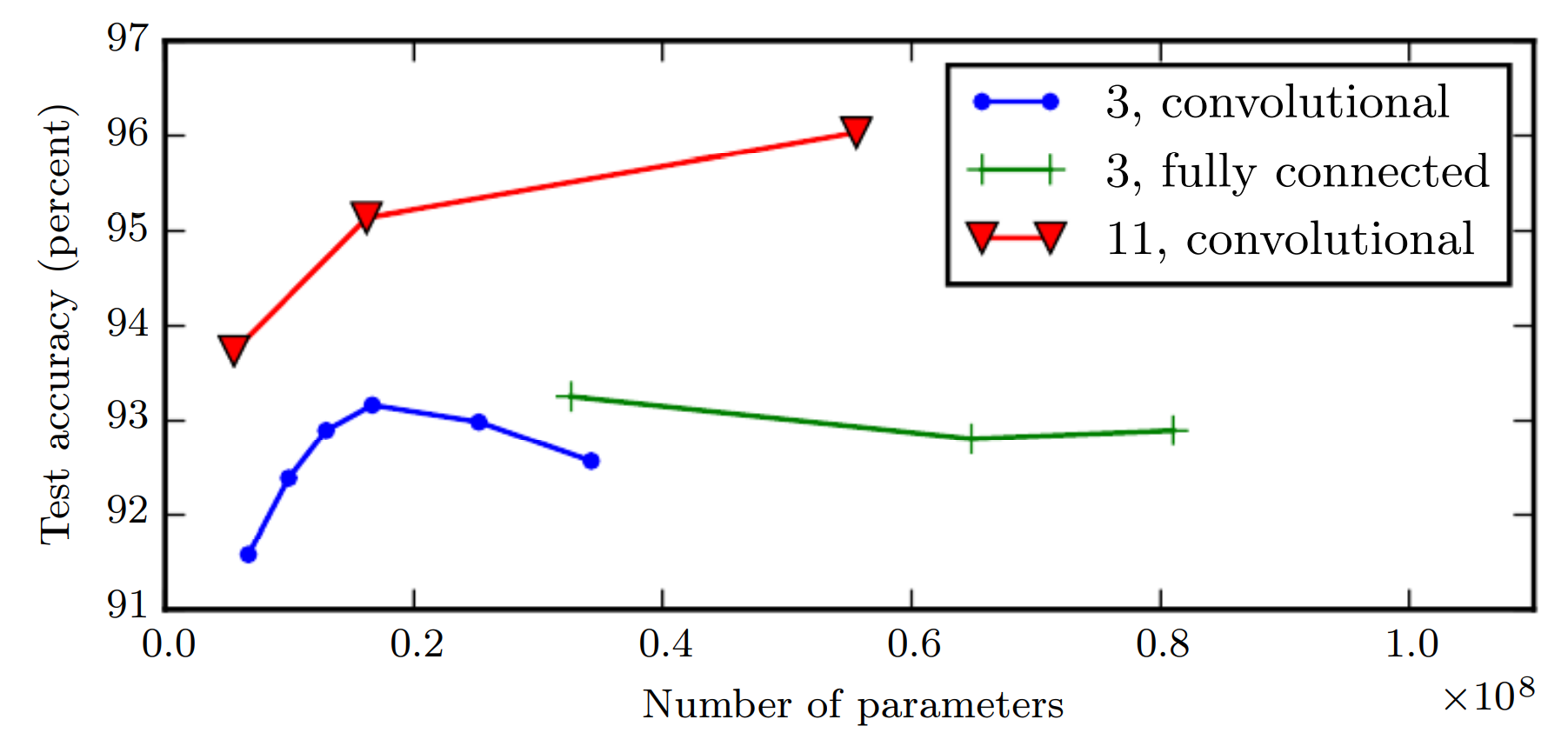
As Goodfellow says in his book “choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions. This can be interpreted from a representation learning point of view as saying that we believe the learning problem consists 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).



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

However, one could argue that the test accuracy in Fig. XX 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. XX 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.



![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).](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.

\*\*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

# # 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.

### ### 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.

$$ C(w) = E(J(z,w)) \qquad \qquad \qquad (3.0) $$.

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

$$ C(w) = E(J(z,w)) = \int J(z,w)dP(z) \qquad \qquad \qquad (3.1) $$

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:

$$ w\_{t+1} = w\_t − \epsilon\_t \nabla\_w J(z, w\_t) \qquad \qquad \qquad (4.0) $$

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 3.1 can be expressed as below.

$$ dP(z) = X N i=1 1 N δ(z − zi) \qquad \qquad \qquad (3.2)$$

$$C(w) = 1 N X N i=1 J(zi , w) \qquad \qquad \qquad (3.3)$$

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

$$ W\_{t+1} = w\_t - \nabla\_w C(w\_t) \qquad \qquad \qquad \qquad $$

$$ = w\_t - t Z \nabla\_w J(z, w\_t) dP(z) \qquad \qquad \qquad \qquad $$

$$ = w\_t - t \sum\_{i=1}^{NxN} \nabla\_w J(z\_i, w\_t) \qquad \qquad \qquad (4.1) $$

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:

$$\overline{J}( \theta ; x,y) = J( \theta ; x,y) + \alpha \Omega( \theta)$$

Where $J( \theta ; x,y) = \left( Xw-y\right) ^{T}\left( Xw-y\right)$, hence we have:

$$ \overline{J}( \theta ; x,y) = \left( Xw-y\right) ^{T}\left( Xw-y\right) +\dfrac{1}{2}\alpha w^{T}w $$

Where

$$ w=\left( X^{T}X+\alpha I\right) ^{-1}X^{T}y $$

The matrix $X^{T}X$ which is proportional to $$\dfrac{1}{m}X^{T}X$$ is replaced by the term $(X^{T}X+\alpha I\right)$.

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