Getting Started with Deep Learning using Keras

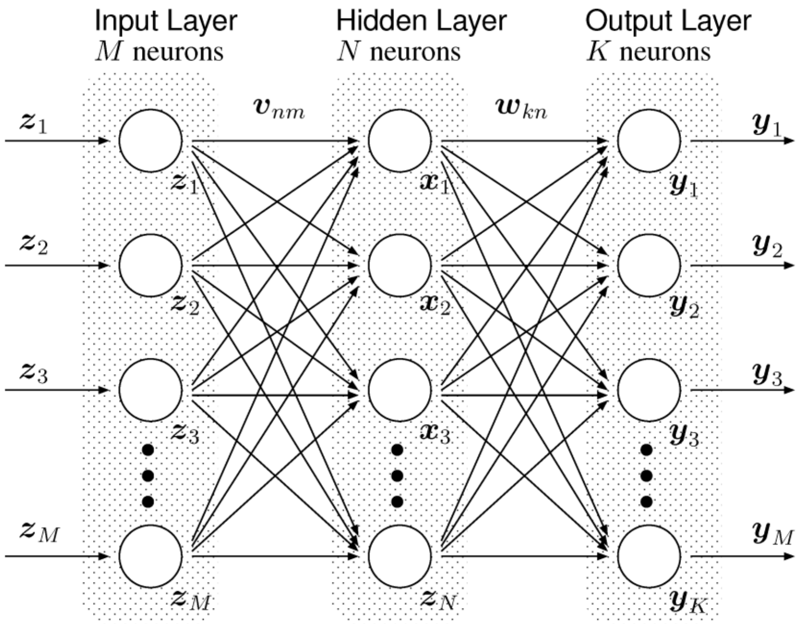
Ogban-Asuquo Ugot

ogbanugot@gmail.com @ogbanugot ogbanugot

Outline

Theory session (Time: 50 minutes):

- Introduction
- What is Deep learning
- Deep Learning Architecture
- How do Neural Networks work?
- Activation functions and output nodes
- How do Neural Networks learn?
- Cost functions
- Backpropagation
- Gradient descent
- Modern practices in deep learning
- Deep learning and hardware requirements
- Limitations of Deep learning
- Summary
- **Q&A session (Time: 10 minutes)**



Outline

Practical session (Time: 40 minutes):

- About Keras
- Data preprocessing
- Building the ANN
- Training the ANN
- Testing the ANN
- Plotting, saving and loading Keras models

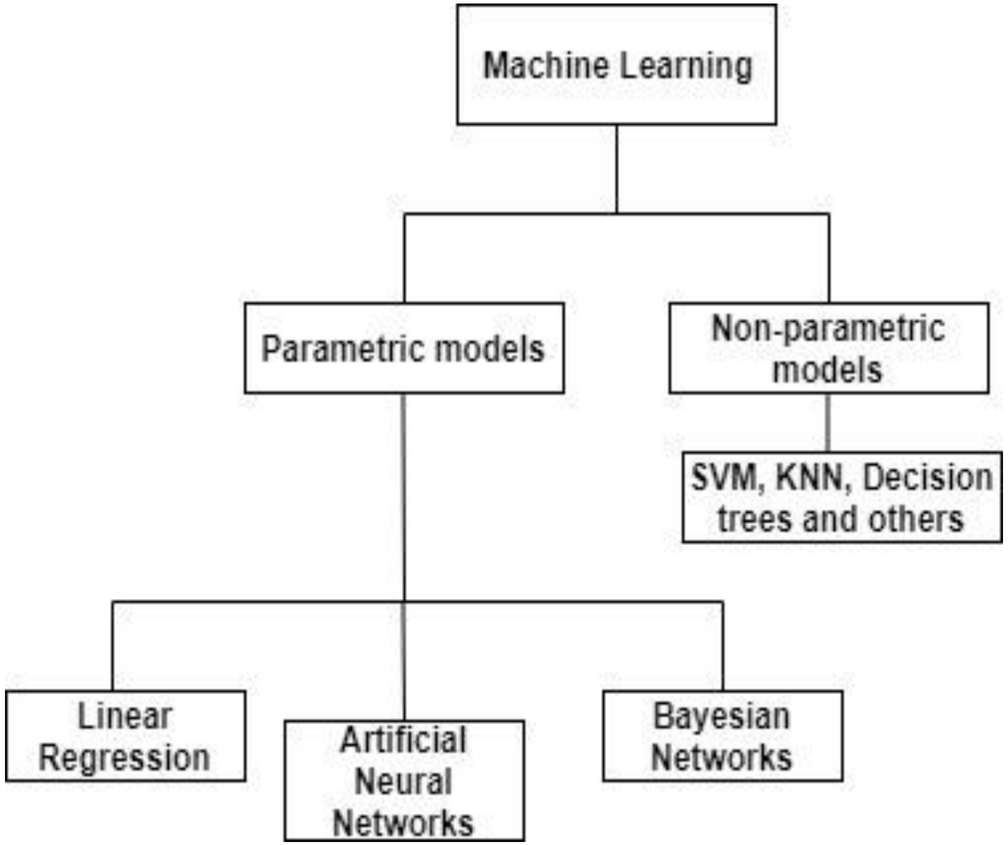
Source Code:

[https://github.com/ogbanugot/Artificial-Neura Network-](https://github.com/ogbanugot/Artificial-Neura)

```python

# Part 2 - Building the ANN

# Importing the Keras libraries and packages

from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

Theory Session

Introduction

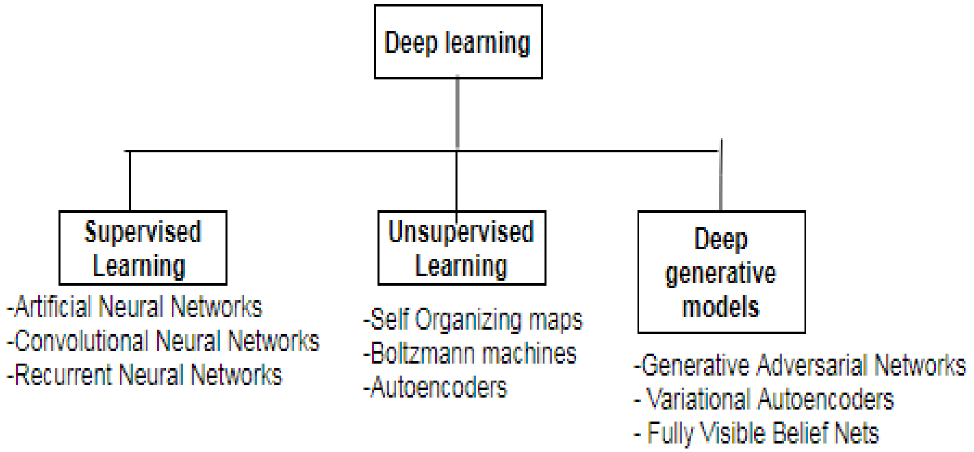


Machine learning taxonomy

- Artificial Intelligence (AI) concerns itself with designing intelligent agents capable of rational decision making in an environment
- Machine learning models learn without being explicitly programmed to do so. The learning model may enhance the AI's performance
- Artificial Neural Networks (ANNs) are a type of machine learning model for learning abstract data representations.

What is Deep Learning?

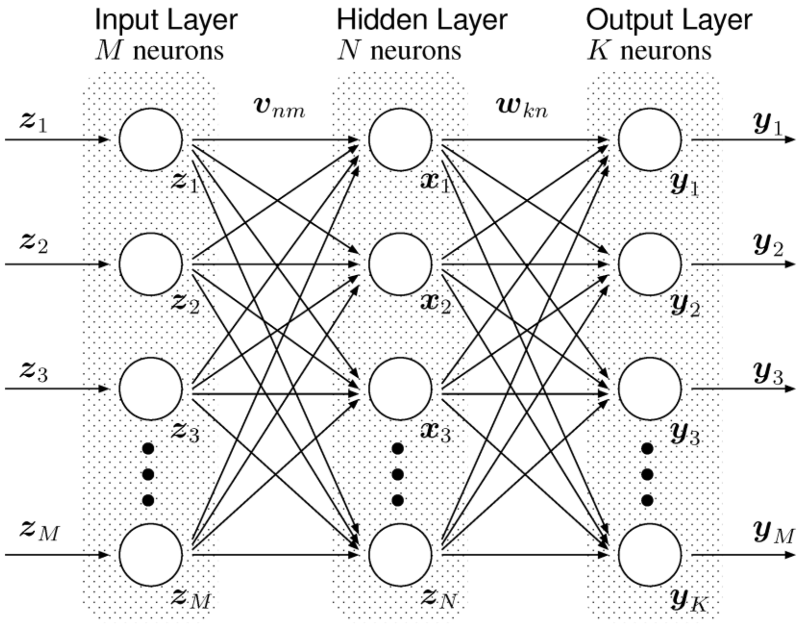
- **Deep learning** is an advanced architecture for artificial neural networks that can learn data representations from larger and more complex datasets.
- Deep learning is built on concepts from the basic Multilayer Perceptron.
- The “deep” in deep learning implies that the multilayer perceptron has multiple hidden layers.
- Deep learning makes use of modern optimization and regularization techniques to achieve improved performance.



Deep learning taxonomy

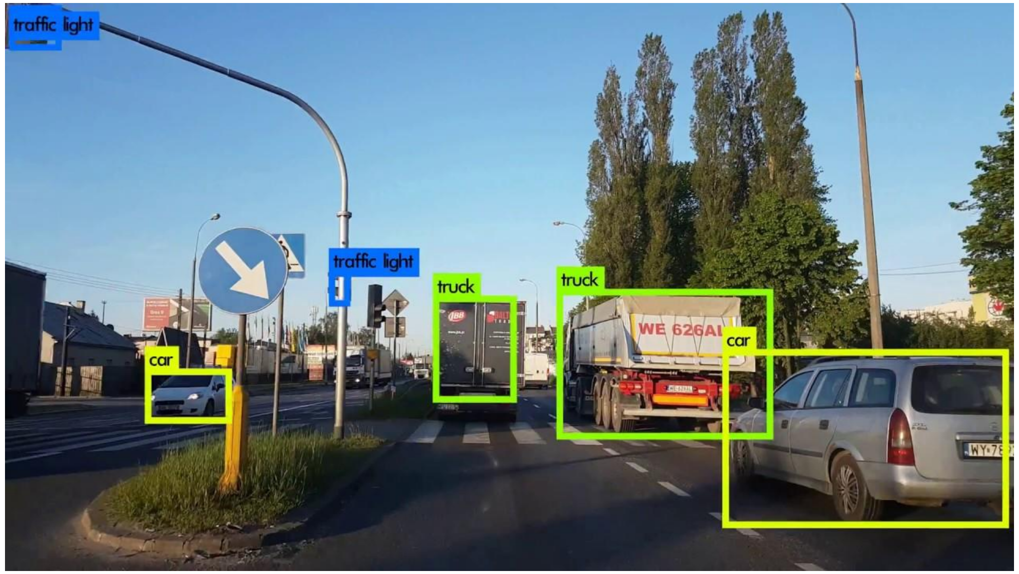
Deep Neural Network Architecture

- Deep neural networks (DNNs) consists of computational nodes (artificial neurons) arranged in layers.
- The layer that takes in input from a dataset is the input layer.
- The layer that gives us our final output is the output layer.
- Hidden layers are everything in between (the input and output layer) and are called “hidden” because their output is not defined in the dataset and therefore must be optimized.



A three layer feed forward network

Applying Deep Neural Nets



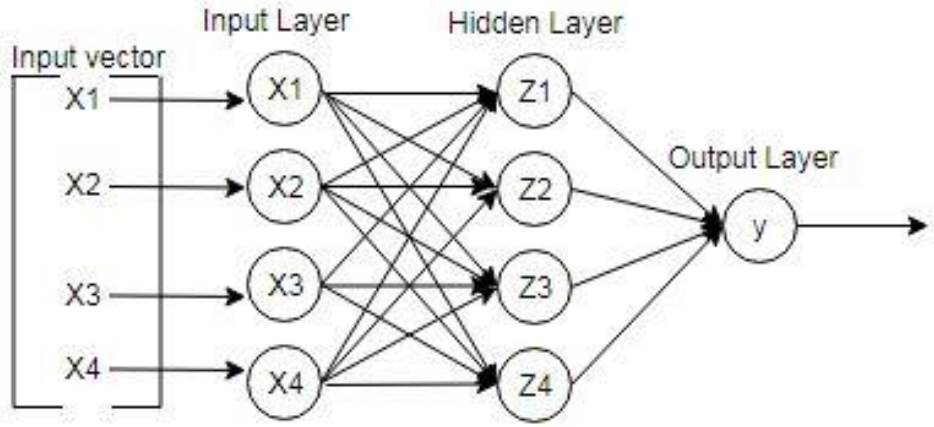
Computer vision object detection tasks

Speech recognition

Expert systems capable of outperforming any human

How do neural networks work?

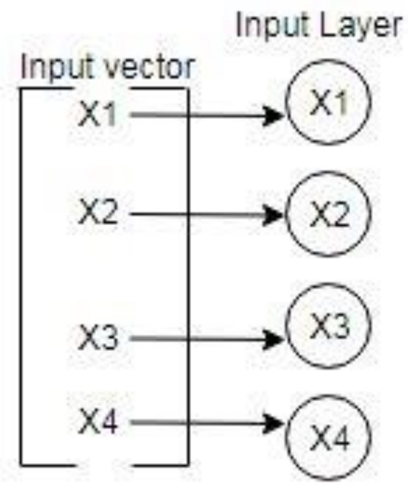
Given the 3 layer neural network;



How do Neural Networks work?

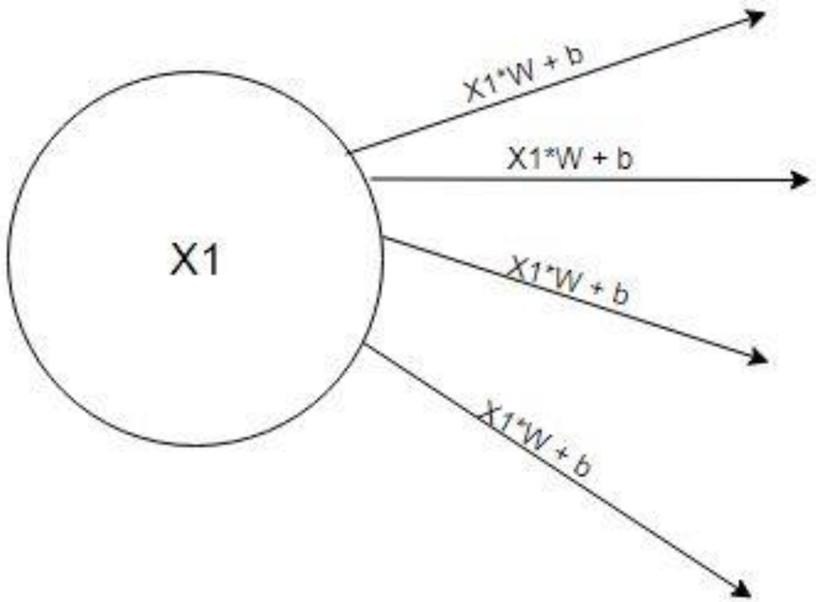
1. Input is fed into the network

An input vector is fed to the input layer, a vector unit to one input node.



How do Neural Networks work?

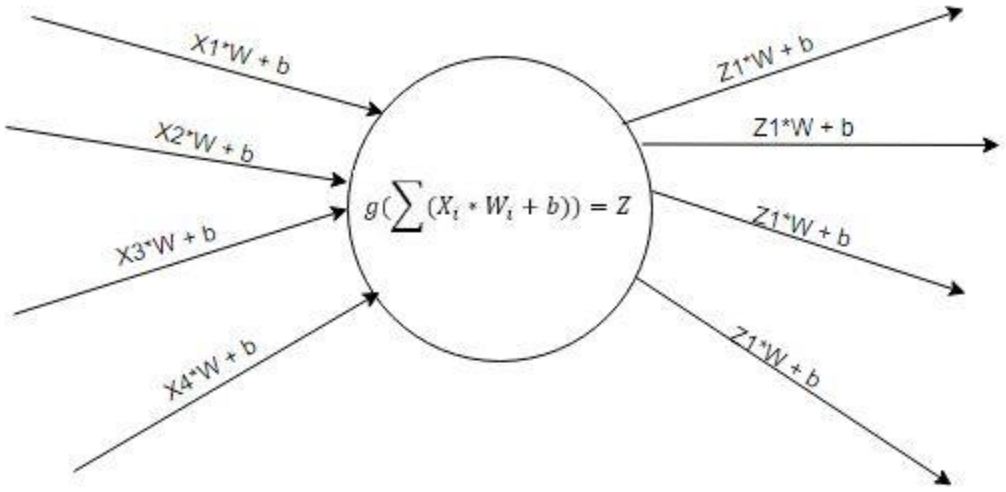
2. Each input node is computed with its weights (W) and bias (b)



$$\boxed{z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}}$$

How do Neural Networks work?

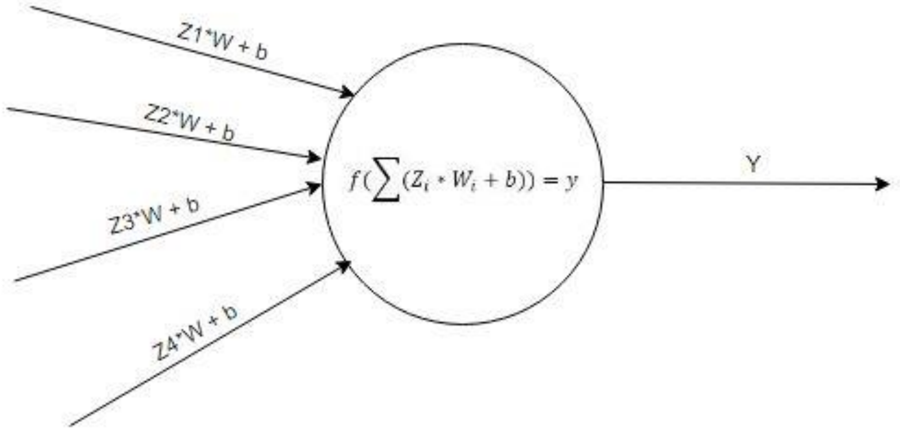
3. The output from an input node is fed to every node of the next layer, the hidden layer.



Each node of the hidden layer performs a summation of all incoming signals, applies an **activation function** to this summation and sends its output to all the nodes of the next layer, which maybe a hidden layer or an output layer.

How do Neural Networks work?

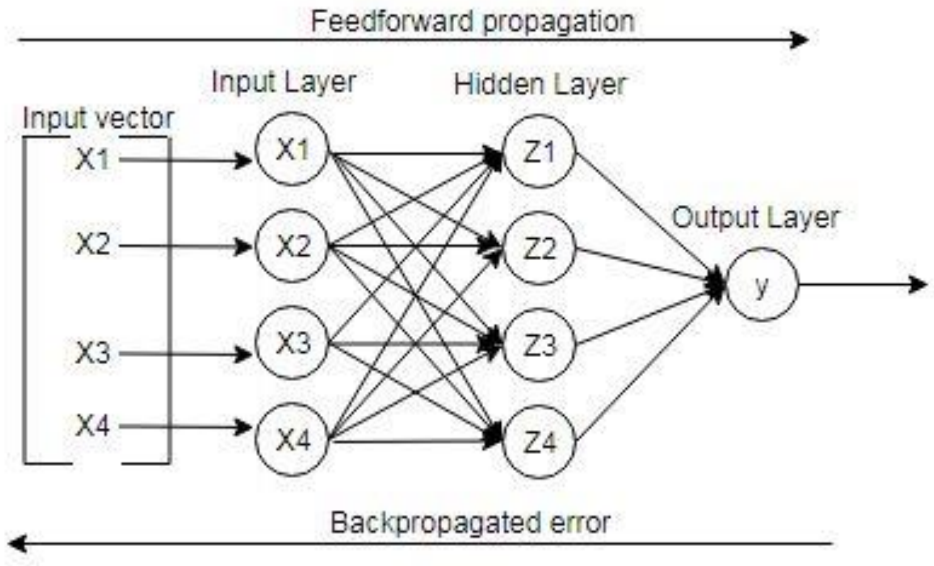
4. If the next layer is an output layer, the output node computes the summation of all incoming signals and transforms this sum with a function to get the output.



Else if the next layer is a hidden layer, step 3 is repeated this time from a hidden layer to another hidden layer, until it gets to the output layer.

How do Neural Networks work?

- The output from a neural network is an estimate of a real value defined in our dataset.
- The process of flowing an input through the network to get an estimated output is know as the feedforward propagation.
- The difference between the estimated output and a real output is known as the error.
- This error is back propagated through the network in a backward direction and is the crucial step in making neural networks learn.
- The backpropagation algorithm describes how to flow the error signal backwards.



Fully Connected Neural Network

Activation functions and output nodes

Activation Functions

- Activation functions transform an incoming signal to a nonlinear value.
- This is more desirable because nonlinear functions are differentiable.

Most used activation functions in the hidden layer are

- Rectified Linear Unit (ReLU) - $\mathrm{g(x)} = \max\{0, x\}$
- Leaky ReLU - $\mathrm{g(x)} \begin{cases} x & if\ x > 0 \\ 0.01x & otherwise \end{cases}$

There are other activation functions

(See Goodfellow et al, 2016 p 289)

Output nodes

- The functions used to define an output node are specific to the general task of a neural network.

In general two task exists;

- Classification and Regression

These tasks make use of different functions in the output node

Classification:

Sigmoid - $f(z) = 1/\ (1 + e^{-z})$

Softmax - $\sigma(z)j = \frac{e^{zj}}{\sum_{k=1}^{k} e^{zk}}\ forj = 1, \ldots, k$

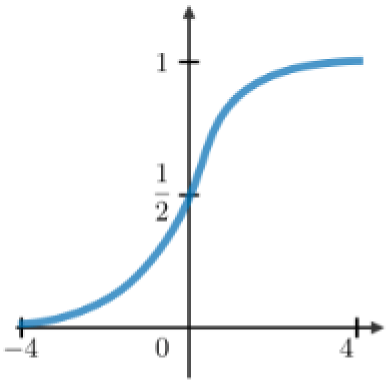
Regression:

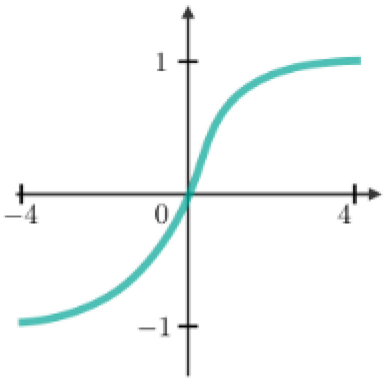
Linear - $f(x) = ax$

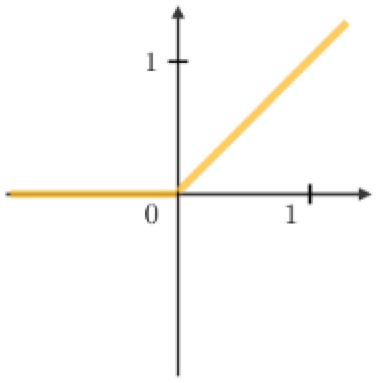
or none

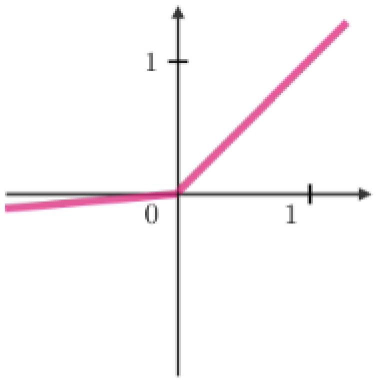
Activation functions

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$









Source: [CS 229 – Machine learning]

How do Neural Networks Learn?

Feedforwarding propagation

- Take input vector
- Calculate the output from each node in each layer.
- Compare the difference between the estimated output of the neural network and the desired output. The difference is known as the error.
- Calculate this error using a cost function.

How do Neural Networks Learn?

Backward propagation

(Backpropagation)

- Back-propagate the calculated error by estimating the gradient of each node in the neural network, from the output layer down to the first hidden layer;
- Update the weights of the neural network so as to minimize the error in the next feedforward processing.

Cost functions

- A cost function is simply the objective function a deep neural network is trying to minimize
- It is a measure of how wrong the model is in terms of its ability to estimate the desired output
- The choice of a cost function is extremely important and we can choose a cost function with respect to the machine learning task.

Classification Tasks:

- Binary cross-entropy function;

$$J(\theta) = -\mathbb{E}_{x \sim p_{data}} log P_{model}(y/x).$$

Regression Tasks:

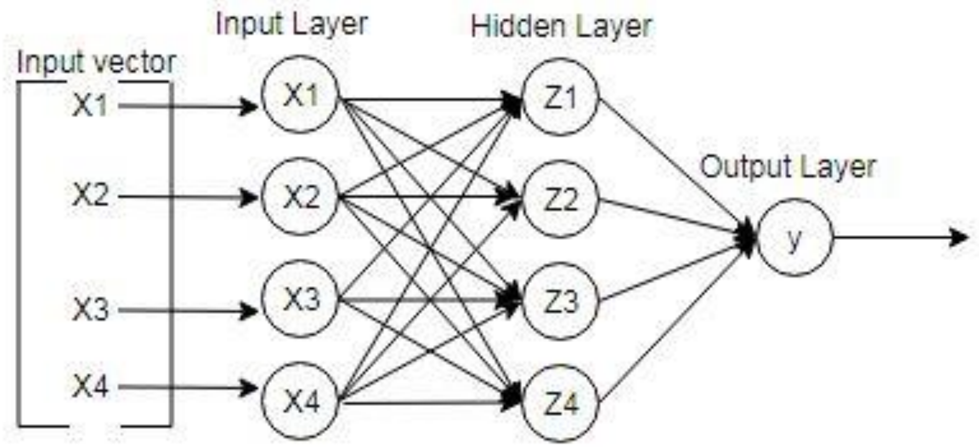
- Mean Squared Error (MSE);

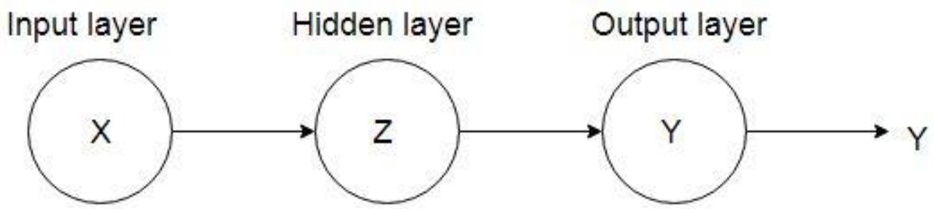
$$mse = \frac{1}{n} \sum_{n=1} (\hat{y} - y)^2$$

- r-squared;
- Mean Log Squared error (MSLE)

Backpropagation

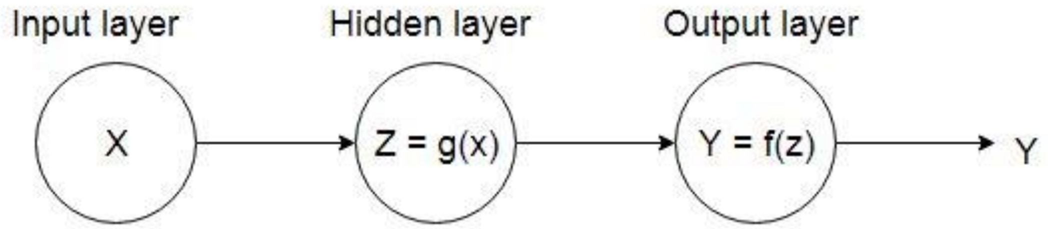
We already know that the backpropagation algorithm is used to train the artificial neural network,
Now lets learn how it works in more detail.





To begin our discussion of backpropagation,
Let us visualize (for simplicity) the neural network on the left as the single path queue on the right.

Backpropagation



Next we define the neural network with its activation function and output node;

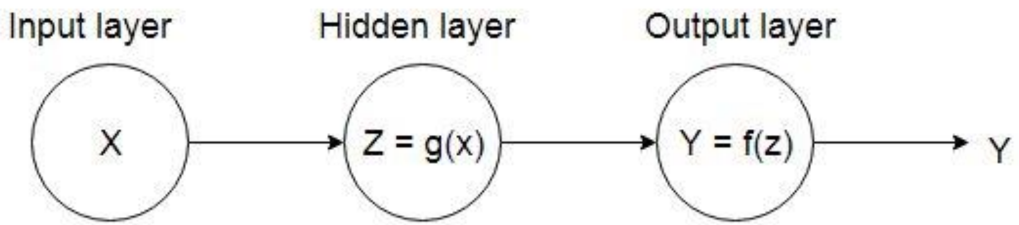
The neural network in the diagram consists of

- x - an input;
- g(x) – the activation function in the hidden layer and
- f(z) the activation function in the output layer.

The output of g(x) is z

The output of f(z) is y (the final output of the neural network)

Backpropagation



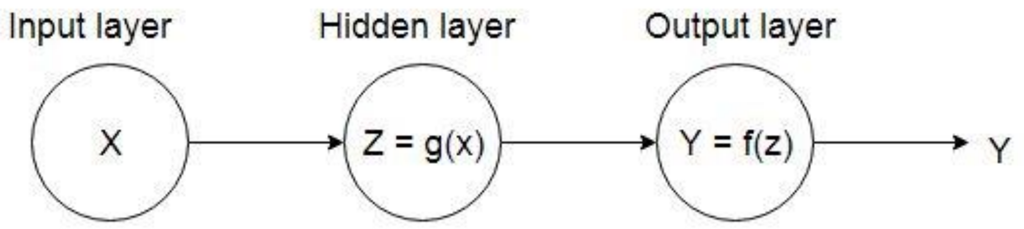
- We can describe the neural network as a link of functions;

$$y = f(g(x))$$

Where y is the estimated output.

- Next, the cost function calculates the error between our estimated output and the desired output. We can call this error $\boldsymbol{e}$.
- Think of $\boldsymbol{e}$ as the output from the cost function.

Backpropagation



- We wont bother calculating the gradient of the error with respect to the output node y because;

$$\frac{\partial \boldsymbol{e}}{\partial y} = \boldsymbol{e}$$

- We go ahead to calculate the gradient of the error $\boldsymbol{e}$ with respect to the output from the hidden node z.

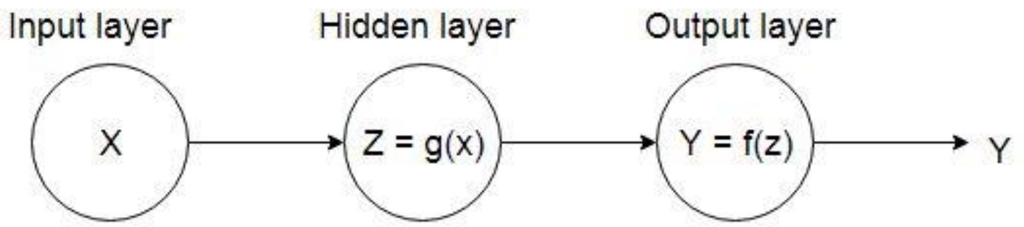
Using the chain rule of calculus, we can express this as;

$$\frac{\partial \boldsymbol{e}}{\partial z} = \frac{\partial \boldsymbol{e}}{\partial y}\frac{\partial y}{\partial z}$$

Backpropagation

$$\frac{\partial \boldsymbol{e}}{\partial z} = \frac{\partial \boldsymbol{e}}{\partial y}\frac{\partial y}{\partial z}$$

Chain rule for the gradient of the error with respect to the hidden node z



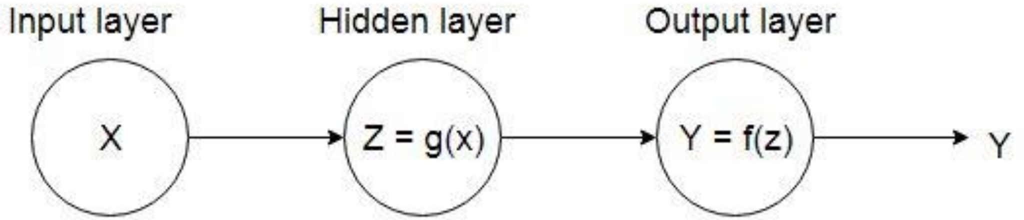
Let's break down the equation

Partial Derivative	Intuitive Understanding
$\frac{\partial \boldsymbol{e}}{\partial z}$	Here, we are asking "What is the partial derivative of the error $\boldsymbol{e}$ with respect to the output node z?"
$\frac{\partial \boldsymbol{e}}{\partial y}$	Here we are taking the partial derivative of the error with respect to the output from the output node y.
$\frac{\partial \boldsymbol{y}}{\partial z}$	Finally, we take the partial derivative of the output from the output node y with respect to the hidden node z.

Backpropagation

$$\frac{\partial \boldsymbol{e}}{\partial z} = \frac{\partial \boldsymbol{e}}{\partial y}\frac{\partial y}{\partial z}$$

Chain rule for the gradient of the error with respect to the hidden node z

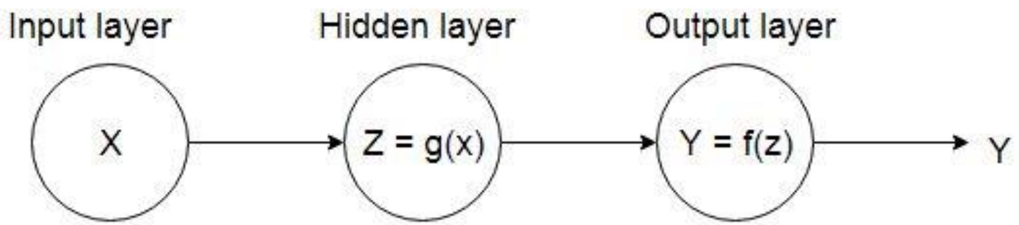


- The simplicity of the chain rule of calculus enables us calculate the gradient of the error $\boldsymbol{e}$ with respect to any parameter in the neural network.
- The backpropagation algorithm recursively calculates the gradient of the error (from the cost function) with respect to all the parameters in the neural network.

Backpropagation

$$\frac{\partial \boldsymbol{e}}{\partial z} = \frac{\partial \boldsymbol{e}}{\partial y}\frac{\partial y}{\partial z}$$

Chain rule for the gradient of the error with respect to the hidden node z



- In order to visualize the actual "backpropagation" of the error;
- consider the gradient of the error with respect to the input x;

$$\frac{\partial \boldsymbol{e}}{\partial x} = \frac{\partial \boldsymbol{e}}{\partial z}\frac{\partial z}{\partial x}$$

- We can see that the gradient $\frac{\partial \boldsymbol{e}}{\partial z}$ which was calculated for the hidden node z is recursively used here to calculate the gradient for the input node x.

Backpropagation

In Summary;

- We estimate an error using a cost function;
- Then, starting from the last hidden layer in the neural network, we find the gradient of the error with respect to each node in that layer and repeat this process for each layer until we work our way backwards to the first hidden layer. This is the backpropagation processing;
- The backpropagation algorithm is actually executed in two phases for each layer in the network. First we calculate the gradients, then we update the weights of the neural network using the gradients. The update of the weights is what enables the actual “learning”.

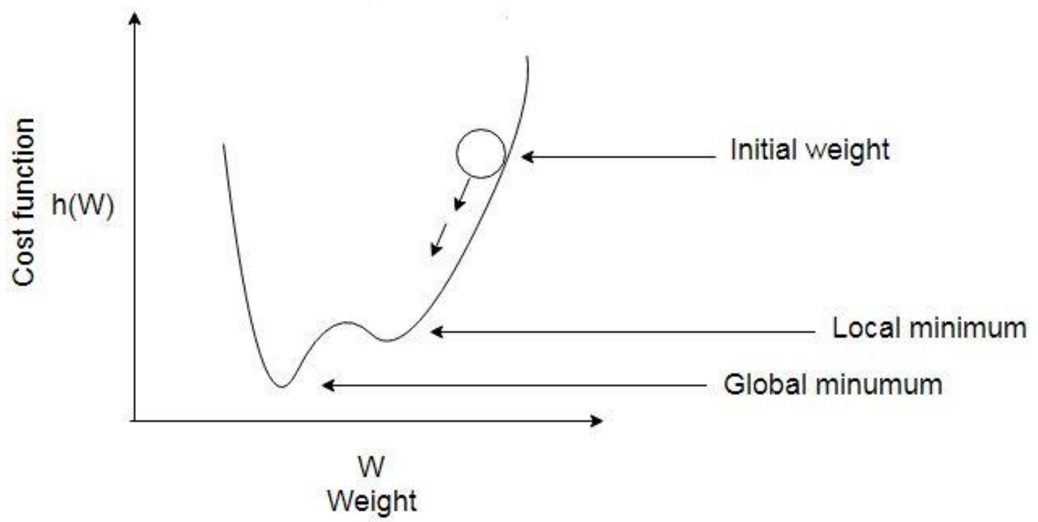
Gradient Descent

What is the motivation for an optimization algorithm for deep learning?

- Simply put, given the parameters of the deep learning model, we cannot precisely determine that we have found the global minimum of the cost function we are trying to minimize.
- One can think of the deep neural network as one big non-linear function. The optimization of a deep learning model is therefore non-convex.
- Therefore, we need an optimization algorithm to help find the global minimum of the cost function.

Enter Gradient Descent....

Gradient Descent



Visualizing the gradient descent algorithm

- Gradient descent is an iterative optimization algorithm for finding the minimum of a cost function.

$$w_i \leftarrow w_i - \alpha \frac{\partial h(w_i)}{\partial w_i}$$

- In the equation above, the cost function $h(w_i)$ calculates the error $\boldsymbol{e}$;
- The gradient of this error with respect to the weights $\frac{\partial h(w_i)}{\partial w_i}$ multiplied by the learning rate α, is subtracted from the current weight w_i.
- This procedure ensures that we optimize the parameter (in this case the weights) that minimizes the error e.

Gradient Descent

- Modern implementations of gradient descent make use of extensions of the gradient descent algorithm grouped under the umbrella term Stochastic Gradient Descent (SGD).
- *Stochastic* because samples from the dataset are selected randomly as opposed to the standard gradient descent algorithm where samples are selected in-order.

This stochastic sampling has shown to improve the optimization procedure and thus find a much preferable minimum for the cost function. Variants of the SGD include;

- Adaptive Moment Estimation (ADAM)
- Adaptive Gradient (AdaGrad)
- Root Mean Squared Propagation (RMSProp)

How do Neural Networks learn? *Revisited*

Feedforwarding propagation

- Take input vector
- Calculate the output from each node in each layer.
- Compare the difference between the estimated output of the neural network and the desired output. The difference is known as the error.
- Calculate this error using a cost function.

Backward propagation (Backpropagation)

- Back-propagate the calculated error by estimating the gradient of each node in the neural network, from the output layer down to the first hidden layer;
- Using an optimization algorithm, update the weights of the neural network so as to minimize the error in the next feedforward processing.

Modern practices in deep learning

Let us briefly define some important practices essential to the success of modern implementations of large scale deep learning models;

- **Regularization**: any modification we make to the learning algorithm that is intended to reduce the generalization error, but not its training error (Goodfellow et al 2016 p 228).
- Example of a regularization technique is ***dropout***, where we randomly remove some nodes in the network along with all of their incoming and outgoing connections. Helps, with reducing overfitting.
- **Batch normalization**: To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

Deep learning and hardware requirements

- Training large scale deep learning models takes time, and therefore requires expensive hardware resources;

Parallel processing with Graphical Processing Units enables training of very large DNNs.
However, buying GPU hardware is expensive.

Google Cloud

aws

Microsoft Azure

Cloud machine learning services provide affordable compute power

Limitations of Deep learning

- Deep neural networks map an input space to an output space and are therefore sensitive to changes in the input data
- Adversarial systems can take advantage of modifications in the input data to fool deep neural networks
- The high reliance on data, annotated or otherwise restricts the deep neural networks ability to generalize
- Deep neural networks lack any abstract "thinking" or "reasoning" models and perform poorly on tasks that require reasoning and strategic planning.

The boy is holding a baseball bat.

An example of an image caption error
Image Source: [Keras blog]

Theory Session Recap

- The strength of Deep learning models is that they are capable of learning extremely complex and abstract data representation.
- The architecture of a deep learning model concerns the number of layers in the model, the number of computational units (nodes), activation functions and cost functions.
- The backpropagation algorithm is the primary training algorithm for deep neural networks.
- Stochastic gradient descent is the primary optimization algorithm used to update the parameters of deep neural network
- The success of modern deep learning implementations can be attributed to the availability of large datasets, powerful computing hardware, modern optimization algorithms and hyperparameter tuning techniques.
- Deep learning models have achieved the state-of-the-art in image classification, speech recognition and mobile robotics.
- Deep learning has its limitations which include, weak generalization and poor performance in reasoning and planning tasks.

Further reading

- Russell, S.J. and Norvig, P., 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.

- Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. *Deep learning* (Vol. 1). Cambridge: MIT press.

Practical Session

Outline

Practical session (Time: 50 minutes):

- About Keras
- Data preprocessing
- Building the ANN
- Training the ANN
- Testing the ANN
- Plotting, saving and loading Keras models

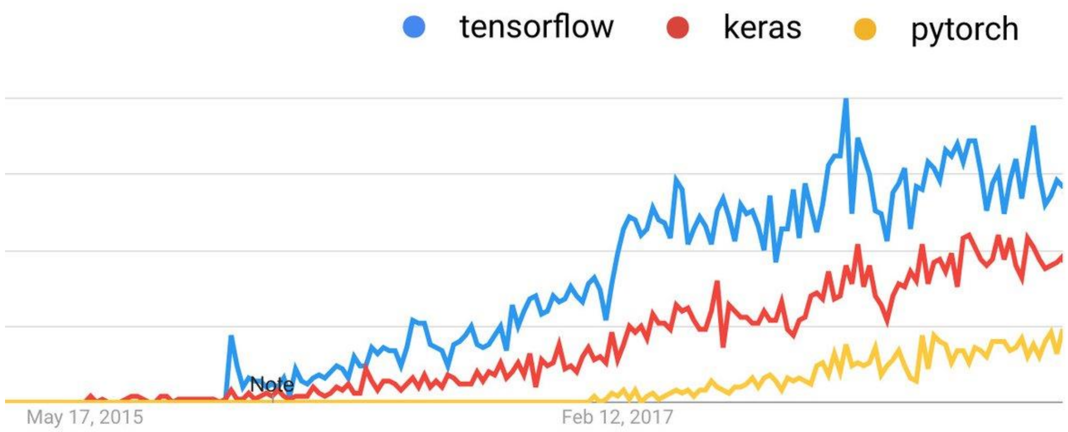
Source Code:

[https://github.com/ogbanugot/Artificial-Neura Network-](https://github.com/ogbanugot/Artificial-Neura)

```python
# Part 2 - Building the ANN

# Importing the Keras libraries and packages

from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

About Keras

- Keras is an open source deep learning library
- Keras initial release was in 2015 and its original author is François Chollet.
- Keras is a high level API for deep learning libraries such as Tensorflow and Theano
- Keras is easy to use and very well documented.
- It abstracts a lot of the complexity of working with lower level libraries without trading functionality and performance.
- Today, there is actually no “keras or Tensorflow” option. As at Tensorflow 1.9, Keras has been adopted as the official API for tensorflow with the inclusion of the tf.keras module in the tensorflow build.



Looking at Keras popularity over the years

Data Preprocessing

Churn Modelling Dataset

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
2	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
3	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
4	3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
5	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
6	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1	0
7	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
8	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
9	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
10	9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.5	0
11	10	15592389	H?	684	France	Male	27	2	134603.88	1	1	1	71725.73	0
12	11	15767821	Bearce	528	France	Male	31	6	102016.72	2	0	0	80181.12	0
13	12	15737173	Andrews	497	Spain	Male	24	3	0	2	1	0	76390.01	0
14	13	15632264	Kay	476	France	Female	34	10	0	2	1	0	26260.98	0
15	14	15691483	Chin	549	France	Female	25	5	0	2	0	0	190857.79	0
16	15	15600882	Scott	635	Spain	Female	35	7	0	2	1	1	65951.65	0
17	16	15643966	Goforth	616	Germany	Male	45	3	143129.41	2	0	1	64327.26	0
18	17	15737452	Romeo	653	Germany	Male	58	1	132602.88	1	1	0	5097.67	1
19	18	15788218	Henderson	549	Spain	Female	24	9	0	2	1	1	14406.41	0
20	19	15661507	Muldrow	587	Spain	Male	45	6	0	1	0	0	158684.81	0
21	20	15568982	Hao	726	France	Female	24	6	0	2	1	1	54724.03	0
22	21	15577657	McDonald	732	France	Male	41	8	0	2	1	1	170886.17	0
23	22	15597945	Dellucci	636	Spain	Female	32	8	0	2	1	0	138555.46	0
24	23	15699309	Gerasimov	510	Spain	Female	38	4	0	1	1	0	118913.53	1
25	24	15725737	Mosman	669	France	Male	46	3	0	2	0	1	8487.75	0
26	25	15625047	Yen	846	France	Female	38	5	0	1	1	1	187616.16	0
27	26	15738191	Maclean	577	France	Male	25	3	0	2	0	1	124508.29	0
28	27	15736816	Young	756	Germany	Male	36	2	136815.64	1	1	1	170041.95	0
29	28	15700772	Nebechi	571	France	Male	44	9	0	2	0	0	38433.35	0
30	29	15728693	McWilliams	574	Germany	Female	43	3	141349.43	1	1	1	100187.43	0

Given the dataset, the Machine learning task is to be able to predict which customers are likely to exit the bank or stay. This is a classification task!

Data Preprocessing

#Import the dataset

7: Import Pandas library

10:Load the dataset from disk.

11: Slice the dataset (and get the features we need) from index 4 to index 13.

12: Slice the dataset and get the target label (the dependent variable) at index 14.

```
2
3  # Part 1 - Data Preprocessing
4
5  # Importing the libraries
6
7  import pandas as pd
8
9  # Importing the dataset
10 dataset = pd.read_csv('Churn_Modelling.csv')
11 X = dataset.iloc[:, 3:13].values
12 y = dataset.iloc[:, 13].values
```

Data Preprocessing
Independent variables for training

	A	B	C	D	E	F	G	H	I	J
1	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
2	619	France	Female	42	2	0	1	1	1	101348.88
3	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
4	502	France	Female	42	8	159660.8	3	1	0	113931.57
5	699	France	Female	39	1	0	2	0	0	93826.63
6	850	Spain	Female	43	2	125510.82	1	1	1	79084.1
7	645	Spain	Male	44	8	113755.78	2	1	0	149756.71
8	822	France	Male	50	7	0	2	1	1	10062.8
9	376	Germany	Female	29	4	115046.74	4	1	0	119346.88
10	501	France	Male	44	4	142051.07	2	0	1	74940.5
11	684	France	Male	27	2	134603.88	1	1	1	71725.73
12	528	France	Male	31	6	102016.72	2	0	0	80181.12
13	497	Spain	Male	24	3	0	2	1	0	76390.01
14	476	France	Female	34	10	0	2	1	0	26260.98
15	549	France	Female	25	5	0	2	0	0	190857.79
16	635	Spain	Female	35	7	0	2	1	1	65951.65
17	616	Germany	Male	45	3	143129.41	2	0	1	64327.26
18	653	Germany	Male	58	1	132602.88	1	1	0	5097.67
19	549	Spain	Female	24	9	0	2	1	1	14406.41
20	587	Spain	Male	45	6	0	1	0	0	158684.81
21	726	France	Female	24	6	0	2	1	1	54724.03
22	732	France	Male	41	8	0	2	1	1	170886.17
23	636	Spain	Female	32	8	0	2	1	0	138555.46
24	510	Spain	Female	38	4	0	1	1	0	118913.53
25	669	France	Male	46	3	0	2	0	1	8487.75
26	846	France	Female	38	5	0	1	1	1	187616.16
27	577	France	Male	25	3	0	2	0	1	124508.29
28	756	Germany	Male	36	2	136815.64	1	1	1	170041.95
29	571	France	Male	44	9	0	2	0	0	38433.35
30	574	Germany	Female	43	3	141349.43	1	1	1	100187.43

Now we have sliced out our relevant independent variables needed for training, from the original dataset

Data Preprocessing

Identifying and handling labels and categories

- The Geography and Gender features are text-values so we need to convert them to numerical values. This is called label encoding.
- These features are also categorical, that is, the classify a feature into a category e.g. Spain, France, Male, female etc. We'll need to one-hot encode them. Why do we need to one-hot encode?

	A	B	C	D	E	F	G	H	I	J
1	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
2	619	France	Female	42	2	0	1	1	1	101348.88
3	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
4	502	France	Female	42	8	159660.8	3	1	0	113931.57
5	699	France	Female	39	1	0	2	0	0	93826.63
6	850	Spain	Female	43	2	125510.82	1	1	1	79084.1
7	645	Spain	Male	44	8	113755.78	2	1	0	149756.71
8	822	France	Male	50	7	0	2	1	1	10062.8
9	376	Germany	Female	29	4	115046.74	4	1	0	119346.88
10	501	France	Male	44	4	142051.07	2	0	1	74940.5
11	684	France	Male	27	2	134603.88	1	1	1	71725.73
12	528	France	Male	31	6	102016.72	2	0	0	80181.12
13	497	Spain	Male	24	3	0	2	1	0	76390.01
14	476	France	Female	34	10	0	2	1	0	26260.98
15	549	France	Female	25	5	0	2	0	0	190857.79
16	635	Spain	Female	35	7	0	2	1	1	65951.65
17	616	Germany	Male	45	3	143129.41	2	0	1	64327.26
18	653	Germany	Male	58	1	132602.88	1	1	0	5097.67
19	549	Spain	Female	24	9	0	2	1	1	14406.41
20	587	Spain	Male	45	6	0	1	0	0	158684.81
21	726	France	Female	24	6	0	2	1	1	54724.03
22	732	France	Male	41	8	0	2	1	1	170886.17
23	636	Spain	Female	32	8	0	2	1	0	138555.46
24	510	Spain	Female	38	4	0	1	1	0	118913.53
25	669	France	Male	46	3	0	2	0	1	8487.75
26	846	France	Female	38	5	0	1	1	1	187616.16
27	577	France	Male	25	3	0	2	0	1	124508.29
28	756	Germany	Male	36	2	136815.64	1	1	1	170041.95
29	571	France	Male	44	9	0	2	0	0	38433.35
30	574	Germany	Female	43	3	141349.43	1	1	1	100187.43

The Geography and Gender fields are text-values and are also categorical

Data Preprocessing

#Encoding categorical data

15: Import the necessary library and its classes

16: Create the LabelEncoder object for the first text feature.

17: Label encode the feature using the fit_transform class and overwrite the old values in the dataset.

18 & 19: Repeats line 16 and 17 for the second text feature in the dataset.

```
14  # Encoding categorical data
15  from sklearn.preprocessing import LabelEncoder, OneHotEncoder
16  labelencoder_X_1 = LabelEncoder()
17  X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
18  labelencoder_X_2 = LabelEncoder()
19  X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
20  onehotencoder = OneHotEncoder(categorical_features = [1])
21  X = onehotencoder.fit_transform(X).toarray()
22  X = X[:, 1:]
23
```

Data Preprocessing

Why do we need to one-hot encode?

- Label encoding a text-valued categorical feature assigns a numerical value to each category.
- If our categories are not **ordinal** i.e. there is no relational order in which the categories appear. Then we should one-hot encode.
- One-hot encoding prevents the learning algorithm from placing importance on the manner in which the categorical values are listed.
- When we one-hot encode, we take the categorical values and represent each one in its own column as a dummy variable.

	A	B	C	D	E	F	G	H	I	J
1	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
2	619	France	Female	42	2	0	1	1	1	101348.88
3	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
4	502	France	Female	42	8	159660.8	3	1	0	113931.57
5	699	France	Female	39	1	0	2	0	0	93826.63
6	850	Spain	Female	43	2	125510.82	1	1	1	79084.1
7	645	Spain	Male	44	8	113755.78	2	1	0	149756.71
8	822	France	Male	50	7	0	2	1	1	10062.8
9	376	Germany	Female	29	4	115046.74	4	1	0	119346.88
10	501	France	Male	44	4	142051.07	2	0	1	74940.5
11	684	France	Male	27	2	134603.88	1	1	1	71725.73
12	528	France	Male	31	6	102016.72	2	0	0	80181.12
13	497	Spain	Male	24	3	0	2	1	0	76390.01
14	476	France	Female	34	10	0	2	1	0	26260.98
15	549	France	Female	25	5	0	2	0	0	190857.79
16	635	Spain	Female	35	7	0	2	1	1	65951.65
17	616	Germany	Male	45	3	143129.41	2	0	1	64327.26
18	653	Germany	Male	58	1	132602.88	1	1	0	5097.67
19	549	Spain	Female	24	9	0	2	1	1	14406.41
20	587	Spain	Male	45	6	0	1	0	0	158684.81
21	726	France	Female	24	6	0	2	1	1	54724.03
22	732	France	Male	41	8	0	2	1	1	170886.17
23	636	Spain	Female	32	8	0	2	1	0	138555.46
24	510	Spain	Female	38	4	0	1	1	0	118913.53
25	669	France	Male	46	3	0	2	0	1	8487.75
26	846	France	Female	38	5	0	1	1	1	187616.16
27	577	France	Male	25	3	0	2	0	1	124508.29
28	756	Germany	Male	36	2	136815.64	1	1	1	170041.95
29	571	France	Male	44	9	0	2	0	0	38433.35
30	574	Germany	Female	43	3	141349.43	1	1	1	100187.43

```
Out[3]:
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)
```

Label encoded array. Notice that the Geography and Gender fields are now numbers

Data Preprocessing

#Encoding categorical data

20: Create the onehotencoder object, the parameter “categorical_ features” specifies the index of the categorical feature in the dataset.

21: We onehotencode the ‘Geography’ feature at index 1.

22: We drop one of the newly created dummy variables at index 1. The final dataset has 11 features.

```
14  # Encoding categorical data
15  from sklearn.preprocessing import LabelEncoder, OneHotEncoder
16  labelencoder_X_1 = LabelEncoder()
17  X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
18  labelencoder_X_2 = LabelEncoder()
19  X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
20  onehotencoder = OneHotEncoder(categorical_features = [1])
21  X = onehotencoder.fit_transform(X).toarray()
22  X = X[:, 1:]
23
```

Data Preprocessing

The Dummy variable trap

- The dummy variable trap is a scenario where two or more independent variables are multicollinear.
- This means that if we know the value of one, we can estimate the value of the other quite accurately.
- After executing line 21 in the source code, our dataset now contains three new dummy variables.
- Notice that each column represents a category, in this case a country.
- If we know the value of the first two columns, we can accurately guess the value of the third. The 3 dummy variables are multicollinear.
- This is why we drop one dummy variable in line 22 to avoid the dummy variable trap. It is for this same reason we do not onehotencode the “Gender” feature.

	0	1	2	3	4	5	6	7	8	9
0	1	0	0	619	0	42	2	0	1	1
1	0	0	1	608	0	41	1	83807.9	1	0
2	1	0	0	502	0	42	8	159661	3	1
3	1	0	0	699	0	39	1	0	2	0
4	0	0	1	850	0	43	2	125511	1	1
5	0	0	1	645	1	44	8	113756	2	1
6	1	0	0	822	1	50	7	0	2	1
7	0	1	0	376	0	29	4	115047	4	1
8	1	0	0	501	1	44	4	142051	2	0
9	1	0	0	684	1	27	2	134604	1	1
10	1	0	0	528	1	31	6	102017	2	0
11	0	0	1	497	1	24	3	0	2	1
12	1	0	0	476	0	34	10	0	2	1
13	1	0	0	549	0	25	5	0	2	0
14	0	0	1	635	0	35	7	0	2	1
15	0	1	0	616	1	45	3	143129	2	0
16	0	1	0	653	1	58	1	132603	1	1
17	0	0	1	549	0	24	9	0	2	1
18	0	0	1	587	1	45	6	0	1	0
19	1	0	0	726	0	24	6	0	2	1
20	1	0	0	732	1	41	8	0	2	1

Three Dummy variables have been added. This leads to the dummy variable trap

Data Preprocessing
Cross validation

Cross validation is the practice of separating the training set from the test set, so that we can evaluate more accurately, the performance of a machine learning model.

#Split the dataset into training and test set

25: Import the necessary class

26: The train_test_split class takes as parameters;

- an array of the training features (X)
- an array of the target variable (y)
- test_size specifies the ratio of the traning data to the test data. In this case 0.2 means 20% of the overall dataset is reserved as the test data, and of course 80% is used as the training data.

```
23
24  # Splitting the dataset into the Training set and Test set
25  from sklearn.model_selection import train_test_split
26  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
27
```

Data Preprocessing
Feature scaling

Feature scaling ensures that the training and test set values are scaled to a uniform range so that the learning algorithm does not place a higher precedence on larger numerical values.

#Feature Scaling

29: Import the StandardScaler

30: Create object of the StandardScaler class

31: Fit and scale the training data using 'fit_transform'

32: Scale the test data using the same object instance and the 'transform' method so that both training and test set are scaled uniformly.

```
28  # Feature Scaling
29  from sklearn.preprocessing import StandardScaler
30  sc = StandardScaler()
31  X_train = sc.fit_transform(X_train)
32  X_test = sc.transform(X_test)
33
```

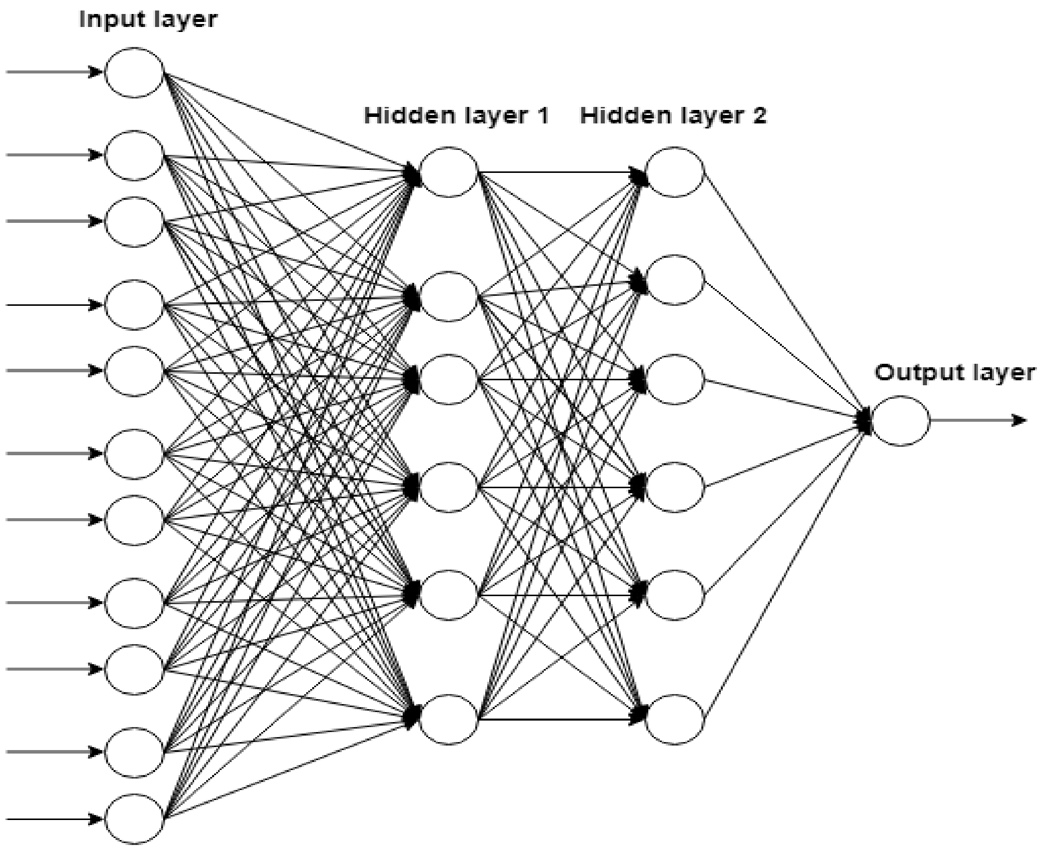
Data preprocessing
(Recap)

Data preprocessing prepares our dataset before we feed it to the learning algorithm. In general the following are key data preprocessing practices;

- Feature engineering
- Label encoding and One-hot-encoding
- Feature scaling (Standardization or Normalization)
- Cross validation (Separating the training and test set)

Other data preprocessing techniques exist for Computer vision and Natural language processing (NLP) machine learning problems.

Building the ANN



We'll be building a four layer deep artificial neural network

Building the ANN

#Importing the Keras libraries and packages

38: Import the Sequential API from Keras.models. This enables us build the neural network graph by adding the layers (and its properties) sequentially, one of after the other.

39: Import the Dense class, this will enable us build fully connected layers where all the nodes are linked to each other.

[Keras Documentation]

```
33
34  # Part 2 - Building the ANN
35
36  # Importing the Keras libraries and packages
37
38  from keras.models import Sequential
39  from keras.layers import Dense
40
```

Building the ANN

#Adding the input layer and first hidden layer

42: Initialize the Sequential class by creating an object instance. We name it classifier because we are building a classifier.

45: You can use the add() method to each layer. Then we use the Dense class to structure the layer. The parameters are;

- Units: the number of nodes in the layer
- Kernel_initializer: Set the initial random weights of the Keras layer
- Activation: the activation function for the nodes
- Input_dim: The number of input nodes to the layer (Only defined for the first hidden layer).

```
41  # Initialising the ANN
42  classifier = Sequential()
43
44  # Adding the input layer and the first hidden layer
45  classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
46
```

Building the ANN

#Adding the second hidden layer

48: Use the add() method again, to add the second hidden layer. The parameters used are the same as the first hidden layer except for the ‘input_dim’ parameter.

#Adding the output layer

45: Finally we add the output layer. The output layer has only one node, and we use the sigmoid function for the classification task.

```
47  # Adding the second hidden layer
48  classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
49
50  # Adding the output layer
51  classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
52
```

Building the ANN

#Compiling the ANN

Use the compile() method to configure the network for training.

48: The parameters for the compile() method used are;

- optimizer: an instance of the gradient descent optimizer
- loss: The cost function or objective function for the network
- metric: List of metrics to be evaluated by model during the training and testing.

```

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

Training the ANN

#Training the ANN

When we train a machine learning model, we ‘fit’ the model to the training data.

Use the fit() method to train the model

48: The parameters for the fit() method used are;

- X: Numpy array of training data
- Y: Numpy array of target data
- batch_size: Number of samples per gradient update
- epochs: an epoch is the number of iterations over the entire X and Y data. This parameter specifies the number of epochs to train the model.

```
# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 50)
```

Training the ANN

```
Epoch 1/50
8000/8000 [==============================] - 6s 691us/step - loss:
0.4771 - acc: 0.7960
Epoch 2/50
8000/8000 [==============================] - 5s 620us/step - loss:
0.4267 - acc: 0.7960
Epoch 3/50
8000/8000 [==============================] - 5s 632us/step - loss:
0.4213 - acc: 0.8066
Epoch 4/50
8000/8000 [==============================] - 5s 590us/step - loss:
0.4173 - acc: 0.8239
```

Screenshot of the training procedure showing the epoch, loss and accuracy

Training results

```
Epoch 48/50
8000/8000 [==============================] - 9s 1ms/step - loss: 0.4007
- acc: 0.8359
Epoch 49/50
8000/8000 [==============================] - 9s 1ms/step - loss: 0.4004
- acc: 0.8359
Epoch 50/50
8000/8000 [==============================] - 9s 1ms/step - loss: 0.4001
- acc: 0.8346
Out[1]: <keras.callbacks.History at 0x7fe13d8f6b70>
```

Training Loss: 0.4001; Training Accuracy: 0.8346

Testing the ANN

#Testing the model

In order to validate the performance of the model, we evaluate its performance on the test set. Use the predict() method to test the model on the test set (X_test)

62: The parameters for the predict() method is the test dataset stored in X_test.

63: The predicted values stored in y_pred range from 0 to 1. This is due to the fact that we made use of the sigmoid function in the output node. In line 63 we are simply saying, store any value greater than 0.5 as 1 and values less than 0.5 as 0.

So a 1 represents a positive that the customer is likely to exit;

And a 0 represents a negative that the customer is likely to exit.

```
58
59  # Part 3 - Testing the model - Making predictions and evaluating the model
60
61  # Predicting the Test set results
62  y_pred = classifier.predict(X_test)
63  y_pred = (y_pred > 0.5)
64
```

Testing the ANN
Confusion matrix

#Making the confusion matrix

The confusion matrix provides a means of evaluating the accuracy of a binary classification (See [sklearn documentation](#))

66: Import the confusion_matrix class.

63: The parameters for confusion_matrix are the ground truth (correct values, y_test) and the predicted values (y_pred).

```
65    # Making the Confusion Matrix
66    from sklearn.metrics import confusion_matrix
67    cm = confusion_matrix(y_test, y_pred)
```

Testing the ANN
...more on the confusion matrix

In the figure on the right, we can see the values of the confusion matrix.

In the matrix;

- Index 0,0 represents the number true negatives
- Index 0,1 represents the number of false positives
- Index 1,0 represents the number of false negatives
- Index 1,1 represents the number of true positives

To get the accuracy,

$$acc = \frac{true\ negatives + true\ positives}{total\ values} \times 100$$

$$\frac{1502 + 216}{2000} \times 100 = 85.9\%$$

	0	1
0	1502	93
1	189	216

Confusion matrix

Plotting and Saving Keras model

#Plotting (visualizing) the keras model

72: Import the plot_model class from keras.utils

73: The parameters needed are the keras model instance, and the file name to save the plotted model.

#Saving the keras model

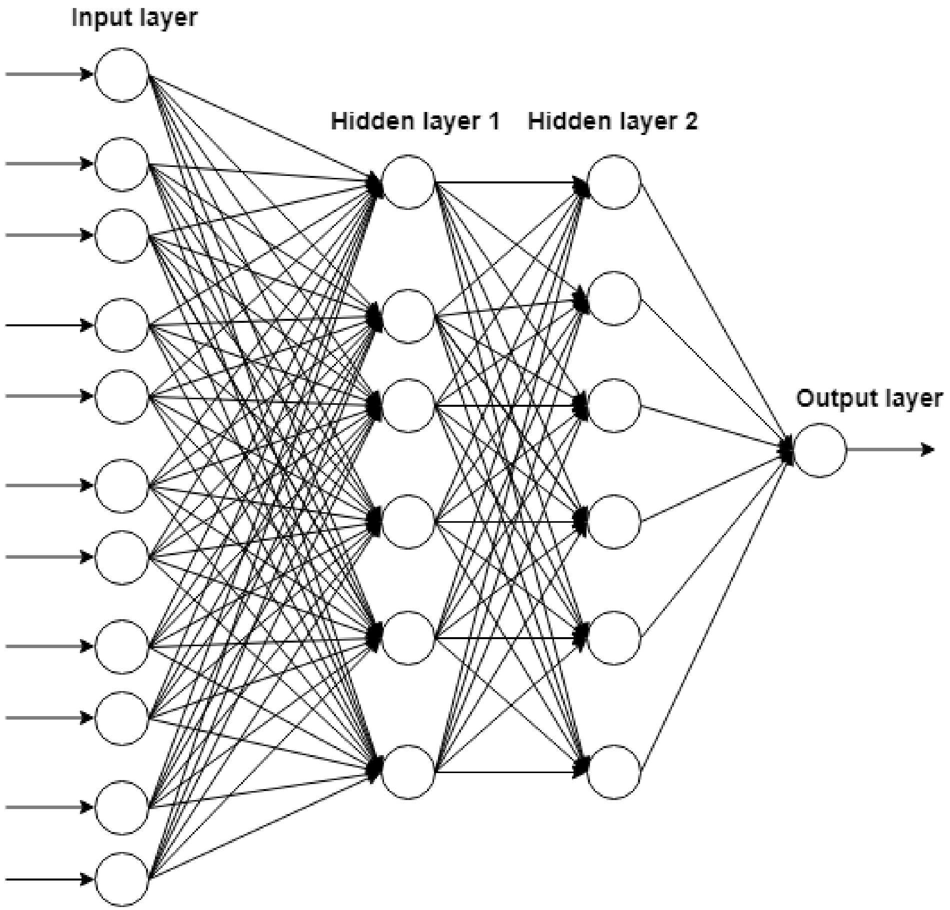
77: Use the to_json() method to return a representation of the model as a JSON string.

78 & 79: Write the contents of model_json in line 77 to file on disk 'model.json'

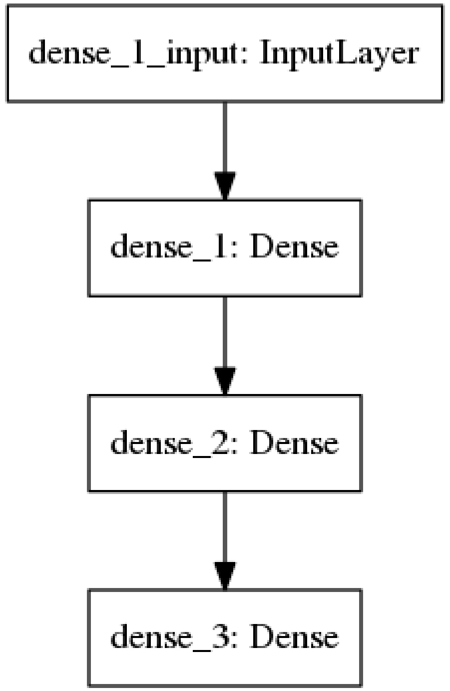
81: Save the weights of the model as a HDF5 file.

```
69  # Part 4 - Ploting, Saving and Loading the model
70
71  #Ploting the model
72  from keras.utils import plot_model
73  plot_model(classifier.model, to_file='model.png')
74
75  #Saving the model
76  # serialize model to JSON
77  model_json = classifier.model.to_json()
78  with open("model.json", "w") as json_file:
79      json_file.write(model_json)
80  # serialize weights to HDF5
81  classifier.model.save_weights("model.h5")
82  print("Saved model to disk")
```

Plotting and Saving Keras model



Conceptualized Neural network architecture



Keras representation of the model

Loading Keras model (From disk)

#Loading the keras model

88: Import the model_from_json class from keras.models

89-91: Read the json file containing the model from disk and store in a variable.

92: Use model_from_json to process the json file and create an instance of the keras model

94: load the model weights using the load_weights() method.

98: Compile the loaded model

101: Make predictions with the loaded model

```
83
84  # later...
85
86  # loading the model
87  # load json and create model
88  from keras.models import model_from_json
89  json_file = open('model.json', 'r')
90  loaded_model_json = json_file.read()
91  json_file.close()
92  loaded_model = model_from_json(loaded_model_json)
93  # load weights into new model
94  loaded_model.load_weights("model.h5")
95  print("Loaded model from disk")
96
97  # compile loaded model
98  classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
99
100 # Predicting the Test set results with loaded model
101 y_pred = loaded_model.predict(X_test)
102
```

Practical Session Recap

- Data preprocessing using Sci-kit learn's **LabelEncoder** and **OneHotEncoder** class.
- Cross validation using Sci-kit learn's **train_test_split** class.
- Feature scaling using Sci-kit learn's **StandardScaler** class.
- Built the ANN using the **Keras.models.Sequential** functional API and **Keras.layers.Dense** class.
- Evaluated the training results
- Tested the model on a test set
- Evaluated the test set performance using the **confusion_matrix** class
- Plotted and Saved the model using Keras **plot_model()** and **to_json()** methods
- Loaded the model using Keras **model_from_json()** and **load_weights()** method.

Home work

On your own, try to make a single prediction (on one customer) using the deep learning model you built.

Future work

- Try training the model for more epochs. Does this improve accuracy?
- Try to improve the models performance by looking into regularization techniques e.g. Dropout.
- Build a regression model using Keras!
- Keras has multiple layers for Computer vision and Natural language tasks; try to implement a project in one of these applied areas
- Further reading on the theoretical concepts behind deep learning.

Recommended Book on Deep learning with Python;

- Deep learning with Python by François Chollet

Thank you and Goodluck!