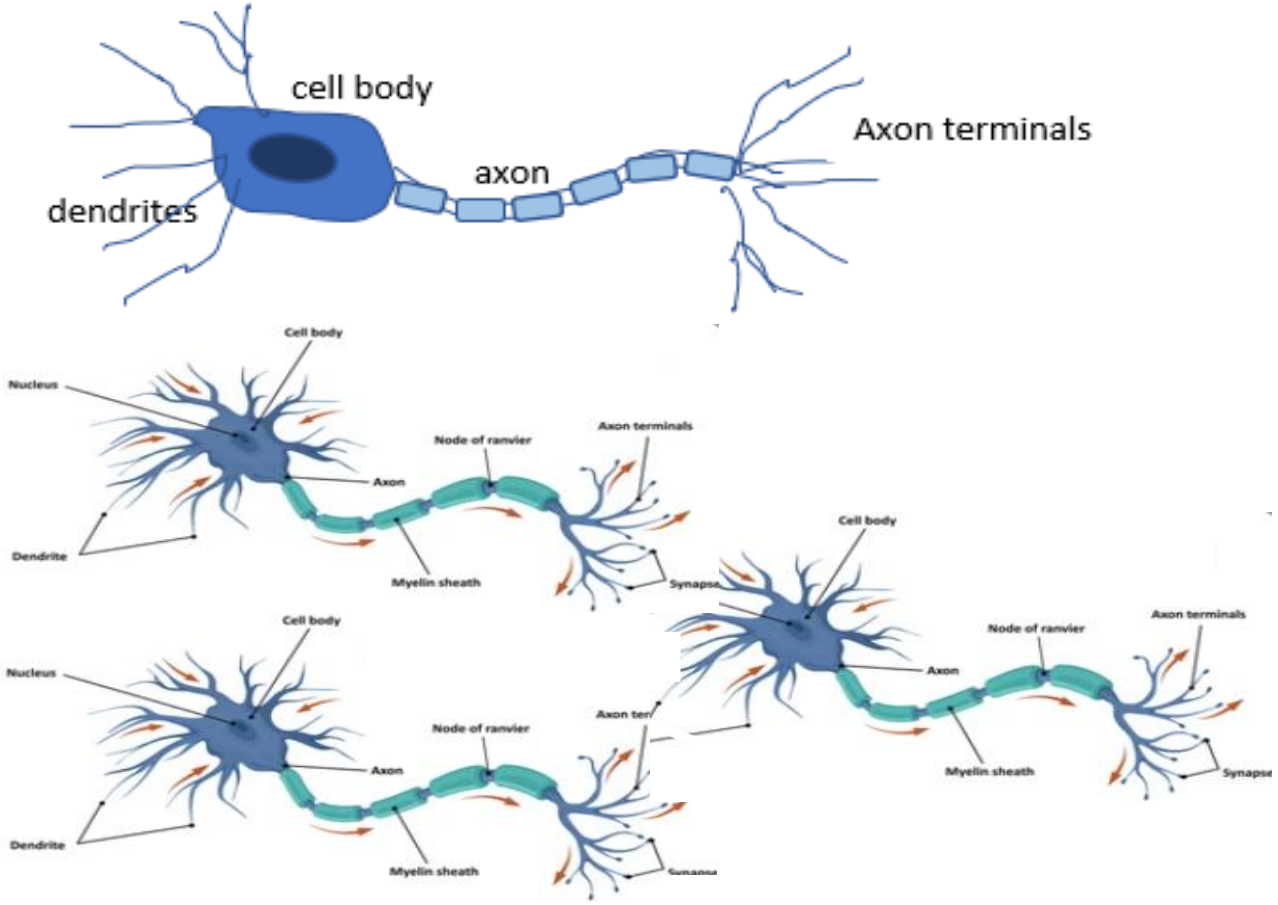
QF634 APPLIED QUANTITATIVE RESEARCH METHODS
LECTURE 7

Lecturer: Prof Lim Kian Guan

ARTIFICIAL INTELLIGENCE

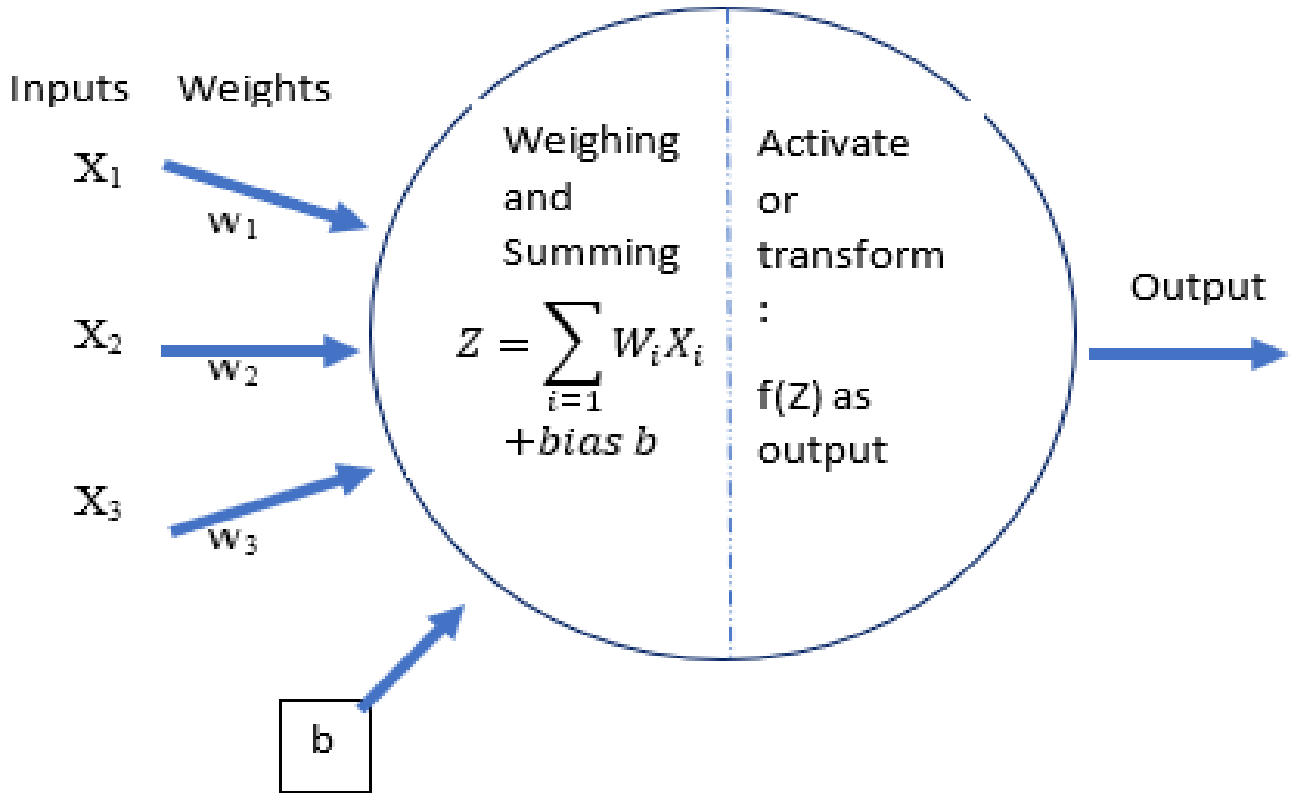
Artificial Neural Network I

Neurons – Nerve Cells in the Brain



- Neurons are information messengers. They transmit information between different areas of the brain and the rest of the nervous system.
- Neurons communicate with each other by sending neurotransmitters, across a tiny spaces – synapse -- between the axons and dendrites of adjacent neurons.
- Neurons can connect with other neurons to form a chain of complex input and output responses.

Single Layer Perceptron



- Takes inputs or features of a subject or case, $\{X_1, X_2. X_3\}$
- Aggregates it with corresponding weights $\{W_1, W_2, W_3\}$, adds a bias term b

$$\sum_{i=1}^{3} W_i X_i + b$$

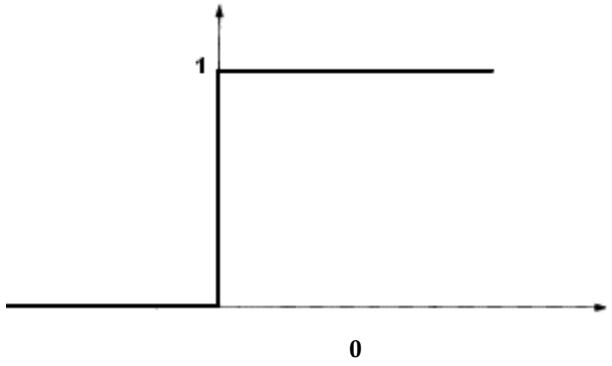
- Then computes the output

$$f\left(\sum_{i=1}^{3} W_i X_i + b\right)$$

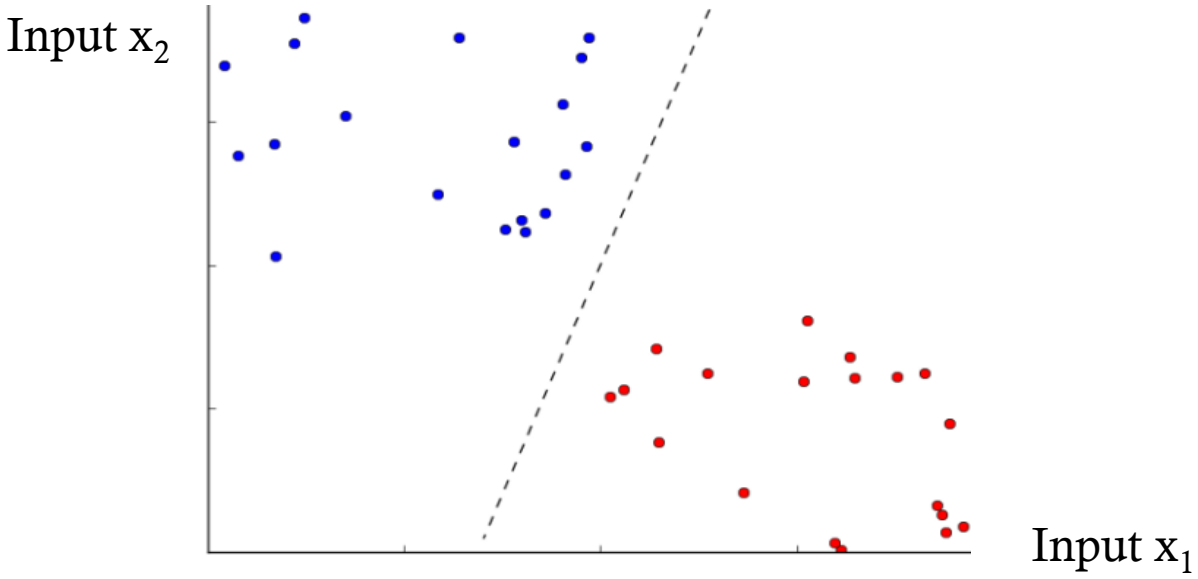
- The bias (or adjustment of mean) can be treated like an input of 1 with weight b.
- Weights W_j's in a NN are initiated typically as small positive or negative numbers, e.g., between (-1,+1).

Example of Activation Function in a Single Layer Perceptron

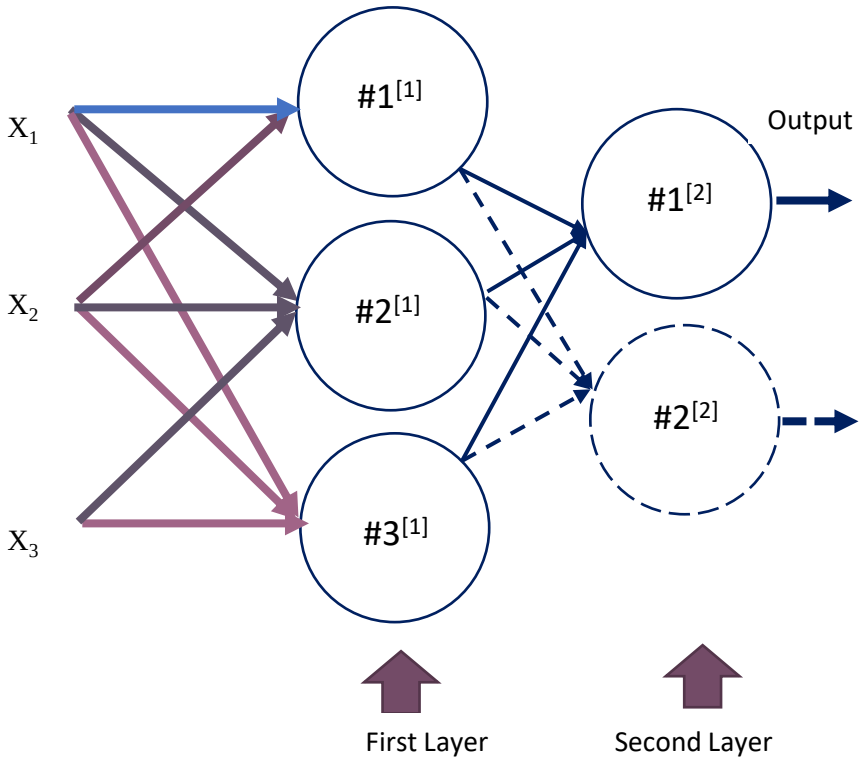
$f(z) = 1_{z > \text{threshold a}}$



This kind of binary output is typical in a linear binary classifier – here's an example where there are two inputs



MultiLayer Perceptron (MLP)



Many-to-One ANN

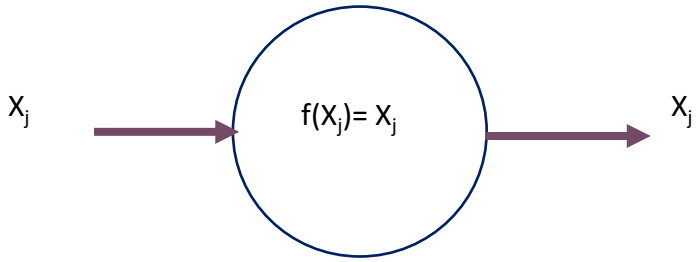
Many-to-Many ANN

No. of output nodes is one if the model is training to predict a single number or a binary classification. But no. of nodes can be more than one if the classification is not just binary but many categories.

Rule of thumb is that the no of nodes in hidden layer could be about 2/3 those of in input layer + output layer, and less than twice the no. in input layer.

Pruning removes nodes where input weights are close to zero – pruning occurs during training of model. Normalizing features also avoids some weights being close to 0 as out-scale feature sizes are standardized.

MultiLayer Perceptron (MLP)

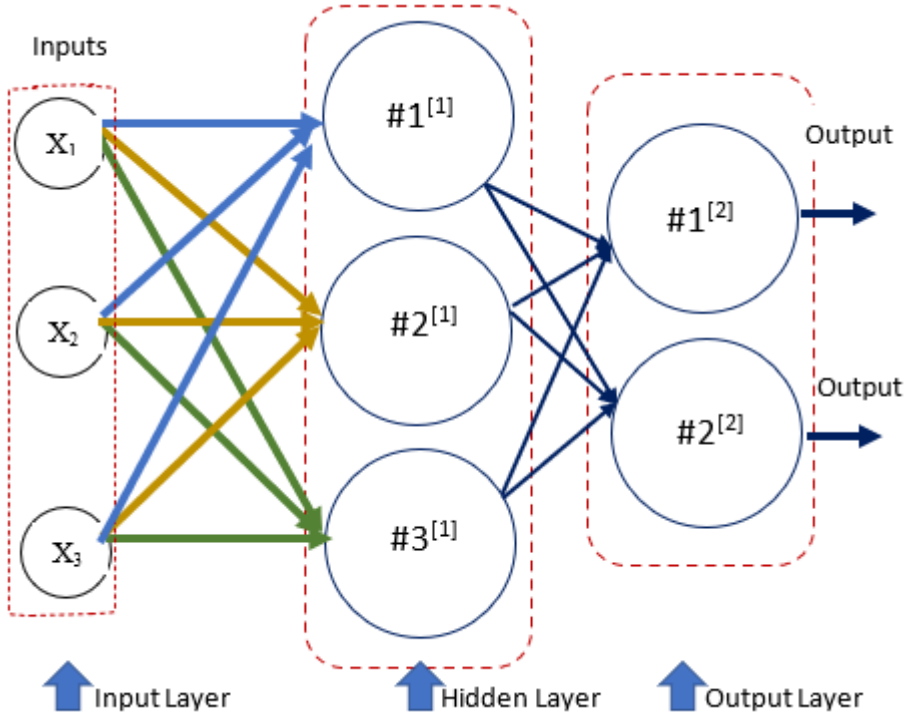


If we consider an input X_j producing identical output X_i (with a trivial identity activation function), then we can treat an input as a neuron.

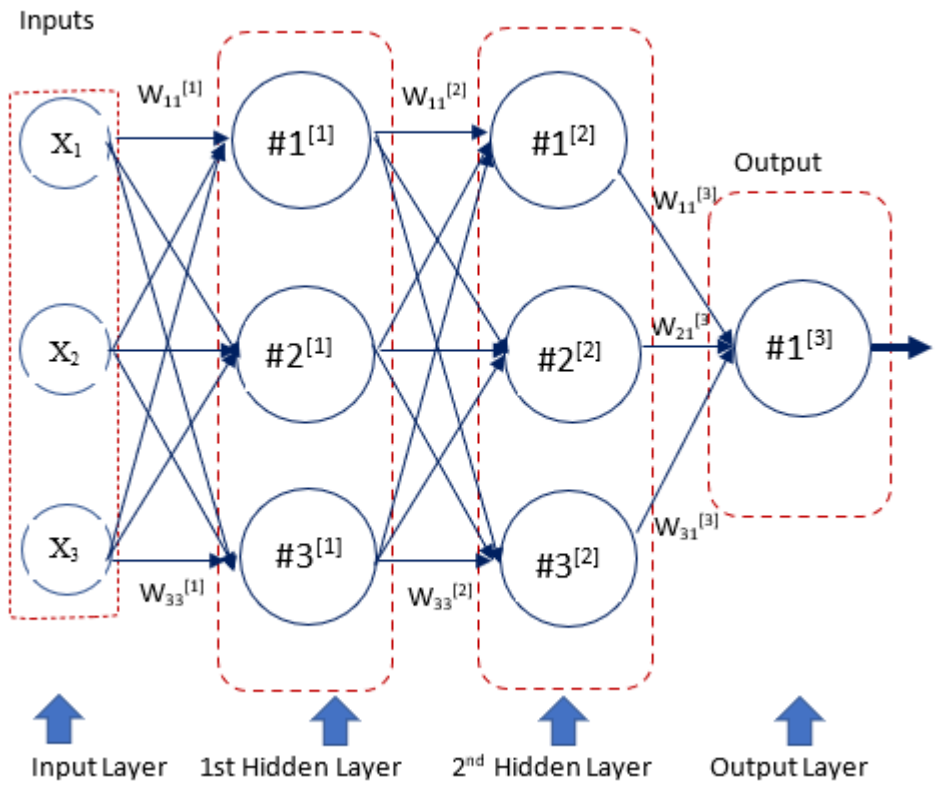
Middle layer called "hidden layer" as the computed outputs from the nodes in this layer are not explicitly observed

For the same prediction or classification problem, an ANN can be made wider (larger number of input nodes or neurons in the same layer) and/or deeper (higher number of hidden layers). These affect the size (total number of neurons in the NN).

Each neuron will connect with all neurons in the forward pass process.



MultiLayer Perceptron (MLP) – weights and biases



The weights of input X_1 on neurons $\#1^{[1]}$, $\#2^{[1]}$, $\#3^{[1]}$ are respectively $W_{11}^{[1]}$, $W_{12}^{[1]}$, and $W_{13}^{[1]}$. The weights of input X_2 on neurons $\#1^{[1]}$, $\#2^{[1]}$, $\#3^{[1]}$ are respectively $W_{21}^{[1]}$, $W_{22}^{[1]}$, and $W_{23}^{[1]}$. The weights of input X_3 on neurons $\#1^{[1]}$, $\#2^{[1]}$, $\#3^{[1]}$ are respectively $W_{31}^{[1]}$, $W_{32}^{[1]}$, and $W_{33}^{[1]}$.

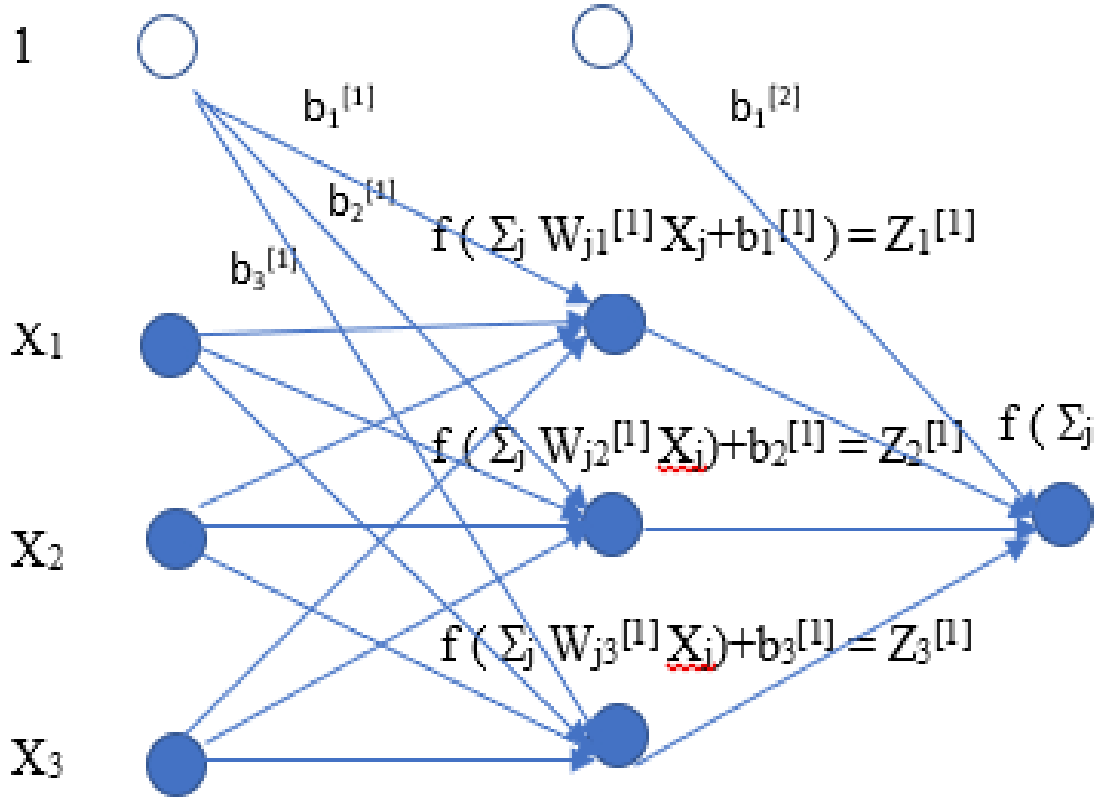
The weights of output from neuron $\#1^{[1]}$ on neurons $\#1^{[2]}$, $\#2^{[2]}$, $\#3^{[2]}$ are respectively $W_{11}^{[2]}$, $W_{12}^{[2]}$, and $W_{13}^{[2]}$. The weights of output from neuron $\#2^{[1]}$ on neurons $\#1^{[2]}$, $\#2^{[2]}$, $\#3^{[2]}$ are respectively $W_{21}^{[2]}$, $W_{22}^{[2]}$, and $W_{23}^{[2]}$. The weights of output from neuron $\#3^{[1]}$ on neurons $\#1^{[2]}$, $\#2^{[2]}$, $\#3^{[2]}$ are respectively $W_{31}^{[2]}$, $W_{32}^{[2]}$, and $W_{33}^{[2]}$.

The weights of outputs from neurons $\#1^{[2]}$, $\#2^{[2]}$, $\#3^{[2]}$ on neuron $\#1^{[3]}$ are respectively $W_{11}^{[3]}$, $W_{21}^{[3]}$, and $W_{31}^{[3]}$. In general, the weight from n^{th} hidden layer $\#j^{[n]}$ neuron to the next layer $\#k^{[n+1]}$ neuron is $wjk^{[n+1]}$.

Artificial Neural Network

- Purpose is to perform prediction or classification on a set of data containing subjects or cases with associated features or characteristics.
- To prepare a model for prediction with new data of features, the model must be trained using many known cases/subjects with their known target statuses and with associated features/characteristics or conditions in the training data set.
- Once the model is adequately trained, the weights are determined optimally by this stage, and the ANN can perform the necessary computations based on these optimal weights (including optimal biases) and the fresh inputs to find the output(s) or target value(s) in the test data set.

Forward Propagation in a Single Hidden Layer MLP



Given the various weights, biases, and inputs, the final output is

$$f(\Sigma_j W_{j1}^{[2]} Z_j^{[1]} + b_1^{[2]}) = \hat{Y}$$

where superscript [1] to output Z_j denotes output is at hidden layer 1. In a standard ANN structure, the activation function f(.) remains the same for every layer.

Examples of Nonlinear Activation Function in a Single Layer Perceptron

- Logistic Sigmoid Function (S-Curve)

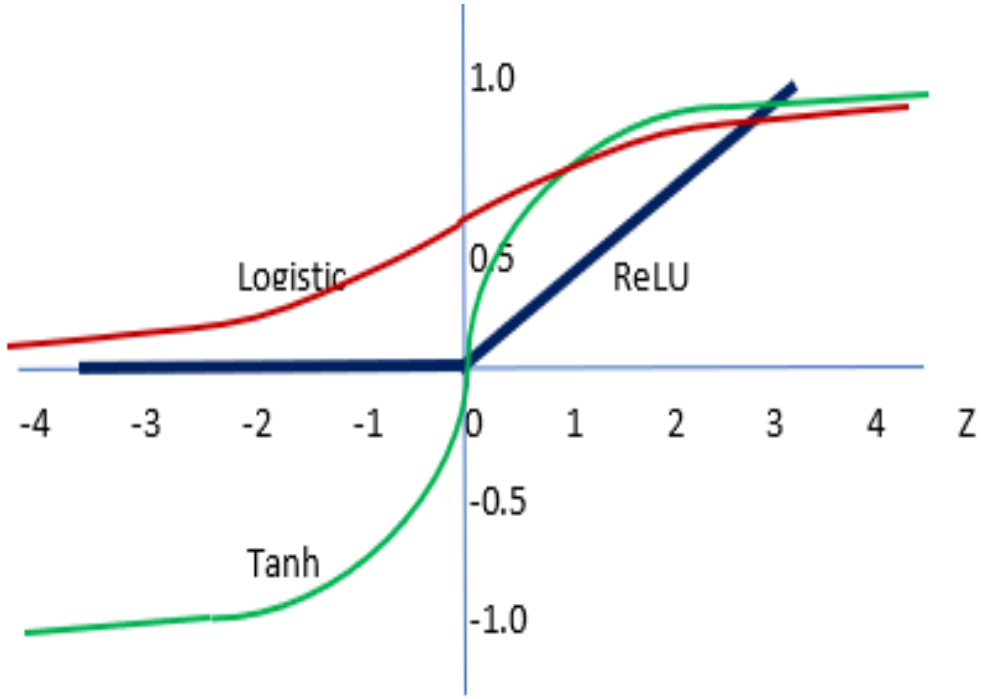
$$f(z) = (1 + e^{-z})^{-1}$$

- Hyperbolic Tangent Function (S-Curve)

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

- Rectifier Linear Unit Function (ReLU)

$$f(z) = \max(z, 0)$$



Error or Loss Function for Continuous Variable Target

- The training prediction error is an error or loss function based on the training data set of all training sample points and measures 'deviations' of NN outputs (target predictions) of all cases with the actual outputs of the associated cases.
- Training the model is to find optimal weights and biases so that the training set error or loss function is minimized or reduced toward zero.
- Commonly used error/loss function is the average least squares of deviations (or mean square error, MSE) L^2:

$$L(Y, \hat{Y}) \equiv \frac{1}{N}\sum_{k=1}^{N} L\left(Y_k, \hat{Y}_k\right) = \frac{1}{N}\sum_{k=1}^{N}\left(Y_k - \hat{Y}_k\right)^2$$

- An alternative is the mean absolute loss or L^1 loss function

$$L(Y, \hat{Y}) = \frac{1}{N}\sum_{k=1}^{N}\left|Y_k - \hat{Y}_k\right|$$

- In scenarios where there are outliers, MSE >> MAE. MAE is thus more robust in the presence of some outliers, i.e., producing more stable results. However, MSE has better convergence properties when it comes to finding optimal weights.

Error or Loss Function for Categorical Variable Target

- Typically, a binary classification prediction using features would provide a probability estimate $\hat{P}$ of the target status whether it is 1. Otherwise, 1-$\hat{P}$ would be estimate of the probability the target status is 0.

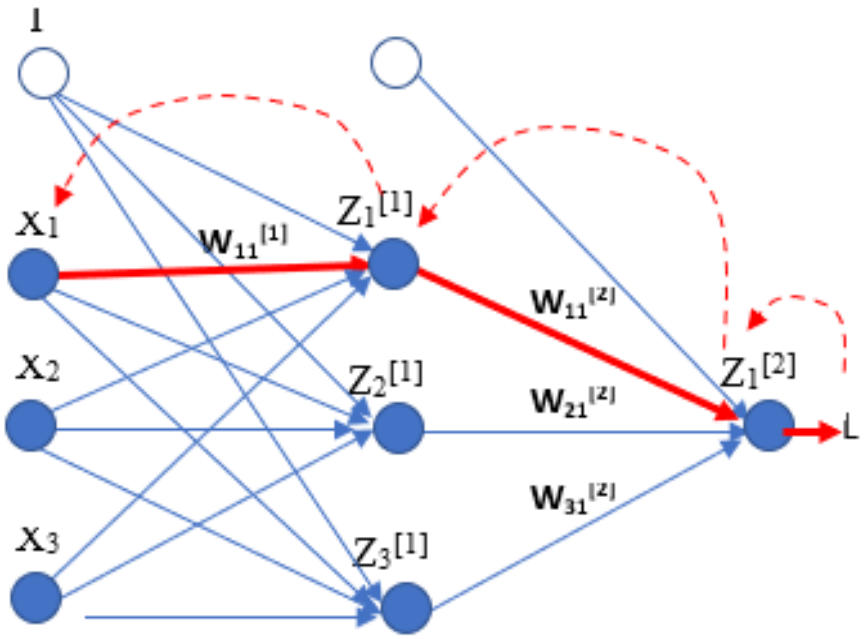
- A useful error/loss function for the binary classification is binary cross-entropy that employs estimate $\hat{P}_k$for each case k: $L(Y, \hat{P}) = \frac{1}{N}\sum_{k=1}^{N} L(Y_k, \hat{P}_k)$ where

$$L(Y_k, \hat{P}_k) = -[\, Y_k \, ln \, \hat{P}_k + (1-Y_k) \ln (1 - \hat{P}_k) \,].$$

  $\hat{P}_k$ is the predicted probability that $Y_k = 1$, and $1 - \hat{P}_k$ is the predicted probability that $Y_k = 0$.

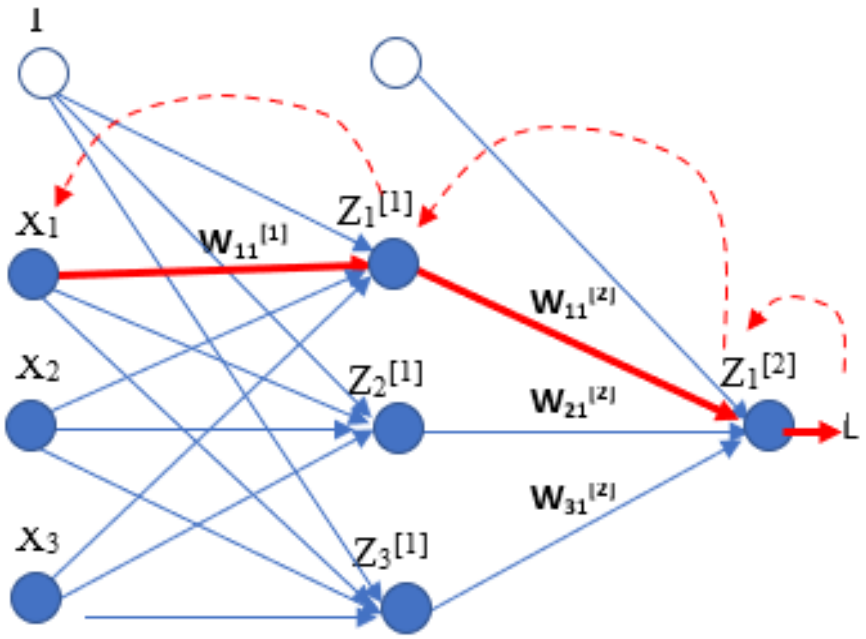
- Note that here when $Y_k = 1$, the loss function is reduced when $\hat{p}_k \in (0,1)$ is higher. When $Y_k = 0$, the loss function is reduced when $\hat{p}_k \in (0,1)$ is lower. This loss function therefore captures the idea of ensuring the estimated or predicted probability of ($Y_k = 1$), $\hat{P}_k$, is high when $Y_k = 1$, and that the predicted probability of ($Y_k = 1$), $\hat{P}_k$, is low when $Y_k = 0$.

Backward Propagation



- In each epoch or forward propagation of the entire training data set from inputs to final output(s) in the output layer, the predicted outputs are compared with the actual (target) outputs and the loss function $L(Y, \hat{Z})$ is computed.
- When loss is not acceptable, the model needs further improvement – we need to train the NN further, by updating/revising/fine-tuning the weights and bias and then run a second epoch, and so on until it hopefully ends with an acceptable loss function outcome.
- The Backward Propagation is the mechanism of training the NN toward an acceptable loss before proceeding for application to the test data set.
- Backward propagation involves backwards pass that performs computations by moving backward from the loss function at the final output to the weight or bias inputs.

Backward Propagation



- In this structure there are altogether 16 parameters of weights and biases to update/revise/fine-tune: $\{W_{11}^{[1]}, W_{21}^{[1]}, W_{31}^{[1]}, b_1^{[1]}, W_{12}^{[1]}, W_{22}^{[1]}, W_{32}^{[1]}, b_2^{[1]}, W_{13}^{[1]}, W_{23}^{[1]}, W_{33}^{[1]}, b_3^{[1]}, W_{11}^{[2]}, W_{21}^{[2]}, W_{31}^{[2]}, b_1^{[2]}\}$.
- To update $W_{11}^{[1]}$ toward minimizing the error/loss objective function, a standard approach is to use the gradient descent method/algorithm or its variants.
- The gradient descent method requires the computation of the partial slope of the objective function with respect to each of the weights and biases. The estimation uses the chain rule. In the single layer MLP, the diagram below shows

$$\frac{\partial L(Y,\hat{Z})}{\partial W_{11}^{[1]}} = \frac{\partial \sum_{k=1}^{N} \frac{1}{N} L(Y_k,\hat{Z}_k)}{\partial W_{11}^{[1]}}$$

$$\frac{\partial L(Y_k,\hat{Z}_k)}{\partial W_{11}^{[1]}} = \left(\frac{\partial L(Y_k,\hat{Z}_k)}{\partial Z_{1,k}^{[2]}} \times \frac{\partial Z_{1,k}^{[2]}}{\partial \Sigma_j (W_{j1}^{[2]} Z_{j,k}^{[1]} + b_1^{[2]})}\right) \times \left(\frac{\partial \Sigma_j (W_{j1}^{[2]} Z_{j,k}^{[1]} + b_1^{[2]})}{\partial Z_{1,k}^{[1]}} \times \frac{\partial Z_{1,k}^{[1]}}{\partial \Sigma_j (W_{j1}^{[1]} X_j + b_1^{[1]})}\right) \times \frac{\partial \Sigma_j (W_{j1}^{[1]} X_j + b_1^{[1]})}{\partial W_{11}^{[1]}}$$

Backward Propagation

For example, if $L(Y,\hat{Y}) = \frac{1}{N}\sum_{k=1}^{N}(Y_k - \hat{Y}_k)^2$, then $\frac{\partial L(Y_k,\hat{Y}_k)}{\partial Z_{1,k}^{[2]}} = 2(Y_k - \hat{Y}_k)$.

If f(X) = f (X) = 1/(1 + e^{-X}), then $\frac{\partial Z_{1,k}^{[2]}}{\partial \sum_j (W_{j1}^{[2]} Z_{j,k}^{[1]} + b_1^{[2]})} = (1-\hat{Y}_k)\,\hat{Y}_k$.

$$\frac{\partial \sum_j (W_{j1}^{[2]} Z_{j,k}^{[1]} + b_1^{[2]})}{\partial Z_{1,k}^{[1]}} = W_{11}^{[2]} \quad , \quad \frac{\partial Z_{1,k}^{[1]}}{\partial \sum_j (W_{j1}^{[1]} X_j + b_1^{[1]})} = (1-Z_{1,k}^{[1]})\,Z_{1,k}^{[1]} \text{ , and } \frac{\partial \sum_j (W_{j1}^{[1]} X_j + b_1^{[1]})}{\partial W_{11}^{[1]}} = X_1$$

Hence,

$$\frac{\partial L(Y_k,\hat{Y}_k)}{\partial W_{11}^{[1]}} = 2(Y_k - \hat{Y}_k) \times (1 - \hat{Y}_k)\,\hat{Y}_k \times W_{11}^{[2]} \times (1 - Z_{1,k}^{[1]})\,Z_{1,k}^{[1]} \times X_1$$

and

$$\frac{\partial L(Y,\hat{Y})}{\partial W_{11}^{[1]}} = \frac{1}{N}\sum_{k=1}^{N} 2(Y_k - \hat{Y}_k) \times (1 - \hat{Y}_k)\,\hat{Y}_k \times W_{11}^{[2]} \times (1 - Z_{1,k}^{[1]})\,Z_{1,k}^{[1]} \times X_1.$$

Backward Propagation

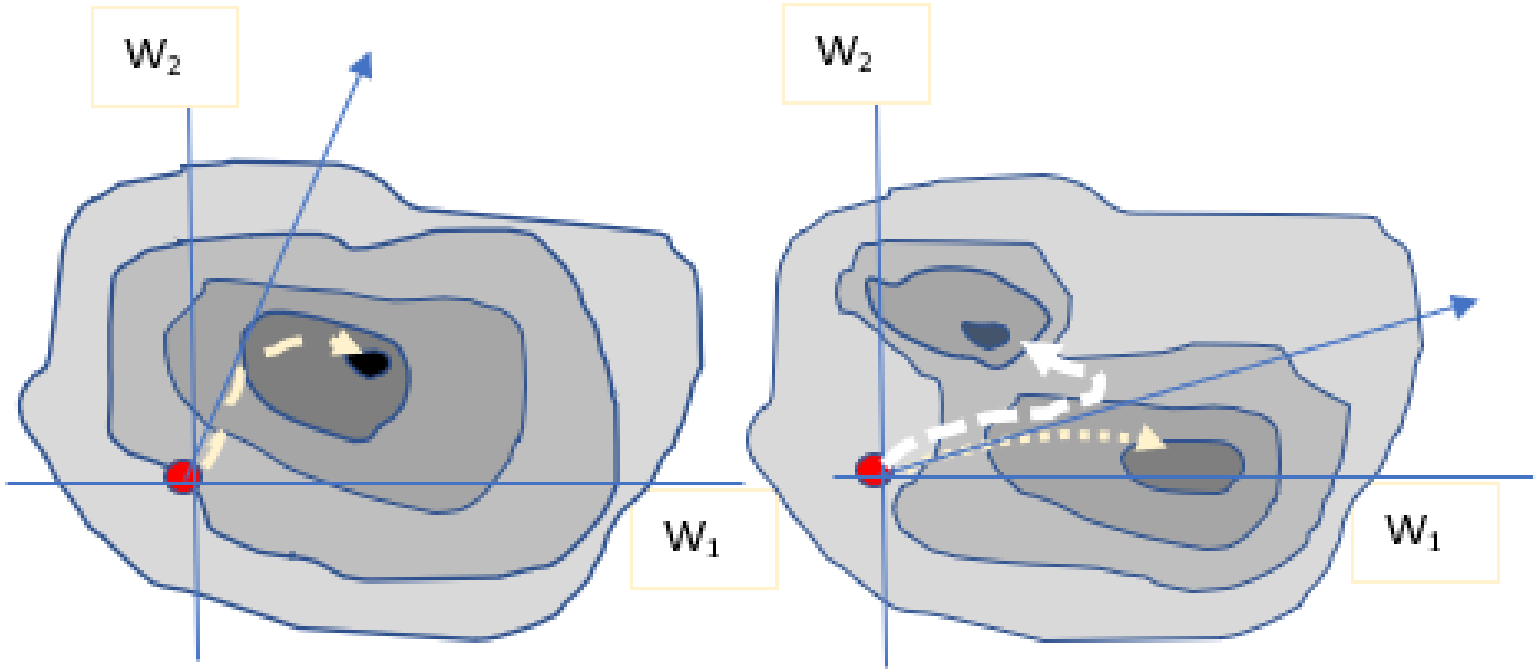
The purpose of finding all derivatives of loss function with respect to all the weights and biases in the ANN is as follows. Suppose for a particular weight (or bias) W, $\partial L/\partial W > (<) \ 0$, then we could try to attain a lower loss by decreasing (increasing) the weight (or bias). To do so, we could find a revised weight (or bias) by adjusting it lower (higher) from the previous one. Hence the adjustment for each epoch updating of all weights at the same time is as follows.

$$W_{ij}^{*[r]} = W_{ij}^{[r]} - \alpha \frac{\partial L}{\partial W_{ij}^{[r]}}$$

where updated weight is $W^{*}_{ij}{}^{[r]}$, and $\alpha > 0$ is a learning rate that determines how fast is the descent toward the minimum. α is a hyperparameter in this algorithm.

Gradient Descent (method of optimization)

The vector $\rightarrow$ representing the linear combination of $(W_1 - \alpha\, \partial L/\partial W_1\,,\, W_2 - \alpha\, \partial L/\partial W_2)$ shows the fastest descent from the current point. Note $\partial L/\partial W_2 > \partial L/\partial W_1$ on the left graph.



Convergence Strategy to Minimum Loss

- Convergence to minimum (or reasonably stable) loss – typically can be achieved in faster computational time if we use a smaller batch size of N/m, m>0. The parameter updates occur m times or m iterations in each epoch, i.e., all the data need to pass through before one epoch is counted. Less number of partial derivatives to computer with small m.
- If E number of epochs are run, then in total there will be E × m number of iterations in updating the parameters in the training. More iterations with mini-batches can improve convergence rate, though convergence can be more volatile.
- Another popular variant is the stochastic gradient descent (SGD) that randomly sequentially computes the gradients and update for each sample point within the batch or within the mini-batch. The updating of the stochastic gradient descent partial derivatives for each sample point can cause the convergence to the minimum error/loss function to be more volatile than when using the (full) batch or mini-batch training data set. However, there are far more iterations.
- Another approach to speed convergence is to adjust the learning rate. One such popular algorithm is the Adam (adaptive moment estimation).

Worked Example – Data

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

- The data set -- "Churn_Modelling.csv" – is found in public resource: https://www. kaggle. com/datasets/. The data set contains a bank's 10,000 customers' details, viz., nationality, gender, credit score (higher score means more credit-worthy), age, tenure (years with the bank), bank balance in the period of the data, the number of bank products used by the customer, whether the customer has the bank credit card, is active customer, and estimated USD salary of the customer.
- These serve as features of each customer that could explain the target variable – which is whether the customer exited or left the bank.
- In the latter the customer had closed his bank account during the data period and the classification of exit is "1". Otherwise, the customer continues with the bank and the classification of exit is "0".

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

```
In [3]: #Generating Dependent Variable Vectors
        Y = data.iloc[:,-1].values
        X = data.iloc[:,3:13]
        X.head()
```

Out[3]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	France	Female	42	2	0.00	1	1	1	101348.88
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
2	502	France	Female	42	8	159660.80	3	1	0	113931.57
3	699	France	Female	39	1	0.00	2	0	0	93826.63
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

In code lines [4] and [5], the Gender variable and the Geography variable are transformed to dummy or 1, 0 variables.

In [4]:
```
#Generating Dependent Variable Vectors
Y = data.iloc[:,-1].values
X = data.iloc[:,3:13]
X['Gender']=X['Gender'].map({'Female':0,'Male':1})
### above is used instead of a more complicated package involving -- from sklearn.preprocessing import LabelEncoder
### converts Female -- 0, Male -- 1, i.e. hot-encoding categorical variables
print (X['Gender'])
```

```
0       0
1       0
2       0
3       0
4       0
       ..
9995    1
9996    1
9997    0
9998    1
9999    0
Name: Gender, Length: 10000, dtype: int64
```

In [5]:
```
#Encoding Categorical variable Geography
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct =ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[1])],remainder="passthrough")
X = np.array(ct.fit_transform(X))
### Geography is transformed into France -- 1,0,0; Spain -- 0,0,1; Germany -- 0,1,0.
### Moreover -- this encoded vector of ones-zeros is now put in first 3 cols. Credit Score pushed to 4th col.
```

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

Code line [6], [7] split the dataset into training and test data sets. Note that validation set is bypassed here as the model is predetermined to be a MLP and the training set itself is used to try out different hyperparameters to select the good performing hyperparameter values. The hyperparameters are the number of hidden layers (depth) and the number of neurons in each layer (width).

In [6]:
```
### convert X to dataframe X1
X1 = pd.DataFrame(X)
X1.head()
### Note there are 12 features including onehotencoder for the Geography feature--
### The features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme
```

In [7]:
```
#Splitting dataset into training and testing dataset
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,random_state=1)
```

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

Next comes the scaling of the data (both features and target variable) which is useful if the features are data with different orders of magnitudes.

In sklearn.preprocessing.StandardScaler(), centering and scaling happens independently on each feature. The fit method is calculating the mean and variance of each of the features present in the data. The transform method is transforming all the features (and target variable) using the respective features' (target)'s mean and variance.

In [8]:
```
#Performing Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

Code lines [9] to [12] use the tf.keras.models.Sequential() API to build the MLP neural network model.

In [9]:
```
### This is the very first step while creating NNmodel -- you can rename this model. Here we are going to create NNmodel object
### by using a certain class of Keras named Sequential. As a part of tensorflow 2.0, Keras is now integrated with
### tensorflow and is now considered as a sub-library of tensorflow. The Sequential class is a part of the models module
### of Keras library which is a part of the tensorflow library now.
### It used to be "import tensorflow as tf; from tensorflow import keras; from tensorflow.keras import layers"
### See documentation at https://keras.io/guides/sequential_model/

#Initialising the NN model name -- NNmodel
NNmodel = tf.keras.models.Sequential()
### Sequential specifies to keras that the model NNmodel is created sequentially and the output of each layer added
### is input to the next specified layer. Note that keras Sequential is not appropriate when the model has multiple outputs
```

In [10]:
```
### Creating a network that has 1 hidden layer together with 1 input layer and 1 output layer.
#Adding First Hidden Layer
NNmodel.add(tf.keras.layers.Dense(units=2,activation="sigmoid"))
### units = 2 refer to 2 neurons in hidden layer
### modelname.add is used to add a layer to the neural network -- need to specify as an argument what type of layer --
### Dense is used to specify the fully connected layer - i.e. all neurons are forward connect to all forward layer nodes
```

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

```
In [11]: ### now we create the output layer
         #Adding Output Layer
         NNmodel.add(tf.keras.layers.Dense(units=1,activation="sigmoid"))
         ### Only 1 output neuron
```

For a binary classification problem as above, actual case output is 1 or 0. Hence we require only one neuron to output layer - output could be estimated probability of case's actual output = 1. In the binary output case, the suitable activation function is the sigmoid function.

```
In [12]: ### After creating the layers -- require compiling the NNmodel. Compiling allows the computer to run and understand the program
         ### without the need of more fundamental steps in the programming. Compiling adds other elements or linking other libraries, and
         ### such that after compiling the results are readily computed e.g. in a binary executable program as an output.
         #Compiling NNmodel
         NNmodel.compile(optimizer="adam",loss="binary_crossentropy",metrics=['accuracy'])
         ### Note optimizer here is a more sophisticated version of the Mean Square loss
```

Compile method above accepts inputs -- (1) optimizer: specifies which optimizer to be used in order to perform stochastic gradient descent (2) error/loss function, e.g., 'binary_crossentropy' here. For multiclass classification, it should be categorical_crossentropy, (3) metrics is the performance metrics to use in order to compute performance. 'accuracy' is one such performance metric.

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

In code line [14], we run the training of the NN on the training data set. This uses the '.fit' in TensorFlow-Keras.

```
In [14]: #### Last step in creation of NNmodel NNmodel is trained on the training set here with Tensor-Keras .fit based on Compiler
         #Fitting NNmodel
         history=NNmodel.fit(X_train,Y_train,batch_size=8000,epochs = 100)
```

```
Epoch 95/100
1/1 [==============================] - 0s 0s/step - loss: 0.6114 - accuracy: 0.7150
Epoch 96/100
1/1 [==============================] - 0s 0s/step - loss: 0.6108 - accuracy: 0.7163
Epoch 97/100
1/1 [==============================] - 0s 0s/step - loss: 0.6102 - accuracy: 0.7176
Epoch 98/100
1/1 [==============================] - 0s 2ms/step - loss: 0.6097 - accuracy: 0.7195
Epoch 99/100
1/1 [==============================] - 0s 0s/step - loss: 0.6091 - accuracy: 0.7201
Epoch 100/100
1/1 [==============================] - 0s 0s/step - loss: 0.6085 - accuracy: 0.7211
```

It is noted that after 100 epochs, the loss is 0.6085 and accuracy (% of correct predictions of exit or retention) is 0.7211 (72.11%).

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

Code line [15] provides a summary of the architecture/structure of this NN

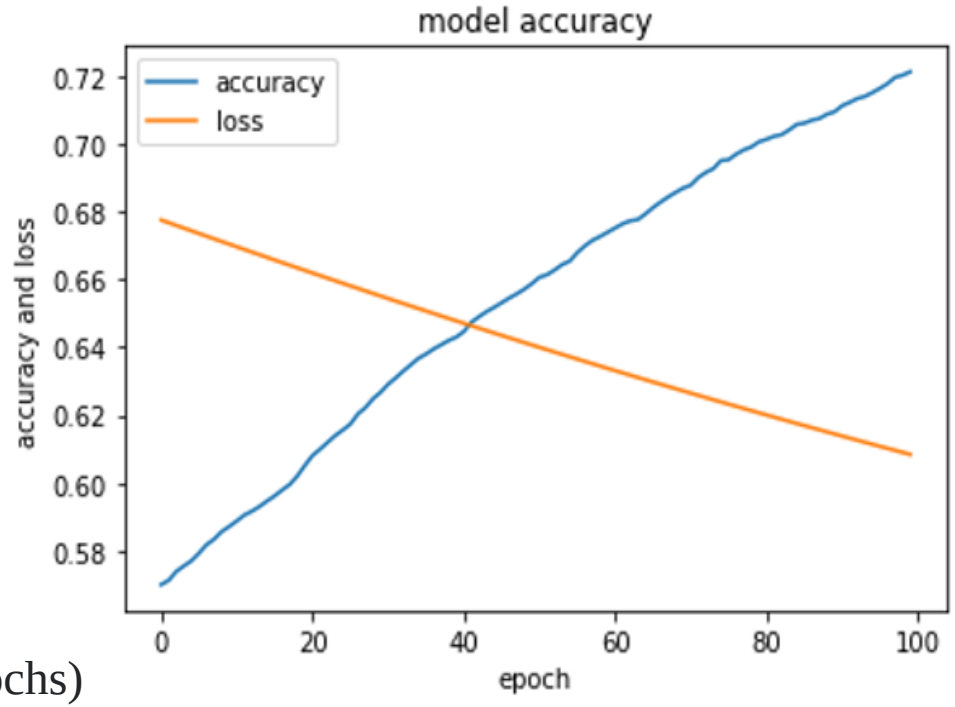
```
In [15]: NNmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (8000, 2)                 26

 dense_1 (Dense)             (8000, 1)                 3

=================================================================
Total params: 29
Trainable params: 29
Non-trainable params: 0
_________________________________________________________________
```

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

The outputs of losses and accuracies in code line [14] can be captured in the 'history' output of the NNmodel. They are plotted as shown in [16].

```
In [16]: import matplotlib.pyplot as plt
         print(history.history['accuracy'])
         print(history.history['loss'])
         plt.plot(history.history['accuracy'])
         plt.plot(history.history['loss'])
         plt.title('model accuracy')
         plt.ylabel('accuracy and loss')
         plt.xlabel('epoch')
         plt.legend(['accuracy', 'loss'], loc='upper left')
         plt.show()
```



The plot shows that as the number of training iterations (epochs) increases, the loss decreases, and the accuracy increases to about 72%.

Please upload Chapter7-1.ipynb and follow the computing steps in Jupyter Notebook

In code line [29], we use '.evaluate' (based on the trained NN in [14]) to predict outputs for X_train and then compare with the actual outputs in Y_train. This returns a loss of 0.6079 and accuracy of 0.7229.

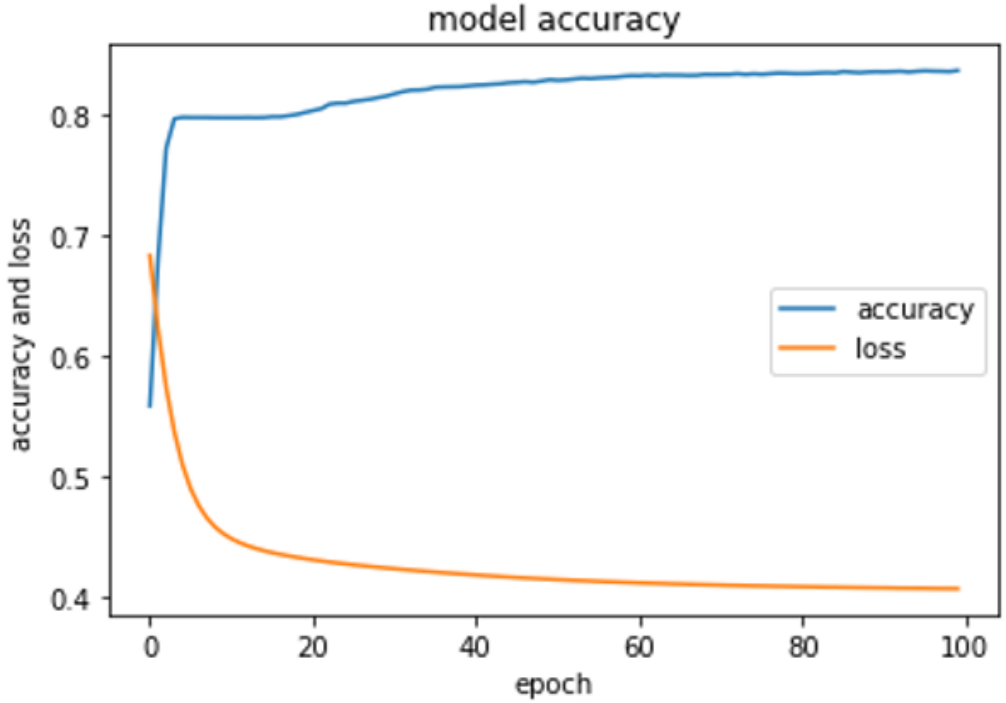
```
In [29]: ### Now we use the trained NNmodel to predict output in X_train sample
         NNmodel.evaluate(X_train,Y_train)  ### evaluates the loss and accuracy as specified in the Compiler

         250/250 [==============================] - 0s 520us/step - loss: 0.6079 - accuracy: 0.7229

Out[29]: [0.6078730225563049, 0.7228749990463257]
```

In code line [40], we finally use the trained NN to perform prediction of exits in the test sample (of 2000 points). This is verified in [41], [42] as 72.50%.

```
In [40]: ### Now we use the trained NNmodel to predict output in X_test sample
         NNmodel.evaluate(X_test,Y_test)  ### evaluates the loss and accuracy as specified in the Compiler

         63/63 [==============================] - 0s 521us/step - loss: 0.6028 - accuracy: 0.7250

Out[40]: [0.6028271913528442, 0.7250000238418579]
```

Improving the NN and Understanding the Results

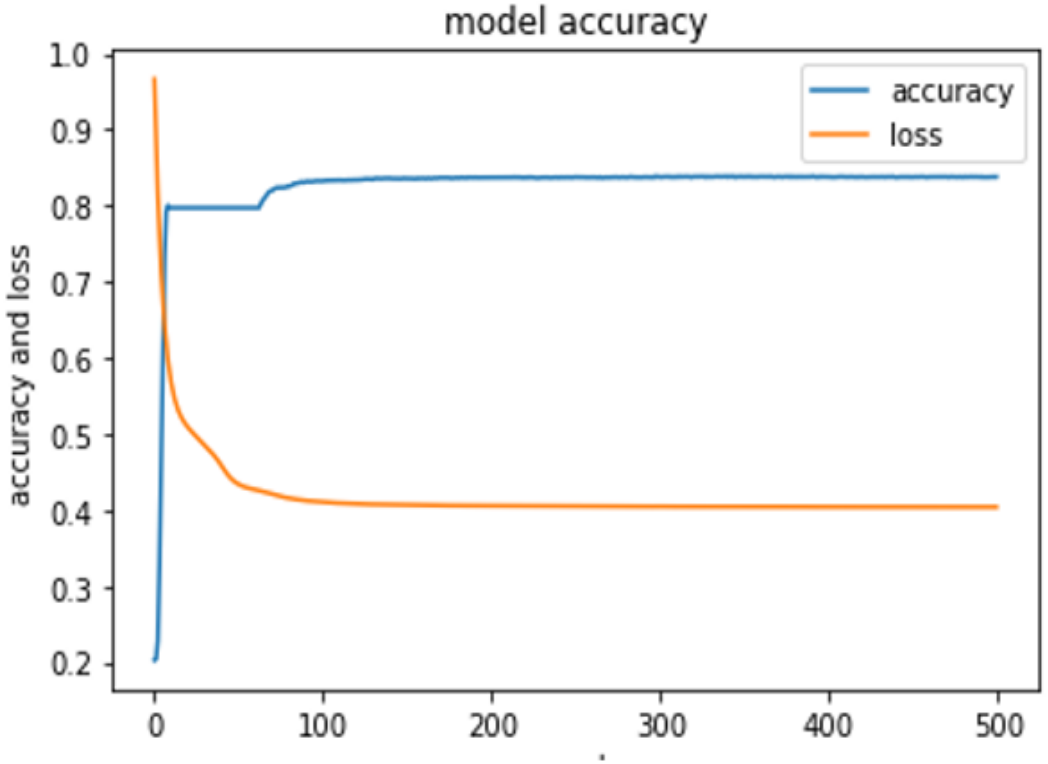
Sometimes a single batch may not yield robust or stable results. We increase the number of backward passes and iterations, hence parameter revisions, by using "mini-batches" of batch size = 100 (instead of the entire batch size of 8000), and still using 100 epochs. This yields the following loss and accuracy graph. See demonstration file Chapter7-2.ipynb.



After 3 epochs or 240 iterations (each epoch has 80 mini-batch iterations; 80 = 8000/100), the accuracy had already climbed to over 77%. The loss and accuracy stabilize at higher epochs, ending with training accuracy of 83.58%. The test accuracy is 83.35%.

Improving the NN

We also document the case of keeping batch size = 100 but using to 500 epochs. This created a total of 80 × 500 = 40,000 iterations. This yields the following loss and accuracy graph. See demonstration file Chapter7-3.ipynb.



The loss and accuracy stabilize after about 10 epochs, ending with training accuracy of 83.80%. The test accuracy is 83.30%. Both accuracies are about similar to the case of batch size = 100 and 100 epochs.

Predictions for the additional models/hyperparameters: (1), (2), (3)

We employ the same batch size = 100 and 500 epochs, but (1) adjust the optimizer to SGD instead of Adam, (2) add a second hidden layer with now 4 neurons in both the first and second hidden layers. We create an even deeper NN in (3) where the number of neurons in the first and the second hidden layers are increased from 4 to 8.

Metrics:	(1)	(2)	(3)
Training Set			
Loss	0.4193	0.3424	0.3230
Accuracy (%)	81.69%	86.11%	86.80%
Test Set			
Loss	0.4130	0.3359	0.3319
Accuracy (%)	82.35%	86.25%	86.25%

The prediction accuracy using the estimated logit regression coefficients (where estimated probability > 0.5 is designated as predicting 1, otherwise 0) shows 81.25%. See demonstration files Chapter7-4,-5,-6.ipynb.

We also perform a logit regression of the exit variable (1 or 0) on the feature variables using package 'statsmodel' in python. The regression outputs are summarized as follows. See demonstration file Chapter7-7.ipynb. The estimated coefficients and their statistical significance provide some observations to help understand how the NN outputs indicate regarding the influence of the features on the prediction of 'exit'.

Logit Regression Results

Dep. Variable:	y	No. Observations:	8000
Model:	Logit	Df Residuals:	7988
Method:	MLE	Df Model:	11
Date:	Fri, 21 Oct 2022	Pseudo R-squ.:	0.1490
Time:	10:57:49	Log-Likelihood:	-3432.6
converged:	True	LL-Null:	-4033.5
Covariance Type:	nonrobust	LLR p-value:	6.481e-251

	coef	std err	z	P>|z|	[0.025	0.975]
Constant	-1.6549	0.035	-47.918	0.000	-1.723	-1.587
France	-0.1232	1.73e+06	-7.12e-08	1.000	-3.39e+06	3.39e+06
Spain	0.2178	1.51e+06	1.44e-07	1.000	-2.95e+06	2.95e+06
Germany	-0.0768	1.49e+06	-5.14e-08	1.000	-2.93e+06	2.93e+06
CrScore	-0.0535	0.030	-1.771	0.077	-0.113	0.006
Gender	-0.2657	0.030	-8.768	0.000	-0.325	-0.206
Age	0.7477	0.030	24.800	0.000	0.689	0.807
Tenure	-0.0279	0.030	-0.924	0.355	-0.087	0.031
Balance	0.1694	0.036	4.715	0.000	0.099	0.240
Products	-0.0403	0.031	-1.319	0.187	-0.100	0.020
CrCard	-0.0367	0.030	-1.220	0.222	-0.096	0.022
Active	-0.5499	0.032	-16.996	0.000	-0.613	-0.486
Salary	0.0161	0.030	0.530	0.596	-0.043	0.076

It is seen that (at 10% 2-tailed significance level or a p-value of < 0.10), lower credit score (CrScore), Gender (Male = 1, Female = 0), higher Age, higher Balance, and low activity (Active) increases exit probability, and vice-versa.

Understanding the Results

We can use the trained NN in (3) to check on the effect of changing the feature input levels on predicted probability of exit. For example, in checking the effect of credit score levels, the following outputs show that the output of NNmodel.predict(tr1) is decreasing as the credit score increases.

```
### Now we use the trained weights and biases to try to predict based on new cases
### Increasing credit score here
tr1=sc.transform([ [1,  0, 0, 400, 1, 40, 3, 60000, 2, 1, 1, 100000],
                   [1,  0, 0, 500, 1, 40, 3, 60000, 2, 1, 1, 100000],
                   [1,  0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 100000],
                   [1,  0, 0, 700, 1, 40, 3, 60000, 2, 1, 1, 100000],
                   [1,  0, 0, 800, 1, 40, 3, 60000, 2, 1, 1, 100000]])
print(tr1)  ### tr.shape is (5,12)
### Predicting result for 5 cases/subjects
print(NNmodel.predict(tr1))
### note in each recompute -- this no. will change slightly because of the random initiation of the weights
```

```
[[0.03330604]
 [0.02859351]
 [0.0247239 ]
 [0.02155222]
 [0.01895096]]
```

It is consistent with the negative coefficient of CrScore in the logit regression. Increasing credit score, increasing tenure, lesser balance produce a lower probability of exit. The other features show some mixed results.

Homework 4 Graded Exercise:

Chapter7-9.ipynb (this file is provided only after grading)

The data set -- "Churn_Modelling.csv" – is found in public resource: https://www.kaggle. com/datasets/. The data set contains a bank's 10,000 customers' details, viz., nationality, gender, credit score (higher score means more credit-worthy), age, tenure (years with the bank), bank balance in the period of the data, the number of bank products used by the customer, whether the customer has the bank credit card, is active customer, and estimated USD salary of the customer.

These serve as 10 features of each customer that could explain the target variable "exited" – which is whether the customer exited or left the bank. The classification of "exited" is 1 if the customer left the bank. "exited" = 0 which is where the customer continues with the bank.

Homework 4 Graded Exercise:

Construct a similar MultiLayer Perceptron but with 3 hidden layers each with 6 neurons. Use batch size = 100 and # epochs = 500.

Find the training and the test prediction accuracies. (Hint: If you get accuracies below 80%, re-run as the initialized weights may be at end of uniform spectrum.)

Also check if nationality, gender, and age affected the predicted probability of exit?

End of Class