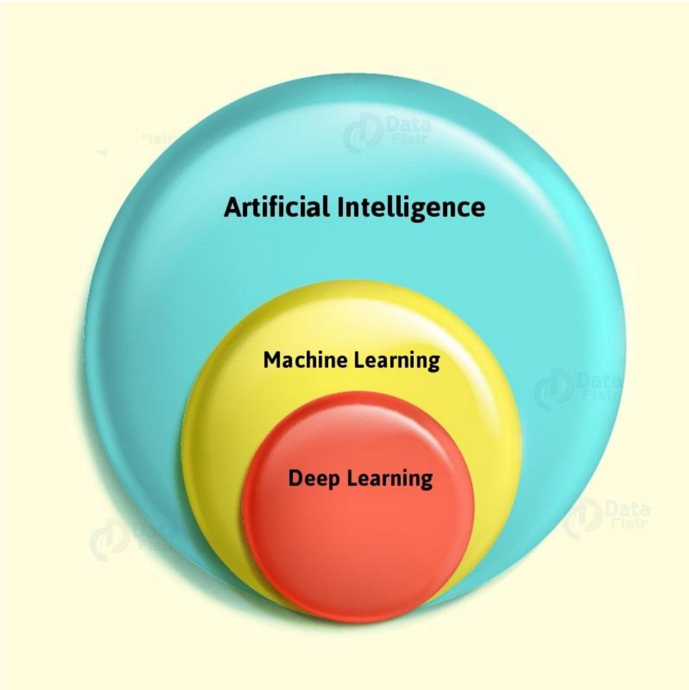
1. Deep learning – Introduction

Contents

1. Deep learning – Introduction

1. Artificial intelligence vs Machine Learning vs Deep Learning



1.1. Machine learning

- ✓ Machine learning is a part of artificial intelligence
- ✓ Machine Learning is a technique to learn from the data and apply the prediction on new data

Examples

- ✓ Amazon using machine learning to give better product choice recommendations to their costumers based on their preferences.
- ✓ Netflix uses machine learning to give better suggestions to their users of the TV series or movie or shows that they would like to watch & many more

1.2. Deep learning

- ✓ Deep learning is a subset of machine learning.
- ✓ The main difference between deep and machine learning is, machine learning models works better but the model still needs some guidance.
- ✓ If a machine learning model returns an inaccurate prediction then the programmer needs to fix that problem explicitly.
- ✓ In the case of deep learning, the model does it by itself.

Example

- ✓ Automatic car driving system is a good example of deep learning.

1.3. Artificial intelligence

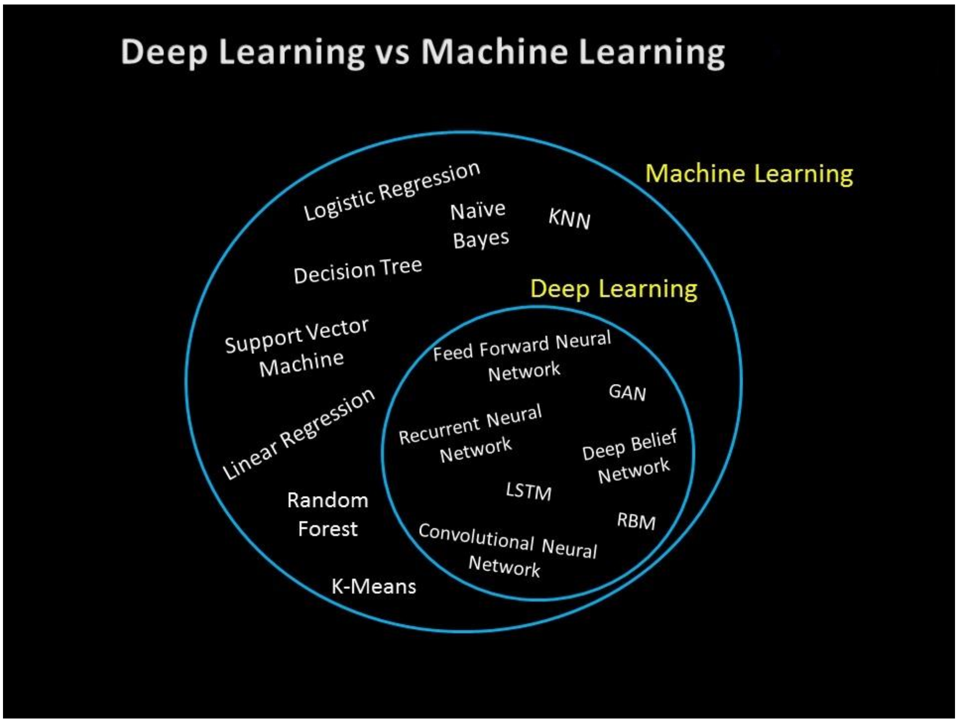
- ✓ AI means the ability of computer program to function like a human brain.
- ✓ Machine learning and deep learning are the subsets of AI
- ✓ The MOTO of AI is to replicate a human brain, the way a human brain thinks, works and functions.
- ✓ Currently AI is not yet fully implemented but we are very near to establish that too.

Example

- ✓ Sophia, the most advanced AI model present today.

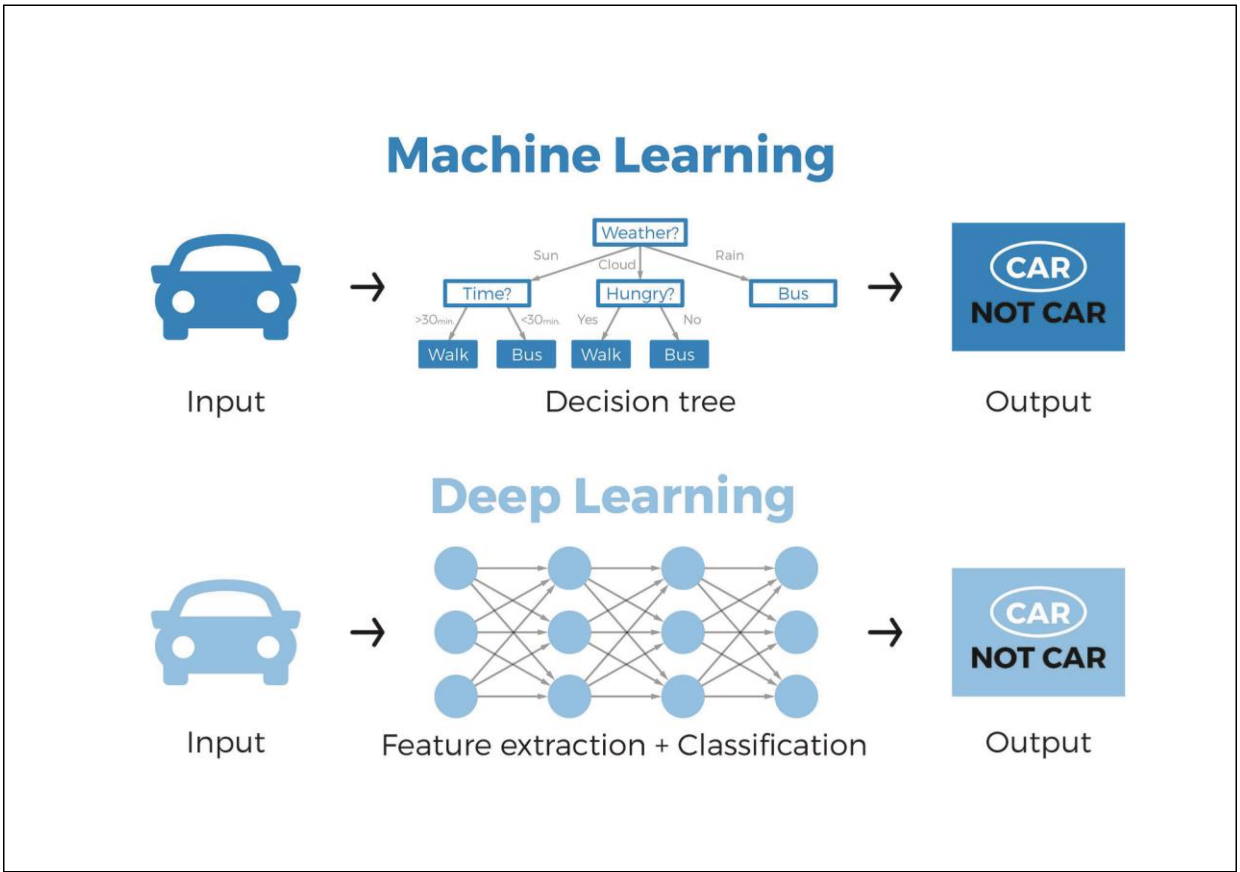
2. Different models

- ✓ In ML we have worked with different models like linear regression, logistic regression, KNN, Decision Tree & etc



3. Let's understand Deep learning

- ✓ Deep learning is a subset of machine learning mainly using in Artificial Neural Networks, which are inspired by the human brain.
- ✓ Deep learning is able to capture required features automatically and solving the problems.
- ✓ DL is the technique that comes closest to the way humans learn.
- ✓ Deep learning methods use neural network architecture.
- ✓ That is why deep learning is often referred to as "deep neural networks"



4. Differences between Deep Learning and Machine Learning

4.1. Data

- ✓ Deep learning models works with very huge data
- ✓ The more data you give to neural network, it will show much greater accuracy than other machine learning models.

4.2. Computational Hardware CPU, GPU

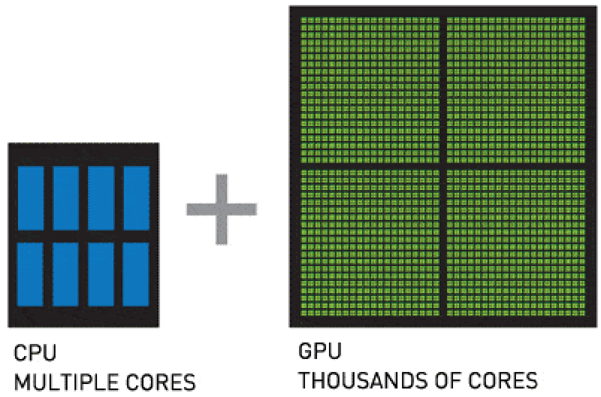
- ✓ ML models works with CPU[Central Processing Units]
- ✓ DL models works with GPU[Graphical Processing Units] and TPS[Tensor Processing Units]
    - o These are highly advanced processing systems

4.3. CPU

- ✓ It is Central Processing Unit
- ✓ It is a brain to the computer
- ✓ Processing serial instructions
- ✓ Good at speed during processing
- ✓ Consumes more memory during processing
- ✓ Companies like Intel, ARM and AMD produce the CPU and companies

4.4. GPU

- ✓ It is Graphical Processing Unit
- ✓ Processing parallel instructions
- ✓ Having very good speed compare to CPU
- ✓ Consumes very less memory
- ✓ Companies like Nvidia, AMD's ATI produce the GPU

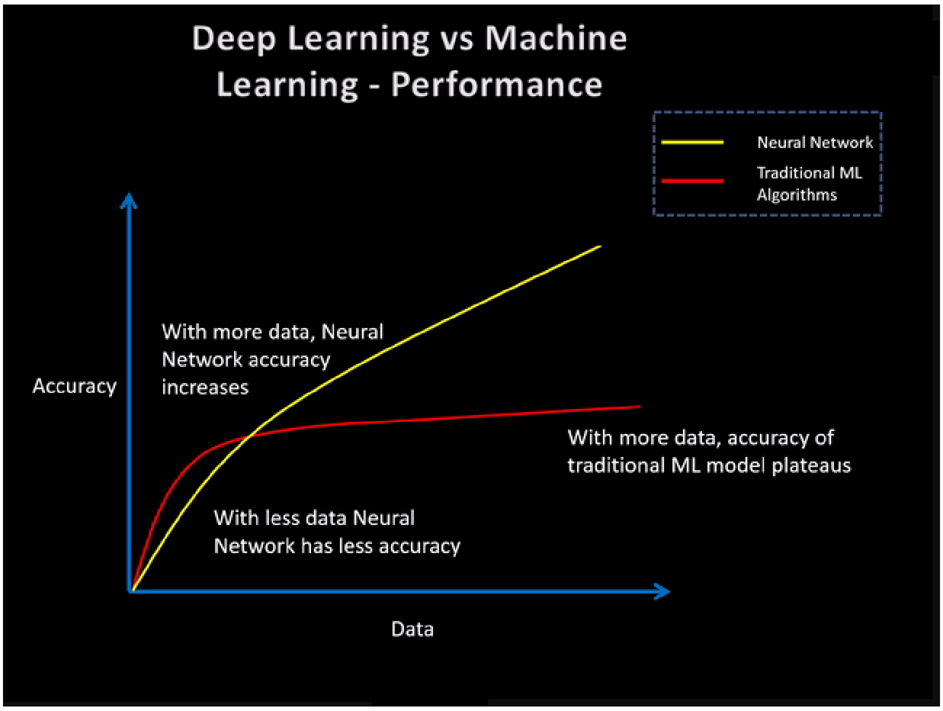


5. Feature selection

- ✓ In deep learning neural networks take care of the feature selection automatically.
- ✓ It decides a particular feature is important or not, and reduces the corresponding weights to almost zero.
- ✓ In machine learning algorithm, feature selection plays an important role and we need to do keep focus on this

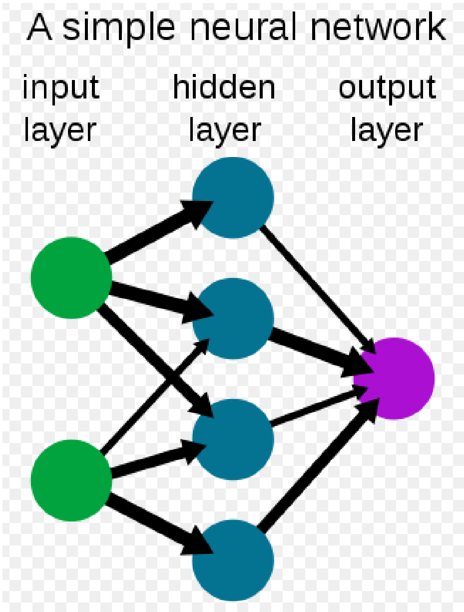
6. Performance

- ✓ Deep learning gives very good performance compare to machine learning



7. Neural networks

- ✓ Deep learning is implemented with the help of Neural Networks.
- ✓ A neural network is a network or circuit of neurons to solve the problem.
- ✓ The idea behind Neural Network is the biological neurons, which is nothing but a brain cell.
- ✓ Neural networks may have multiple hidden layers.



8. Types of neural networks

- ✓ There are mainly 3 types of neural networks in deep learning

    - Artificial Neural Networks (ANN) or Feed-Forward Neural network
    - Convolution Neural Networks (CNN)
    - Recurrent Neural Networks (RNN)

8.1. Deep learning applications

- ✓ Self Driving Cars
- ✓ Fraud News Detection
- ✓ Natural Language Processing
- ✓ Virtual Assistants
- ✓ Entertainment
- ✓ Visual Recognition
- ✓ Fraud Detection
- ✓ Healthcare
- ✓ Language Translations & etc

2. Deep Learning – Libraries

Contents

2. Deep Learning – Libraries

1. Tensorflow

- ✓ Tensorflow is a deep learning framework
- ✓ This library was created by Google in the year of 2015 and it is open source.
- ✓ TensorFlow is the most famous deep learning library.
- ✓ It is entirely based on Python programming language and use for numerical computation and data flow, which makes machine learning faster and easier.
- ✓ TensorFlow is based on graph computation

Logo

Tensorflow installation

```
pip install tensorflow
```

Update

- ✓ Tensorflow 2nd version onwards keras library was in built
- ✓ So, we no need to install keras separately

2. Pytorch

- ✓ Pytorch is a deep learning framework
- ✓ Pytorch is a deep learning library created by Facebook in the year of 2016 and it is an open source

Logo

Note

- ✓ CNTK is another deep learning framework created by Microsoft but not that much popular compare to tensorflow and pytorch

3. Keras

- ✓ Keras is an open-source high-level Neural Network library, which was written in Python, is capable enough to run on Theano, TensorFlow, or CNTK.
- ✓ It was developed by one of the Google engineers, Francois Chollet.
- ✓ It is user-friendly, extensible, and faster experimentation with deep neural networks.
- ✓ It supports Convolutional Networks and Recurrent Networks individually also their combination.

4. Steps to create deep learning models with Keras

4.1. Define your model.

- ✓ Create a Sequential model and add configured layers.

4.2. Compile your model.

- ✓ Specify loss function and optimizers and call the compile() method on the model

4.3. Fit your model.

- ✓ Train the model on a sample of data by calling the fit () method on the model.

4.4. Make predictions

- ✓ Use the model to generate predictions on new data by calling the methods such as evaluate() or predict() on the model

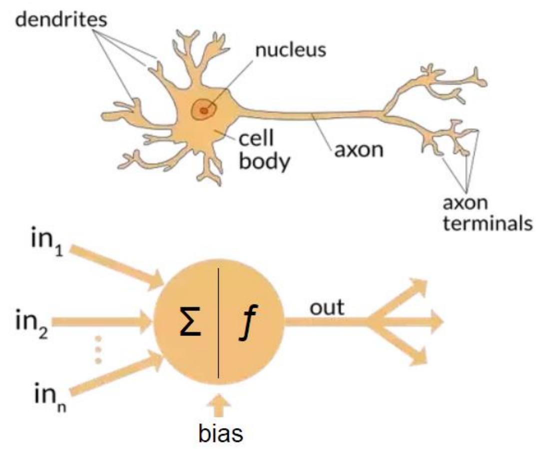
3. Deep Learning – Terminology

Contents

3. Deep Learning – Terminology

1. Neuron

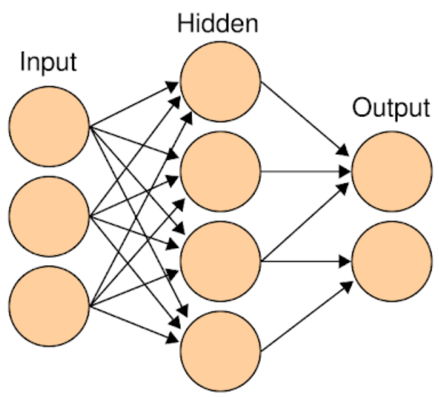
- ✓ Neuron forms the basic element of our brain.
- ✓ A group of neurons used to create neural network.
- ✓ A neuron receives an input, processes it and generates an output.



2. MLP (Multi-Layer Perceptrons)

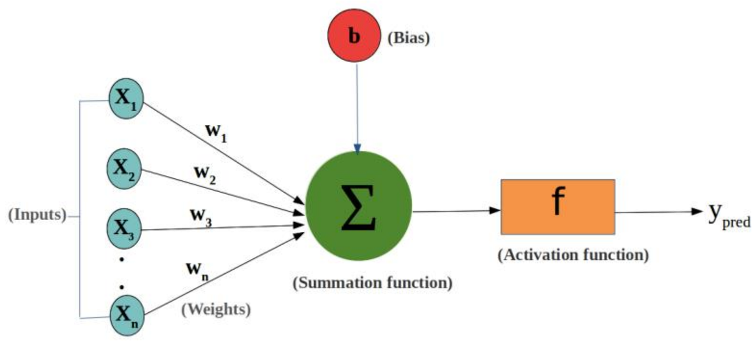
- ✓ A single neuron may not perform the complex tasks.
- ✓ So, it's required to use group of neurons to perform a complex task.
- ✓ In simple network we do have like,
    - Input layer.
    - Hidden layer.
    - Output layer.

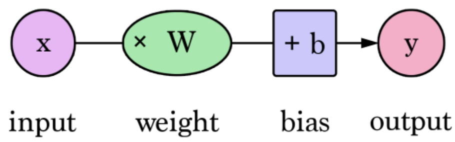
- ✓ Each layer has multiple neurons.
- ✓ All neurons in each layer are connected to all the neurons in the next layer.



3. Neural network

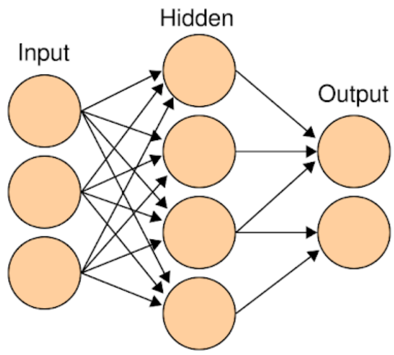
- ✓ Neural Network is the backbone of deep learning.
- ✓ A Neural Network is combinations of basic Neurons also called as Perceptrons.
- ✓ The goal of a neural network is to find the mapping function.
    - Neurons having weights and bias which is updated during training.





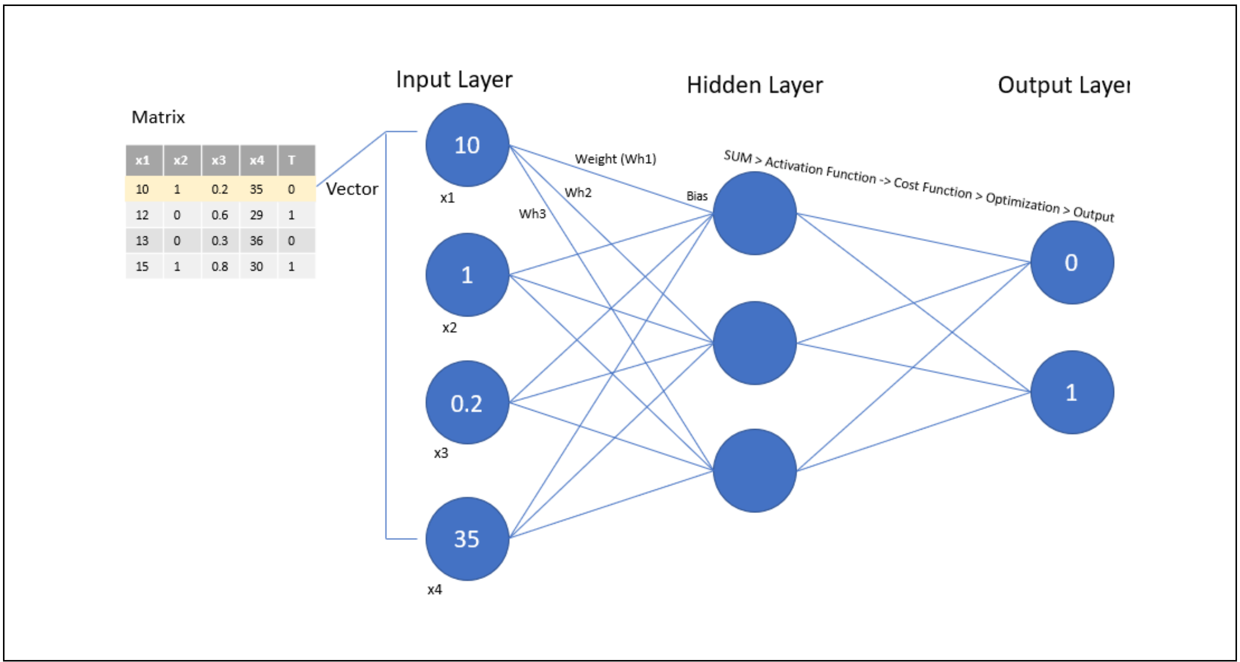
4. Input, Hidden layers & Output

- ✓ Input layer receives the input
- ✓ The processing layers are the hidden layers within the network.
  - o These layers perform specific tasks on the incoming data.
  - o These layers can pass result to the next layers
- ✓ Output layer generates the output
- ✓ Input and output layers are visible but hidden layers are hidden



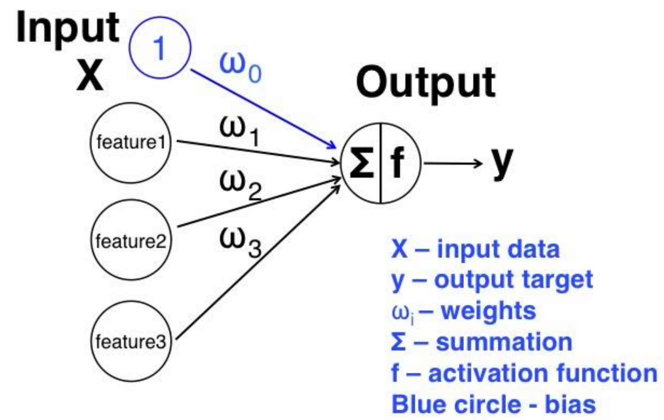
5. Weights

- ✓ When input enters into the neuron, it is multiplied by a weight.
- ✓ Assuming that, if a neuron has two inputs, then each input have separated weights.
- ✓ Here weights will be initialized randomly and these weights are updated during the model training.
    - Let's assume the input is **a** value and weight is **W1**.
    - Then after passing through the node the input becomes **a*W1**



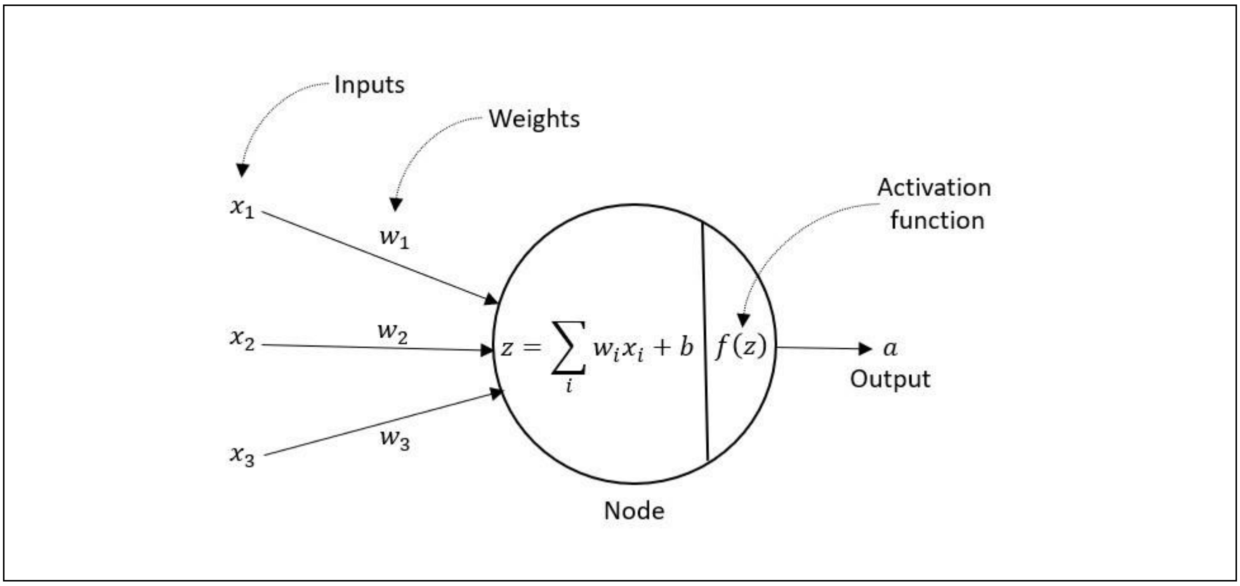
6. Bias

- ✓ In addition to the weights, bias also added to the input.
- ✓ After adding the bias, the result would look like, a * W1 + bias.



7. Activation Function

- ✓ The activation function translates the input signals to output signals.
- ✓ After applying activation function then the its looks like,
    - f(a*W1+b)
    - Here f() is the activation function.
- ✓ The activation function puts a nonlinear transformation to the linear combination



8. Types of activation functions

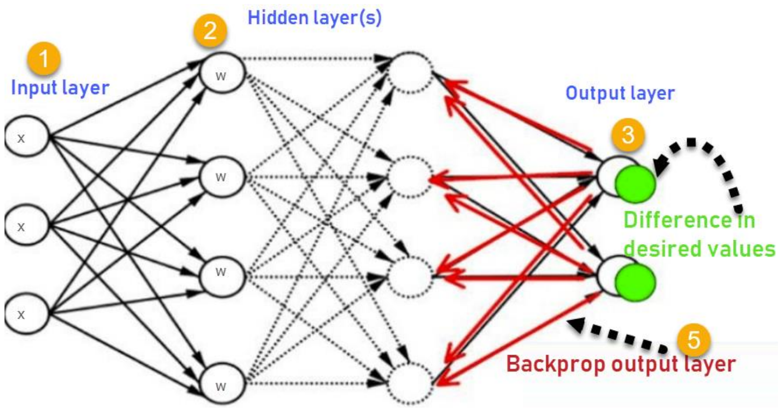
- ✓ Sigmoid
- ✓ Linear
- ✓ Tanh or hyperbolic tangent
- ✓ ReLU(Rectified Linear Units)
- ✓ Softmax

9. Forward Propagation

- ✓ In forward propagation, the information will be travelled into forward direction.
- ✓ The input layer provides input to the hidden layers and then the output is generated.
- ✓ In forward propagation input will not be travelled to backward direction.

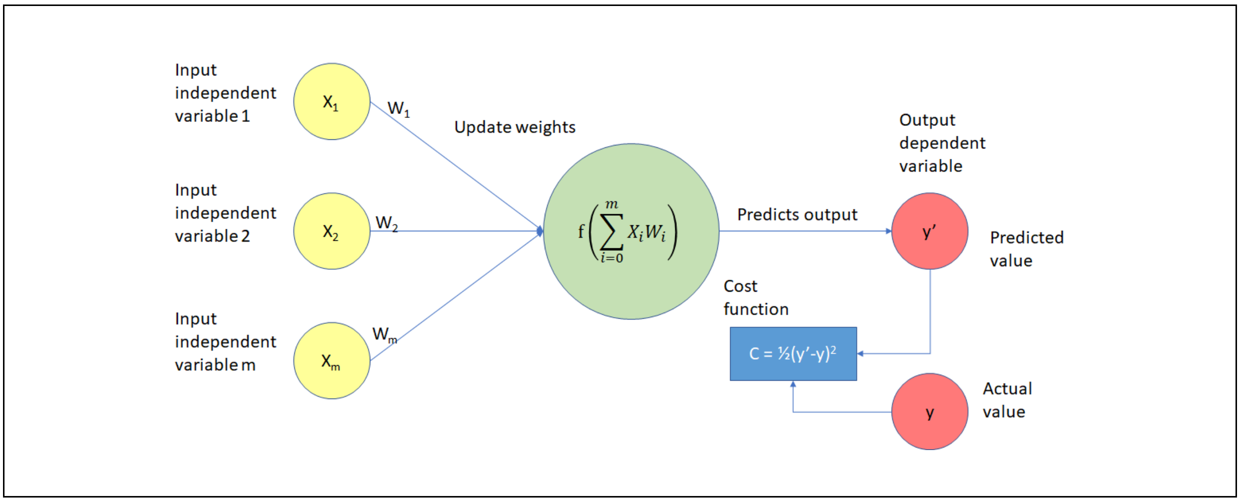
10. Back propagation

- ✓ During training, the network will get the results.
- ✓ These results will be compared with actual outputs by using loss/cost function.
- ✓ During comparing it will get error.
- ✓ To minimize this error internally weights supposed to be adjusted.
- ✓ So here back propagation helps to adjust the error.
- ✓ Back propagation means the,
    - The inputs results + error will travel in backward direction to adjust the weights



11. Cost Function

- ✓ When we create a network, the network tries to predict the output as close as possible to the actual value.
- ✓ We can measure this accuracy of the network by using the cost/loss function.
- ✓ The goal of running network is,
  - Increase our prediction accuracy
  - Reduce the error.
  - Minimizing the cost function.

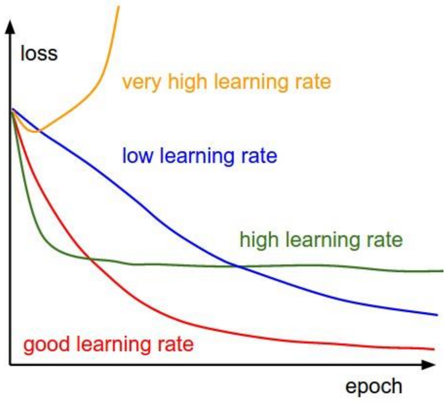


12. Gradient Descent

- ✓ Gradient descent is an optimization algorithm for minimizing the cost.

13. Learning Rate

- ✓ The learning rate is kind of hyper parameter to minimize the cost function in every iteration.
- ✓ We should choose the learning rate very carefully.
- ✓ If learning rate is large then it may miss minimum error point.
- ✓ If learning rate is very small then it takes long time to reach minimum error.
- ✓ So, optimize value is required.



14. Batches

- ✓ While training a neural network, instead of sending the entire input in onetime, generally it divides into several chunks of equal size randomly.
- ✓ It would be really good practice to train the model with batch of data instead of entire data.

15. Epochs

- ✓ The training of the neural network with all the training data for one cycle.

4. Deep Learning – Multilayer Perceptrons Models in Keras

Contents

4. Deep Learning – Multilayer Perceptrons Models in Keras

Steps to create MLP model in keras

- ✓ Model creation (Neural Network Model) by using Keras
- ✓ Compile the model
- ✓ Model training
- ✓ Model prediction

Behind the steps

- ✓ Model creation
    - Model Layers
        - Weights initialization
        - Activation function

- ✓ Compile the model.
    - Optimization
    - loss function
    - metrics

- ✓ Model training
    - Epochs
    - Batch size

- ✓ Model predictions

1. Create model (Neural Network Model) by using Keras

- ✓ Sequential is a predefined class in keras package
- ✓ By using this we can create a model.

```
from tensorflow.keras.models import Sequential

model = Sequential()
```

1.1. Model layers

- ✓ Once model created then we need to add layers to model.
    - Creating layers means, create object to Dense class
- ✓ We need to specify number of features to input layer.
    - Below example, 8 means its features

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


model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ))

model.add(layers_hidden_input)
```

1.2. Important properties to layers

- ✓ Main properties,
  - o Weight initialization.
  - o Activation functions.

1.2.1. Weight Initialization

- ✓ By using kernel_initializer
  - o random_uniform, random_normal, zero

Example

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


model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform")

model.add(layers_hidden_input)
```

1.2.2. Activation Function

- ✓ We need to use activation functions like,
  - softmax
  - rectified linear (ReLU),
  - tanh,
  - sigmoid

Example

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


model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)
```

2. Model Compilation

- ✓ Once model created then we need to compile the model (Neural Network Model).
- ✓ During this step TensorFlow converts the model into a graph so the training can be carried out efficiently.
- ✓ We can compile by calling compile() method
- ✓ Important attributes in compile() method,
    - Model optimizer
    - Loss function
    - Metrics

Example

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


model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)

model.compile(optimizer = ..., loss = ..., metrics = ...)
```

3. Model Training

- ✓ Once model created and compiled then next step is to train the model.
- ✓ We can train the model by using fit(….) method

Example

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


model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)

model.compile(optimizer = ..., loss = ..., metrics = ...)

model.fit(X, y, epochs = ..., batch_size =...)
```

4. Model Prediction

- ✓ We model training is done then we can predict with new data.
    - model.predict(X): To generate network output for the input data
    - model.evaluate(X, y): To calculate the loss values for the input data

Example

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


model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)

model.compile(optimizer = ..., loss = ..., metrics = ...)

model.fit(X, y, epochs = ..., batch_size =...)

model.predict(X)
```

5. Deep Learning – First Neural Network with Keras

Contents

5. Deep Learning – First Neural Network with Keras

1. Implementing first neural network using keras

- ✓ Importing the libraries
- ✓ Loading Dataset
- ✓ Data preparation
- ✓ Splitting the dataset
- ✓ Model creation
- ✓ Model compilation
- ✓ Model training
- ✓ Prediction

2. Dataset explanation

- ✓ We are going to work with **pima-indians-diabetes.csv**
- ✓ This Dataset is related to health care domain.
- ✓ Pima Indians are a Native American group that lives in Mexico and Arizona, USA.
- ✓ It describes patient medical record data for Pima Indians and whether they had a diabetes within five years.
- ✓ The Pima Indian Diabetes dataset consisting of Pima Indian females 21 years and older is a popular benchmark dataset.
- ✓ It is a binary classification problem (onset of diabetes as 1 or not as 0).
- ✓ All of the input variables that describe each patient are numerical.

Feature	Description	Data type	Range
Preg	Number of times pregnant	Numeric	[0, 17]
Gluc	Plasma glucose concentration at 2 Hours in an oral glucose tolerance test (GTIT)	Numeric	[0, 199]
BP	Diastolic Blood Pressure (mm Hg)	Numeric	[0, 122]
Skin	Triceps skin fold thickness (mm)	Numeric	[0, 99]
Insulin	2-Hour Serum insulin (µh/ml)	Numeric	[0, 846]
BMI	Body mass index [weight in kg/(Height in m)]	Numeric	[0, 67.1]
DPF	Diabetes pedigree function	Numeric	[0.078, 2.42]
Age	Age (years)	Numeric	[21, 81]
Outcome	Binary value indicating non-diabetic /diabetic	Factor	[0,1]

3. Input and output from the Dataset

3.1. Input Variables (X):

- ✓ 1. Number of times pregnant
- ✓ 2. Plasma glucose concentration at 2 hours in an oral glucose tolerance test
- ✓ 3. Diastolic blood pressure (mm Hg)
- ✓ 4. Triceps skin fold thickness (mm)
- ✓ 5. 2-hour serum insulin (μIU/ml)
- ✓ 6. Body mass index (weight in kg/(height in m))
- ✓ 7. Diabetes pedigree function
- ✓ 8. Age (years)

3.2. Output Variables (y):

- ✓ 1. Class variable (0 or 1)

Program	Loading csv file
Name	demo1.py
Input file	pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

print(dataset)
```

Output

```
[[  6.    148.     72.    ...   0.627  50.       1.    ]
 [  1.     85.     66.    ...   0.351  31.       0.    ]
 [  8.    183.     64.    ...   0.672  32.       1.    ]
 ...
 [  5.    121.     72.    ...   0.245  30.       0.    ]
 [  1.    126.     60.    ...   0.349  47.       1.    ]
 [  1.     93.     70.    ...   0.315  23.       0.    ]]
```

Program Name	Split into input (X) and output (y) variables demo2.py
Input file	pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Splitting data done")
```

Output

```
Splitting data done
```

4. Create the model

- ✓ First step is, we need to create the model by using Sequential class
- ✓ Once model created then we need to add layers to the model

Program	Model creation
Name	demo4.py
Input file	pima-indians-diabetes.csv

```
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

model = Sequential()

layer1 = Dense(12, input_shape = (8, ), activation = 'relu')
layer2 = Dense(8, activation = 'relu')
layer3 = Dense(1, activation = 'sigmoid')

model.add(layer1)
model.add(layer2)
model.add(layer3)

print("Model created")
```

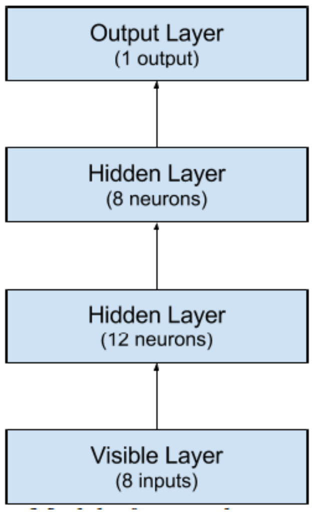
Output

```
Model created
```

4.1. Input shape and activation functions

- ✓ Fully connected layers are defined using the Dense class.
- ✓ The model expects rows of data with **8** features (the input_shape = (**8**,) argument).
- ✓ The first hidden layer has **12** nodes and uses the relu activation function.
- ✓ The second hidden layer has **8** nodes and uses the relu activation function.
- ✓ The output layer has **1** node and uses the sigmoid activation function.

- ✓ The line of code, Dense layer is doing two things, creating first hidden and input layers.



5. Compile the keras model

- ✓ Once the model created then we need to compile the model

Program	Compile the keras model
Name	demo5.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model
model = Sequential()

layer1 = Dense(12, input_shape = (8, ), activation = 'relu')
layer2 = Dense(8, activation = 'relu')
layer3 = Dense(1, activation = 'sigmoid')

model.add(layer1)
model.add(layer2)
model.add(layer3)

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

print("Model compiled")
```

Output

```
Model created
```

5.1. Loss function, optimiser and metrics

- ✓ We need to provide loss function to evaluate a set of weights.
- ✓ The optimizer is used to search through different weights for the network.
- ✓ This loss is for a binary classification problems and is defined in Keras as "binary_crossentropy".
- ✓ We have given optimizer value as adam.
    - This is more efficient gradient descent algorithm.

6. Fit the model

- ✓ Once model compiled then we need to train the model
- ✓ By using fit(…) method we can train the model.

Program	Fit the model
Name	demo6.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()


print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")



print("Step 2: Loading the dataset")

dataset = loadtxt(
   'pima-indians-diabetes.csv',
   delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]
```

```
print("Step 4: Splitting the dataset: Optional")

print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

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

print("Step 7: Model training")

model.fit(
    X,
    y,
    epochs = 150,
    batch_size = 10,
)
```

Output

```
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7211 - loss: 0.5256
Epoch 147/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 3ms/step - accuracy: 0.7631 - loss: 0.5104
Epoch 148/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7820 - loss: 0.4735
Epoch 149/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 3ms/step - accuracy: 0.7645 - loss: 0.4969
Epoch 150/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7715 - loss: 0.4861
←[1m24/24←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7010 - loss: 0.5844
```

7. Evaluate the model

- ✓ Once training is done then we need to evaluate the performance of the network.
- ✓ By using evaluate() method we can evaluate.
- ✓ This method return two values,
    - The first will be the loss of the model on the dataset.
    - The second will be the accuracy of the model on the dataset.

Program	Fit the model
Name	demo7.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
   'pima-indians-diabetes.csv',
   delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]
```

```
print("Step 4: Splitting the dataset: Optional")

print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

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

print("Step 7: Model training")

model.fit(
   X,
   y,
   epochs = 150,
   batch_size = 10,
)

# Evaluate the keras model

_, accuracy = model.evaluate(X, y)
print("Accuracy is:", accuracy*100)
```

Output

```
Epoch 147/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 1ms/step - accuracy: 0.7728 - loss: 0.4662
Epoch 148/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 1ms/step - accuracy: 0.8010 - loss: 0.4503
Epoch 149/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 1ms/step - accuracy: 0.7617 - loss: 0.4846
Epoch 150/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 1ms/step - accuracy: 0.7996 - loss: 0.4454
←[1m24/24←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 679us/step - accuracy: 0.7554 - loss: 0.4755
Accuracy is: 78.25520634651184
```

8. Prediction

- ✓ By using predict(...) method we can do the prediction.
- ✓ We are using sigmoid activation function on the output layer.
    - The predictions will be a probability in the range between 0 and 1

Program	Model prediction
Name	demo8.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

import numpy as np
from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
   'pima-indians-diabetes.csv',
   delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]
```

```
print("Step 4: Splitting the dataset: Optional")

print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

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

print("Step 7: Model training")

model.fit(
    X,
    y,
    epochs = 150,
    batch_size = 10,
)
```

```
print("Step 8: Prediction")

ip = [[6, 148, 72, 35, 0, 33.6, 0.627, 50]]

input_data = np.array(ip)
result = model.predict(input_data)

print()
print(result)
```

Output

```
Epoch 148/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 1ms/step - accuracy: 0.7216 - loss: 0.5420
Epoch 149/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7353 - loss: 0.5331
Epoch 150/150
←[1m77/77←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 1ms/step - accuracy: 0.6918 - loss: 0.5576
Step 8: Prediction
←[1m1/1←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 78ms/step

[[0.7929609]]
```

Note

- ✓ Here model done prediction for first five rows compared to the expected class value.
- ✓ You can see that most rows are correctly predicted.
- ✓ We got 79.2% accuracy with good model performance.

9. verbose = 0

✓ If we provide verbose = 0 then progress bar will be not displayed.

Program	Model prediction
Name	demo9.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
   'pima-indians-diabetes.csv',
   delimiter = ','
)

print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]


print("Step 4: Splitting the dataset: Optional")
```

```
print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

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

print("Step 7: Model training")

model.fit(
   X,
   y,
   epochs = 150,
   batch_size = 10,
   verbose = 0

)

# Evaluate the keras model

_, accuracy = model.evaluate(X, y)
print("Accuracy is:", accuracy*100)
```

Output

```
Step 1: Importing libraries
2024-08-16 12:49:00.945172: I tensorflow/
erent numerical results due to floating-p
e environment variable `TF_ENABLE_ONEDNN_
2024-08-16 12:49:02.237748: I tensorflow/
erent numerical results due to floating-p
e environment variable `TF_ENABLE_ONEDNN_
Step 2: Loading the dataset
Step 3: Data preparation
Step 4: Splitting the dataset: Optional
Step 5: Model(Neural Network) creation
Step 5.1: Creating layers and add to the
2024-08-16 12:49:05.673091: I tensorflow/
e available CPU instructions in performan
To enable the following instructions: AVX
s.
Step 6: Model compilation
Step 7: Model training
←[1m24/24←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[
Accuracy is: 69.79166865348816
```

10. Model summary

✓ By using summary method, we can check the summary of the model.

Program	Model summary
Name	demo10.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Step 4: Splitting the dataset: Optional")
```

```
print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

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

print("Step 7: Model training")

model.fit(
   X,
   y,
   epochs = 150,
   batch_size = 10,
   verbose = 0

)

# Model Summary

model.summary()
```

Output

```
Step 6: Model compilation
Step 7: Model training
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12)	108
dense_1 (Dense)	(None, 8)	104
dense_2 (Dense)	(None, 1)	9

```
Total params: 665 (2.60 KB)
Trainable params: 221 (884.00 B)
Non-trainable params: 0 (0.00 B)
Optimizer params: 444 (1.74 KB)
```

11. Image of the model

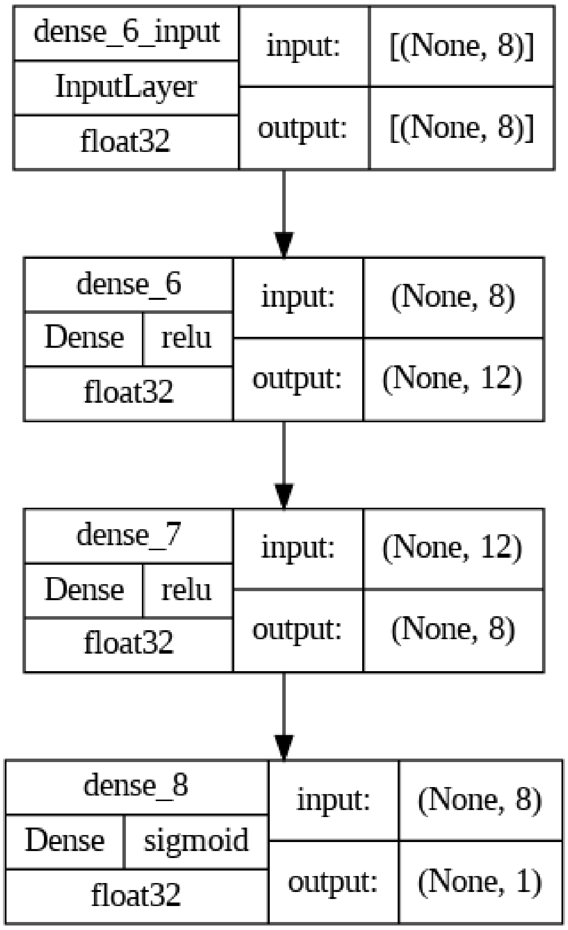
✓ We can get the image of the model

Program	Image of the model
Name	demo11.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model

model = Sequential()

layer1 = Dense(12, input_shape = (8, ), activation = 'relu')
layer2 = Dense(8, activation = 'relu')
layer3 = Dense(1, activation = 'sigmoid')

model.add(layer1)
model.add(layer2)
model.add(layer3)

# compile the keras model
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

# fit the keras model on the dataset
model.fit(X, y, epochs = 150, batch_size = 10, verbose = 0)
```

```
from tensorflow.keras.utils import plot_model

plot_model(model, to_file = 'model.png', show_shapes = True,
show_dtype = True, show_layer_names = True, expand_nested =
True, show_layer_activations = True)
```

Output



6. Deep Learning – Evaluate Model Performance

Contents

6. Deep Learning – Evaluate Model Performance

1. Model evaluation

- ✓ We can evaluate the model by using below ways,
    - Data Splitting
        - Use an automatic verification dataset
        - Use a manual verification dataset
    - Manual k-Fold Cross-Validation

2. Data splitting - Automatic Verification Dataset

- ✓ Once model compiling is done then we need to train the model.
- ✓ validation_split keyword argument helps in automation verification dataset.
- ✓ We can train the model by using fit() method
    - For fit(validation_split = 0.33) method we can provide validation_split = 0.33 as keyword argument.
- ✓ We can give 0.2 or 0.33 for 20% or 33% of your training data held back for validation.

Program	Model evaluation training
Name	demo6.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()


print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
   'pima-indians-diabetes.csv',
   delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]



print("Step 4: Splitting the dataset: Optional")
```

```
print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

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

print("Step 7: Model training")

model.fit(
   X,
   y,
   validation_split = 0.33,
   epochs = 150,
   batch_size = 10,
)
```

Output

```
Epoch 147/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7713 - loss: 0.4595 - val_accuracy:
 0.7441 - val_loss: 0.5430
Epoch 148/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7874 - loss: 0.4698 - val_accuracy:
 0.7283 - val_loss: 0.6000
Epoch 149/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7521 - loss: 0.4930 - val_accuracy:
 0.7520 - val_loss: 0.5419
Epoch 150/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7752 - loss: 0.4899 - val_accuracy:
 0.7008 - val_loss: 0.5801
```

Note

- ✓ Running the example, you can see that the verbose output on each epoch shows the loss and accuracy on both the training dataset and the validation dataset.

3. Data splitting - Use a Manual Verification Dataset

- ✓ train_test_split() function helps in manual verification of the dataset.
- ✓ Let me reminder we already worked with train_test_split() function in machine learning.

Program Model evaluation training
Name demo6.py
Input file pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split


import warnings
warnings.filterwarnings("ignore")


print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)


print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]
```

```
print("Step 4: Splitting the dataset: Optional")

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size = 0.33,
    random_state = 7
)


print("Step 5: Model(Neural Network) creation")

model = Sequential()


print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)


print("Step 6: Model compilation")

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

```
print("Step 7: Model training")

model.fit(
    X_train,
    y_train,
    validation_data = (X_test, y_test),
    epochs = 150,
    batch_size = 10
)
```

Output

```
Epoch 148/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7712 - loss: 0.4804 - val_accuracy:
 0.7283 - val_loss: 0.5412
Epoch 149/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7858 - loss: 0.4621 - val_accuracy:
 0.7638 - val_loss: 0.5658
Epoch 150/150
←[1m52/52←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 2ms/step - accuracy: 0.7714 - loss: 0.4710 - val_accuracy:
 0.7362 - val_loss: 0.5260
```

4. Manual k-Fold Cross-Validation

- ✓ By using k fold cross validation we can evaluate the model.
- ✓ Let me remind, we already worked with k fold cross validation in machine learning.
- ✓ It provides very good performance on the model.
- ✓ It splits the dataset into k subsets and used to apply training and validate on subsets.
- ✓ Finally it used to get average score from all models.

Program	K fold cross validation
Name	demo3.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import StratifiedKFold
import numpy as np

import warnings
warnings.filterwarnings("ignore")


# load the dataset
dataset = np.loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state
= 7)

cvscores = []

for train, test in kfold.split(X, y):
	# create model
	model = Sequential()

	model.add(Dense(12, input_shape=(8,), activation='relu'))
	model.add(Dense(8, activation='relu'))
	model.add(Dense(1, activation='sigmoid'))

	# Compile model
	model.compile(loss='binary_crossentropy',
```

```
            optimizer='adam', metrics = ['accuracy'])

            # Fit the model
            model.fit(X[train], y[train], epochs=150, batch_size=10,
            verbose=0)

            # evaluate the model
            scores = model.evaluate(X[test], y[test])

            print(("Accuracy:", scores[1]*100))

            cvscores.append(scores[1] * 100)

    print("Mean Accuracy", np.mean(cvscores))
```

Output

```
('Accuracy:', 59.740257263183594)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 0s/step - accuracy: 0.7113 - loss: 0.5755
('Accuracy:', 68.83116960525513)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 0s/step - accuracy: 0.7152 - loss: 0.5587
('Accuracy:', 72.72727489471436)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 8ms/step - accuracy: 0.7212 - loss: 0.6130
('Accuracy:', 72.36841917037964)
←[1m3/3←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m0s←[0m 0s/step - accuracy: 0.7644 - loss: 0.4472
('Accuracy:', 76.31579041481018)
Mean Accuracy 71.36192798614502
```

7. Deep Learning – Save the model

Contents

7. Deep Learning – Save the model

1. Save the model

- ✓ Based on requirement we can save the model
- ✓ Generally, in deep learning process,
    - o model will be saved into json file
    - o model weights will be saved into HDF5(Hierarchical Data Format)
- ✓ HDF is more convenient for storing large arrays of real values, as we have in the weights of neural networks.

Program	Save the model
Name	demo1.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.models import model_from_json

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model

model = Sequential()

model.add(Dense(12, input_shape = (8, ), activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

# compile the keras model
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics =
['accuracy'])

model.fit(X, y, epochs = 150, batch_size = 10)

model_json = model.to_json()

# saving the model
with open("model.json", "w") as json_file:
        json_file.write(model_json)

# saving the model weights
model.save_weights("model.weights.h5")

print("Saved model to disk")

print("Done")
```

Output

```
Epoch 147/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5480 - accuracy: 0.7201
Epoch 148/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5500 - accuracy: 0.7227
Epoch 149/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5493 - accuracy: 0.7292
Epoch 150/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5509 - accuracy: 0.7279
Saved model to disk
Done
```

model.json

- ✓ The json format of the model looks like the following:


```
{
 "class_name": "Sequential",
 "config": {
  "name": "sequential",
  "layers": [
   {
    "class_name": "InputLayer",
    "config": {
     "batch_input_shape": [
      null,
      8
     ],
     "dtype": "float32",
     "sparse": false,
     "ragged": false,
     "name": "dense_input"
    }
   },
   {
    "class_name": "Dense",
    "config": {
     "name": "dense",
     "trainable": true,
     "batch_input_shape": [
      null,
      8
     ],
     "dtype": "float32",
     "units": 12,
```

```
        "activation": "relu",
        "use_bias": true,
        "kernel_initializer": {
          "class_name": "GlorotUniform",
          "config": {
            "seed": null
          }
        },
        "bias_initializer": {
          "class_name": "Zeros",
          "config": {}
        },
        "kernel_regularizer": null,
        "bias_regularizer": null,
        "activity_regularizer": null,
        "kernel_constraint": null,
        "bias_constraint": null
      }
    },
    {
      "class_name": "Dense",
      "config": {
        "name": "dense_1",
        "trainable": true,
        "dtype": "float32",
        "units": 8,
        "activation": "relu",
        "use_bias": true,
        "kernel_initializer": {
          "class_name": "GlorotUniform",
          "config": {
            "seed": null
          }
        },
        "bias_initializer": {
          "class_name": "Zeros",
          "config": {}
        },
        "kernel_regularizer": null,
        "bias_regularizer": null,
        "activity_regularizer": null,
        "kernel_constraint": null,
        "bias_constraint": null
      }
    },
    {
      "class_name": "Dense",
```

```
        "config": {
          "name": "dense_2",
          "trainable": true,
          "dtype": "float32",
          "units": 1,
          "activation": "sigmoid",
          "use_bias": true,
          "kernel_initializer": {
            "class_name": "GlorotUniform",
            "config": {
              "seed": null
            }
          },
          "bias_initializer": {
            "class_name": "Zeros",
            "config": {}
          },
          "kernel_regularizer": null,
          "bias_regularizer": null,
          "activity_regularizer": null,
          "kernel_constraint": null,
          "bias_constraint": null
        }
      }
    ]
  },
  "keras_version": "2.8.0",
  "backend": "tensorflow"
}
```

Program Name	Loading the model from json file demo2.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from keras.models import model_from_json
import numpy as np

# load the dataset
dataset = np.loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

json_file = open('model.json' , 'r')

model_j = json_file.read()
model = model_from_json(model_j)
model.load_weights("model.weights.h5")
print("Loaded model from disk")

model.compile(loss = 'binary_crossentropy' , optimizer = 'rmsprop' ,
metrics=['accuracy'])
score = model.evaluate(X, y)

print(score)
```

Output

```
Loaded model from disk
24/24 [==============================] - 0s 924us/step - loss: 0.4692 - accuracy: 0.782
[0.46923649311065674, 0.7825520634651184]
```

8. Deep Learning – Best Model with Check Pointing

Contents

8. Deep Learning – Best Model with Check Point

1. Check point

- ✓ Deep learning models can take hours, days or even weeks to train and if a training run is stopped unexpectedly, you can lose a lot of work.
- ✓ Now we need to learn how to how we can checkpoint the deep learning models during training.
- ✓ The checkpoint captures the weights of the model.
- ✓ These weights can be used to make predictions for ongoing training.

Program	Checkpoint Neural Network Model Improvements
Name	demo1.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.callbacks import ModelCheckpoint

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model

model = Sequential()

model.add(Dense(12, input_shape = (8, ), activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

# compile the keras model
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics =
['accuracy'])


# checkpoint
filepath = "weights-improvement-{epoch:02d}-{val_accuracy:.2f}.keras"

checkpoint = ModelCheckpoint(filepath, monitor= 'val_accuracy' , verbose=1,
save_best_only = True, mode= 'max' )

callbacks_list = [checkpoint]

model.fit(X, y, validation_split=0.33, epochs=150, batch_size=10, callbacks =
callbacks_list, verbose=0)

print("Done")
```

Output

```
………………………
………………………
Epoch 102: val_accuracy did not improve from 0.74016
Epoch 103: val_accuracy did not improve from 0.74016

Epoch 104: val_accuracy improved from 0.74016 to 0.74803, saving model
to weights-improvement-104-0.75.hdf5

Epoch 105: val_accuracy did not improve from 0.74803

Epoch 106: val_accuracy did not improve from 0.74803

Epoch 107: val_accuracy did not improve from 0.74803

Epoch 108: val_accuracy did not improve from 0.74803
```

9. Deep Learning – Visualize model training history

Contents

9. Deep Learning – Visualize model training history

1. Visualize the model accuracy and loss

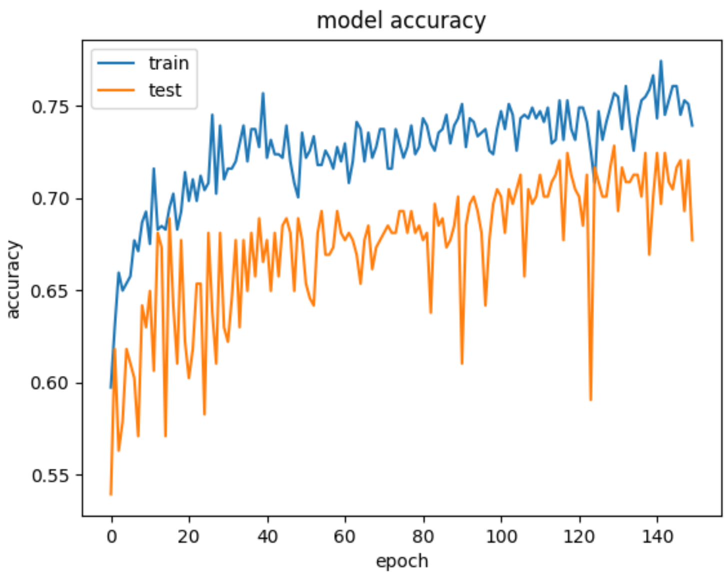
- ✓ We can create plots from the collected history data.
- ✓ We can plot the neural network to model for the Pima Indians onset of diabetes binary classification problem
- ✓ The example collects the history and create two charts
    - A plot of accuracy on the training and validation datasets over training epochs
    - A plot of loss on the training and validation datasets over training epochs

Program	Visualize model training history
Name	demo1.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np

# load pima Indians dataset
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")

# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]

# create model
model = Sequential()

model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Fit the model
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10,
verbose=0)

# list all data in history
print(history.history.keys())

# summarize history for ACCURACY
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')
```

```
plt.show()


# summarize history for LOSS
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')
plt.show()

print("Done")
```

Output




model loss
train
test
1.6
1.4
1.2
1.0
0.8
0.6
loss
0
20
40
60
80
100
120
140
epoch