```
!pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.
```

```
#Tensor Flow
import tensorflow as tf
import numpy as np

# Generate random input data
x = np.random.rand(100).astype(np.float32)
print("Input data (x):", x)

# Define the expected output (observed output)
y = x * 0.2 + 0.2

# Initialize weight and bias as TensorFlow variables
W = tf.Variable(tf.random.normal([1]), dtype=tf.float32)
b = tf.Variable(tf.zeros([1]), dtype=tf.float32)

# Define the mean squared error loss function
def mse_loss():
    y_pred = W * x + b
    loss = tf.reduce_mean(tf.square(y_pred - y))
    return loss

# Initialize the Adam optimizer
optimizer = tf.keras.optimizers.Adam()

for step in range(5000):
    with tf.GradientTape() as tape:
        # Record the forward pass to compute gradients
        loss = mse_loss()

    # Compute gradients
    gradients = tape.gradient(loss, [W, b])

    # Update weights and bias using the optimizer
    optimizer.apply_gradients(zip(gradients, [W, b]))

    if step % 500 == 0:
        # Print current values of W, b, and loss
        current_loss = mse_loss().numpy()
        print(f"Step {step}: W = {W.numpy()}, b = {b.numpy()}, Loss = {current_loss}")
```

```
Input data (x): [0.51037544 0.720212   0.30098298 0.2948662  0.09435506 0.15432854
 0.5934166  0.0738846  0.7458324  0.8412049  0.6217775  0.40883863
 0.75456876 0.56331754 0.6322214  0.81396973 0.08257426 0.6841585
 0.3041146  0.49055675 0.13091464 0.31847522 0.03032102 0.15614218
 0.36965096 0.19662744 0.6770544  0.71373945 0.4826075  0.01518037
 0.557871   0.16785221 0.63037205 0.6543851  0.55686384 0.17839725
 0.3379201  0.42592633 0.6293109  0.03527107 0.6165651  0.38065234
 0.18477175 0.87880427 0.5552781  0.84820366 0.3940719  0.06764966
 0.05110518 0.46766233 0.9527046  0.9039394  0.8361851  0.76819533
```

```
 0.32328105 0.97472024 0.58391833 0.19140992 0.40564215 0.41284883
 0.1519453  0.42220908 0.8389985  0.16241962 0.4154388  0.63810855
 0.6835967  0.08688549 0.77943367 0.4103383  0.1860193  0.9557191
 0.31993428 0.9631959  0.81115276 0.1627856  0.03839776 0.3139801
 0.00919499 0.04126265 0.21253552 0.41979152 0.99156904 0.9962813
 0.9720236  0.2531727  0.6337259  0.5880106  0.4540614  0.8828824
 0.55790275 0.3925333  0.5234228  0.97368306 0.6473126  0.44439787
 0.5885781  0.322056   0.9378256  0.40221557]
Step 0: W = [-1.5380969], b = [0.00099999], Loss = 1.3232738971710205
Step 500: W = [-1.1136128], b = [0.41038737], Loss = 0.3193606436252594
Step 1000: W = [-0.8461263], b = [0.6120857], Loss = 0.09699486941099167
Step 1500: W = [-0.68494165], b = [0.64349794], Loss = 0.06339596956968307
Step 2000: W = [-0.5560062], b = [0.59637624], Loss = 0.04704602062702179
Step 2500: W = [-0.42382202], b = [0.5282534], Loss = 0.03209450840950012
Step 3000: W = [-0.28716764], b = [0.45614326], Loss = 0.01956496201455593
Step 3500: W = [-0.15471043], b = [0.38633186], Loss = 0.010367147624492645
Step 4000: W = [-0.03630579], b = [0.32404494], Loss = 0.00459934351965785
Step 4500: W = [0.05957086], b = [0.2736821], Loss = 0.001623887219466269
```

```
#Keras

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

# Load data while skipping the header row
## INPUT Variables ##
# x1 - Number of times pregnant
# x2 - plasma glucose
# x3 - diastolic blood pressure
# x4 - Triceps skin fold thickness
# x5 - 2-hour serum insulin
# x6 - bmi
# x7 - diabetes pedigree function
# x8 - age (yrs)
## Output Variable ##
# Class Variable - 0 or 1
dataset = np.loadtxt('diabetes.csv', delimiter=',', skiprows=1)

dataset
```

```
array([[  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.   ]])
```

```
# [:,:] - first : is range of rows and second : is columns
# [start:end] - begins at start, ends at end-1
x = dataset[:,0:8]
print(type(x))
print(x.shape)
print("\n")
y = dataset[:,8]
print(y)
```

```
<class 'numpy.ndarray'>
(768, 8)


[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1.
 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0.
 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0.
 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0.
 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0.
 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0.
 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1.
 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0.
 1. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0.
 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1.
 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 0.
 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0.
 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0.
 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1.
 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1.
 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0.]
```

```
# Step 2 - Creating or define the Keras Model
# Sequential Model
# Layer1 -> Layer2 -> Layer3
model = Sequential()
```

```
# The model expects row of data with 8 variables
# 12 = nodes
model.add(Dense(12, input_shape=(8,), activation='relu'))
# Hidden Layer
# 8 = nodes
model.add(Dense(8, activation='relu'))
# Output layer
model.add(Dense(1,activation='sigmoid'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Step 3 - Compile the Keras model
# loss (error)
# optimizer (adam)
# metrics = accuracy
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
#Step 4 - Fit / Train the model
#1 = Epochs - number of iterations / passes
#2 - Batch - sample data
model.fit(x,y, epochs=150, batch_size=10)
# Step 5 - evaluate the model
```

```
Epoch 1/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3322 - loss: 11.2279
Epoch 2/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4156 - loss: 1.3248
Epoch 3/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5310 - loss: 0.8013
Epoch 4/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5742 - loss: 0.7500
Epoch 5/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6074 - loss: 0.7014
Epoch 6/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6069 - loss: 0.6775
Epoch 7/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6376 - loss: 0.6888
Epoch 8/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6402 - loss: 0.6756
Epoch 9/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6251 - loss: 0.6861
Epoch 10/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6597 - loss: 0.6349
Epoch 11/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6682 - loss: 0.6370
Epoch 12/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6940 - loss: 0.6085
Epoch 13/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6594 - loss: 0.6303
Epoch 14/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6727 - loss: 0.6065
Epoch 15/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6732 - loss: 0.6099
Epoch 16/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6661 - loss: 0.6028
Epoch 17/150
```

```
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6889 - loss: 0.6085
Epoch 18/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6864 - loss: 0.5805
Epoch 19/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6934 - loss: 0.5715
Epoch 20/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6802 - loss: 0.5921
Epoch 21/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6848 - loss: 0.5922
Epoch 22/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6927 - loss: 0.5956
Epoch 23/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6832 - loss: 0.5877
Epoch 24/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6834 - loss: 0.5916
Epoch 25/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6640 - loss: 0.5884
Epoch 26/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7083 - loss: 0.5720
Epoch 27/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7008 - loss: 0.5807
Epoch 28/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6554 - loss: 0.6225
Epoch 29/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6809 - loss: 0.5923
```

```
model.evaluate(x,y)
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6723 - loss: 0.5510
[0.5226908922195435, 0.7057291865348816]
```

```
#Theano
!pip install theano
```

```
Collecting theano
  Downloading Theano-1.0.5.tar.gz (2.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.8/2.8 MB 24.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-packages (
Building wheels for collected packages: theano
  Building wheel for theano (setup.py) ... done
  Created wheel for theano: filename=Theano-1.0.5-py3-none-any.whl size=2668109 sha256
  Stored in directory: /root/.cache/pip/wheels/d9/e6/7d/2267d21a99e4ab8276f976f293b4ff
Successfully built theano
Installing collected packages: theano
Successfully installed theano-1.0.5
```

```
import theano
from theano import *
import theano.tensor as T
import numpy as np
import pandas as pd
from theano import function
# scalar variables
v1 = T.dscalar()
v2 = T.scalar()
```

```
# subtraction
sres = v1-v2
#add
ares = v1+v2
#convert the results into functions
calcsres = theano.function([v1,v2],sres)
calcares = theano.function([v1,v2],ares)
calcares(12,23)
calcsres(13,12)
x = T.dmatrix('x')
y = T.dmatrix('y')
# addition
z = x+y
func = function([x,y],z)
m1 = [
[1,2],
[3,4]
]
m2 = [
[4,5],
[6,7]
]
func(m1,m2)
# element wise sum
# 00 -> [1+4] -> 5
```

```
/usr/local/lib/python3.10/dist-packages/theano/scalar/basic.py:2323: FutureWarning: Ir
  self.ctor = getattr(np, o_type.dtype)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-18-8a274cf5f21f> in <cell line: 1>()
----> 1 import theano
      2 from theano import *
      3 import theano.tensor as T
      4 import numpy as np
      5 import pandas as pd

8 frames
/usr/local/lib/python3.10/dist-packages/numpy/__init__.py in __getattr__(attr)
    322
    323         if attr in __former_attrs__:
--> 324             raise AttributeError(__former_attrs__[attr])
    325
    326         if attr == 'testing':

AttributeError: module 'numpy' has no attribute 'bool'.
`np.bool` was a deprecated alias for the builtin `bool`. To avoid this error in
existing code, use `bool` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
The aliases was originally deprecated in NumPy 1.20; for more details and guidance
```

```
#torch
import torch
import numpy as np
# Tensor Initialization
### multiple ways
### 1 - using data
data = [
[1,2],
[3,4]
]
x_data = torch.tensor(data)
print(type(x_data))
```

```
<class 'torch.Tensor'>
```

```
### 2 - using numpy array
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
print(x_np)
print(type(x_np))
### 3 - using another tensor
x_ones = torch.ones_like(x_data)
print("One Tensor: \n",x_ones)
x_rand = torch.rand_like(x_data,dtype=torch.float)
print(x_rand)
#### more ways to create tensors
shape = (2,3)
random_tensor = torch.rand(shape)
print(random_tensor)
print(type(random_tensor))
ones_tensor = torch.ones(shape)
print(ones_tensor)
print(type(ones_tensor))
zeros_tensor = torch.zeros(shape)
print(zeros_tensor)
print(type(zeros_tensor))
tensor = torch.rand(3,4)
print(tensor)
tensor.shape
tensor.dtype
tensor.device
# Tensor Operations
if torch.cuda.is_available():
    tensor = tensor.to('cuda')
    print("Device tensor is stored in ", tensor.device)
# Indexing, Slicing
tensor = torch.ones(4,4)


print(tensor)
print(tensor)
tensor1 = torch.zeros(4,4)
print(tensor1)
tensor2 = torch.cat([tensor,tensor1])
print(tensor2)
# Multiply Operation
tensor.mul(tensor1)
tensor * tensor1
tensor.T
# inplace - change the original tensor
tensor.add_(3)
print(tensor)
# from tensor to numpy
t = torch.ones(5)
print(t)
n = t.numpy()
print(n)
print(type(n))
```

```
tensor([[1, 2],
        [3, 4]])
<class 'torch.Tensor'>
```

```
One Tensor:
```

Start coding or generate with AI.

```
tensor([[0.1650, 0.8264],
        [0.5364, 0.1722]])
tensor([[0.3756, 0.6627, 0.1608],
        [0.4196, 0.7394, 0.8060]])
<class 'torch.Tensor'>
tensor([[1., 1., 1.],
```