```
!pip install numpy

```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in ./.local/lib/python3.10/site-packages (1
```

```python
#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]))
b = tf.Variable(tf.zeros([1]))

# 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):
    # Use optimizer.minimize directly with mse_loss
    optimizer.minimize(mse_loss, var_list=[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): [0.72518516 0.33169788 0.49253738 0.7980687  0.3500872  0.325642
 0.83675    0.98803276 0.1678318  0.77563596 0.14135621 0.6014929
 0.16336598 0.66477495 0.13349536 0.4905786  0.93968487 0.74022484
 0.98231655 0.85630727 0.10832536 0.99236447 0.97800434 0.45206955
 0.6735295  0.49467528 0.01041481 0.22491108 0.739951   0.214348
 0.24958377 0.42658728 0.1820286  0.6376425  0.5754288  0.739015
 0.02902341 0.43740627 0.8978234  0.45345494 0.13880637 0.39580518
 0.72545844 0.6499103  0.10173661 0.64434576 0.6747472  0.5548801
 0.9365164  0.9802911  0.00840634 0.9479032  0.27406564 0.99195105
 0.8794225  0.7173347  0.60858935 0.8839874  0.5271131  0.7728518
 0.40600458 0.9285557  0.92342746 0.6476055  0.26975036 0.40166393
 0.77914923 0.21388817 0.0593919  0.59629405 0.6311362  0.4893019
 0.6793278  0.9212634  0.4635738  0.24637969 0.46619758 0.8605006
 0.3718864  0.458607   0.870112   0.33309686 0.7871344  0.6409242
 0.69143593 0.27031332 0.5176901  0.7108124  0.94753516 0.5215696
 0.5884546  0.19869195 0.60252446 0.11830225 0.24734907 0.32914996
 0.76248324 0.23963624 0.36556277 0.46030986]
Step 0: W = [-1.0325525], b = [0.00099999], Loss = 0.8743147850036621
Step 500: W = [-0.6334619], b = [0.38571402], Loss = 0.12545545399188995
Step 1000: W = [-0.43490958], b = [0.521493], Loss = 0.031669389456510544
Step 1500: W = [-0.33489054], b = [0.5108884], Loss = 0.022445760667324066
Step 2000: W = [-0.24655704], b = [0.46246243], Loss = 0.01574840396642685
Step 2500: W = [-0.15308326], b = [0.407519], Loss = 0.009845343418419361
Step 3000: W = [-0.05987181], b = [0.3526833], Loss = 0.005332118831574917
Step 3500: W = [0.02527468], b = [0.30262804], Loss = 0.002409989247098565
Step 4000: W = [0.09533903], b = [0.26146296], Loss = 0.0008646114729344845
Step 4500: W = [0.1459732], b = [0.23172458], Loss = 0.00023037918435875326
```

```python
#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.   ]])
```

```python
# [:,:] - 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. 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.]
```

```python
# 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'))
```

```
# 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
```

```
77/77 [==============================] - 0s 626us/step - loss: 0.5075 - accura
Epoch 140/150
77/77 [==============================] - 0s 621us/step - loss: 0.5069 - accura
Epoch 141/150
77/77 [==============================] - 0s 630us/step - loss: 0.5208 - accura
Epoch 142/150
77/77 [==============================] - 0s 687us/step - loss: 0.5067 - accura
Epoch 143/150
77/77 [==============================] - 0s 659us/step - loss: 0.5012 - accura
Epoch 144/150
77/77 [==============================] - 0s 626us/step - loss: 0.5089 - accura
Epoch 145/150
77/77 [==============================] - 0s 646us/step - loss: 0.4984 - accura
Epoch 146/150
77/77 [==============================] - 0s 624us/step - loss: 0.5100 - accura
Epoch 147/150
77/77 [==============================] - 0s 624us/step - loss: 0.4972 - accura
Epoch 148/150
77/77 [==============================] - 0s 628us/step - loss: 0.4985 - accura
Epoch 149/150
77/77 [==============================] - 0s 620us/step - loss: 0.5046 - accura
Epoch 150/150
77/77 [==============================] - 0s 626us/step - loss: 0.5011 - accura
<keras.src.callbacks.History at 0x7fb4189da890>
```

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

```
24/24 [==============================] - 0s 808us/step - loss: 0.4854 - accuracy
[0.4853849411010742, 0.7708333134651184]
```

```python
!pip install theano
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: theano in ./.local/lib/python3.10/site-packages (
Requirement already satisfied: six>=1.9.0 in /usr/lib/python3/dist-packages (fro
Requirement already satisfied: scipy>=0.14 in ./.local/lib/python3.10/site-packa
Requirement already satisfied: numpy>=1.9.1 in ./.local/lib/python3.10/site-pack
```

```python
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'>
tensor([[1, 2],
        [3, 4]])
<class 'torch.Tensor'>
One Tensor:
 tensor([[1, 1],
        [1, 1]])
tensor([[0.9257, 0.9564],
        [0.1651, 0.7169]])
tensor([[0.7727, 0.1777, 0.6461],
        [0.0969, 0.1270, 0.0898]])
<class 'torch.Tensor'>
tensor([[1., 1., 1.],
        [1., 1., 1.]])
<class 'torch.Tensor'>
tensor([[0., 0., 0.],
        [0., 0., 0.]])
<class 'torch.Tensor'>
tensor([[0.0230, 0.1503, 0.0781, 0.7515],
        [0.4377, 0.7905, 0.2401, 0.0091],
        [0.8236, 0.6090, 0.9742, 0.1531]])
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
tensor([[4., 4., 4., 4.],
        [4., 4., 4., 4.],
        [4., 4., 4., 4.],
        [4., 4., 4., 4.]])
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
<class 'numpy.ndarray'>
```

```
### 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:
 tensor([[1, 1],
        [1, 1]])
tensor([[0.1032, 0.2760],
        [0.4814, 0.4485]])
tensor([[0.6491, 0.0962, 0.0525],
        [0.9425, 0.9621, 0.8840]])
<class 'torch.Tensor'>
tensor([[1., 1., 1.],
        [1., 1., 1.]])
<class 'torch.Tensor'>
tensor([[0., 0., 0.],
        [0., 0., 0.]])
<class 'torch.Tensor'>
tensor([[0.5538, 0.9957, 0.9594, 0.2374],
        [0.3994, 0.2917, 0.9551, 0.4998],
        [0.1529, 0.9141, 0.3315, 0.8212]])
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
tensor([[4., 4., 4., 4.],
        [4., 4., 4., 4.]
```

Start coding or generate with AI.

```
        [illegible]
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
<class 'numpy.ndarray'>
```