Write a Program tutorial on tensorflow (XOR, AND)

Lab_Internal_1 _VDSS09_Abhishek_122021601009

LAB INTERNAL 1

REGN NO: 122021601009

NAME: ALLENA VENKATA SAI ABHISHEK

CAMPUS: VIZAG

MTECH DSS

NUMPY : NumPy is a Python library used for working with arrays. It has functions for working in domain of linear algebra, fourier transform, and matrices.

TENSORFLOW :

1. It's an open source artificial intelligence library, that uses the data flow graphs, to build models.
2. It helps to create large-scale neural networks with many layers.
3. It is mainly used for:

- Classification,
- Perception,
- Understanding,
- Discovering,
- Prediction
- Creation

KERAS:

1. It offers consistent & simple APIs designed for human beings, not machines.
2. It helps in reducing cognitive load.
3. It minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

AND

```
#we import libraries
import numpy as np
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense

#Dense layer is the regular deeply connected neural network layer.
#It does the below operation on the input and return the output.

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
# numpy array for training data

target_data = np.array([[0],[0],[0],[1]], "float32")
# numpy array for target data

model = Sequential()
#A Sequential model is appropriate for a plain stack of layers where
#each layer has exactly one input tensor and one output tensor & is not
#appropriate when our model has multiple inputs or multiple outputs.
#Any of your layers has multiple inputs or multiple outputs

model.add(Dense(16, input_dim=2, activation='relu'))
# we use activation function RELU, and has 16 nodes, 2D input dimension

model.add(Dense(1, activation='sigmoid'))
#we add output layer with activation function of sigmoid and has 1 output node

model.compile(loss='mean_squared_error',optimizer='adam',metrics=['binary_accuracy'])
#then we compile the model, we use adam optimiser for

model.fit(training_data, target_data, epochs=1000)
#then we fit the model with 1000 epochs and we give the arguments as
# training and target data


scores = model.evaluate(training_data, target_data)
# then we evaluate the model and find the scores


print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
# then we find the binary accuracy which was our metrices

print (model.predict(training_data).round())
#AND GATE
```

```
1/1 [==============================] - 0s 9ms/step - loss: 0.0108 - binary_accuracy
Epoch 976/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0108 - binary_accuracy
Epoch 977/1000
1/1 [==============================] - 0s 10ms/step - loss: 0.0108 - binary_accura
Epoch 978/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.0108 - binary_accuracy
Epoch 979/1000
1/1 [==============================] - 0s 11ms/step - loss: 0.0107 - binary_accura
Epoch 980/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0107 - binary_accuracy
Epoch 981/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0107 - binary_accuracy
Epoch 982/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0106 - binary_accuracy
Epoch 983/1000
1/1 [==============================] - 0s 12ms/step - loss: 0.0106 - binary_accura
```

```
Epoch 984/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0106 - binary_accuracy
Epoch 985/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0106 - binary_accuracy
Epoch 986/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0105 - binary_accuracy
Epoch 987/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.0105 - binary_accuracy
Epoch 988/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0105 - binary_accuracy
Epoch 989/1000
1/1 [==============================] - 0s 10ms/step - loss: 0.0105 - binary_accura
Epoch 990/1000
1/1 [==============================] - 0s 16ms/step - loss: 0.0104 - binary_accura
Epoch 991/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0104 - binary_accuracy
Epoch 992/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.0104 - binary_accuracy
Epoch 993/1000
1/1 [==============================] - 0s 10ms/step - loss: 0.0103 - binary_accura
Epoch 994/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0103 - binary_accuracy
Epoch 995/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.0103 - binary_accuracy
Epoch 996/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.0103 - binary_accuracy
Epoch 997/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0102 - binary_accuracy
Epoch 998/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0102 - binary_accuracy
Epoch 999/1000
1/1 [==============================] - 0s 9ms/step - loss: 0.0102 - binary_accuracy
Epoch 1000/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0102 - binary_accuracy
1/1 [==============================] - 0s 159ms/step - loss: 0.0101 - binary_accur

binary_accuracy: 100.00%
[[0.]
 [0.]
 [0.]
 [1.]]
```

XOR Gate

```
#importing the libraries
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense

#Dense layer is the regular deeply connected neural network layer.
#It does the below operation on the input and return the output.

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
# numpy array for training data
```

```
target_data = np.array([[0],[1],[1],[0]], "float32")
# numpy array for target data

model = Sequential()
#A Sequential model is appropriate for a plain stack of layers where
#each layer has exactly one input tensor and one output tensor & is not
#appropriate when our model has multiple inputs or multiple outputs.
#Any of your layers has multiple inputs or multiple outputs

model.add(Dense(16, input_dim=2, activation='relu'))
# we use activation function RELU, and has 16 nodes, 2D input dimension

model.add(Dense(1, activation='sigmoid'))
#we add output layer with activation function of sigmoid and has 1 output node


model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])
#then we compile the model, we use adam optimiser for

model.fit(training_data, target_data, epochs=1000)
#then we fit the model with 1000 epochs and we give the arguments
#as training and target data

scores = model.evaluate(training_data, target_data)
# then we evaluate the model and find the scores

print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
# then we find the binary accuracy which was our metrices

print (model.predict(training_data).round())
# then we print the model predictions
```

```
1/1 [==============================] - 0s 7ms/step - loss: 0.0263 - binary_accurac
Epoch 976/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.0263 - binary_accurac
Epoch 977/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0262 - binary_accurac
Epoch 978/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0262 - binary_accurac
Epoch 979/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0261 - binary_accurac
Epoch 980/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0261 - binary_accurac
Epoch 981/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.0260 - binary_accurac
Epoch 982/1000
1/1 [==============================] - 0s 14ms/step - loss: 0.0260 - binary_accura
Epoch 983/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0259 - binary_accurac
Epoch 984/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0259 - binary_accurac
Epoch 985/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.0258 - binary_accurac
Epoch 986/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.0258 - binary_accurac
Epoch 987/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0257 - binary_accurac
```

```
Epoch 988/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0257 - binary_accurac
Epoch 989/1000
1/1 [==============================] - 0s 9ms/step - loss: 0.0256 - binary_accurac
Epoch 990/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.0256 - binary_accurac
Epoch 991/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0255 - binary_accurac
Epoch 992/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.0255 - binary_accurac
Epoch 993/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.0254 - binary_accurac
Epoch 994/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.0254 - binary_accurac
Epoch 995/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0253 - binary_accurac
Epoch 996/1000
1/1 [==============================] - 0s 9ms/step - loss: 0.0253 - binary_accurac
Epoch 997/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.0252 - binary_accurac
Epoch 998/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0252 - binary_accurac
Epoch 999/1000
1/1 [==============================] - 0s 9ms/step - loss: 0.0251 - binary_accurac
Epoch 1000/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0251 - binary_accurac
1/1 [==============================] - 0s 274ms/step - loss: 0.0250 - binary_accur

binary_accuracy: 100.00%
[[0.]
 [1.]
 [1.]
 [0.]]
```

✓ 19s completed at 11:25 AM