

DATE: 06.08.2024

Deep Learning Concepts and Architectures

Ex. No : 1

Name: Sabarish Sankaran B

Reg No: 3122215002087

1. Implement simple vector addition in TensorFlow.

DESCRIPTION:

To implement simple vector addition in TensorFlow, create two tensors using `tf.constant` and add them using '`tf.add`'. It computes the element-wise sum of the vectors. Use `.numpy()` to convert the result to a NumPy array for easy display.

PROGRAM:

```
import tensorflow as tf

scalar = tf.constant(7)
scalar.ndim

vector = tf.constant([10, 10])
vector.ndim

matrix = tf.constant([[1, 2], [3, 4]])
print(matrix)
print('The number of dimensions of a matrix is: ' +
      str(matrix.ndim))

matrix = tf.constant([[1, 2], [3, 4]])
matrix1 = tf.constant([[2, 4], [6, 8]])

print(matrix + matrix1)
```

OUTPUT:

```
tf.Tensor(  
  [[1 2]  
   [3 4]], shape=(2, 2), dtype=int32)  
The number of dimensions of a matrix is: 2  
tf.Tensor(  
  [[ 3  6]  
   [ 9 12]], shape=(2, 2), dtype=int32)
```

2. Implement a regression model in Keras.

DESCRIPTION:

To implement a regression model in Keras, define a Sequential model and add layers to it. Start with a dense layer that has one unit and a linear activation function to predict continuous values. Train the model using model.fit with your data, then evaluate its performance using model.evaluate. Finally, use model.predict to make predictions on new data.

PROGRAM:

```
import keras  
from keras import layers  
import numpy as np  
  
X = np.array([[0], [1], [2], [3], [4], [5]], dtype=float)  
y = np.array([0, 1, 2, 3, 4, 5], dtype=float)  
model = keras.Sequential([  
    layers.Input(shape=(1,)),  
    layers.Dense(1)  
)  
  
model.compile(optimizer='adam', loss='mean_squared_error')  
  
model.fit(X, y, epochs=100, verbose=0)
```

```

loss = model.evaluate(X, y)
print("Mean Squared Error:", loss)

new_data = np.array([[0.5], [0.8], [1.0]])
predictions = model.predict(new_data)
print("Predictions:", predictions)

```

OUTPUT:

```

1/1 ----- 0s 56ms/step - loss: 0.3875
Mean Squared Error: 0.38746562600135803
1/1 ----- 0s 32ms/step
Predictions: [[0.52336323]
 [0.8923206 ]
 [1.1382921 ]]

```

3. Implement a SVM model in TensorFlow/Keras Environment.

DESCRIPTION:

To implement an SVM-like model in TensorFlow/Keras, create a custom model by subclassing `tf.keras.Model`. Define a dense layer with a linear activation function and L2 regularization to mimic SVM behaviour. Use a custom hinge loss function to approximate SVM's loss. Compile the model with an optimizer such as SGD and the hinge loss.

PROGRAM:

```

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import SGD

X_train = np.array([[0, 0], [1, 1], [1, 0], [0, 1]],
dtype=np.float32)
y_train = np.array([0, 0, 1, 1], dtype=np.float32)

```

```

class SVM(Model):
    def __init__(self):
        super(SVM, self).__init__()
        self.dense = Dense(1,
kernel_regularizer=tf.keras.regularizers.l2(0.01))

    def call(self, inputs):
        return self.dense(inputs)

model = SVM()

def hinge_loss(y_true, y_pred):
    return tf.reduce_mean(tf.maximum(1. - y_true * y_pred, 0.))

model.compile(optimizer=SGD(learning_rate=0.01), loss=hinge_loss)

model.fit(X_train, y_train, epochs=100, verbose=0)



loss = model.evaluate(X_train, y_train)
print("Loss:", loss)

predictions = model.predict(X_train)
print("Predictions:", predictions)

```

OUTPUT:

```

1/1  0s 47ms/step - loss: 0.8955
Loss: 0.8954741358757019
1/1  0s 33ms/step
Predictions: [[ 0.49999967]
 [-0.06939968]
 [ 0.48943105]
 [-0.05883107]]

```

4. Build a deep neural network model start with linear regression using a single variable.

DESCRIPTION:

To build a deep neural network model starting with linear regression using a single variable, define a Sequential model with multiple layers. Begin with an input layer and several hidden layers with activation functions like ReLU, followed by an output layer with a linear activation function for regression. Compile the model using an optimizer such as Adam and a loss function like mean squared error.

PROGRAM:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

X_train = np.array([[0], [1], [2], [3], [4], [5]],
dtype=np.float32)
y_train = np.array([0, 1, 2, 3, 4, 5], dtype=np.float32)

model = Sequential()
model.add(Dense(1, input_shape=(1,), activation='linear'))

model.compile(optimizer=SGD(learning_rate=0.01),
loss='mean_squared_error')

model.fit(X_train, y_train, epochs=100, verbose=0)

loss = model.evaluate(X_train, y_train)
print("Mean Squared Error:", loss)

new_data = np.array([[0.5], [1.5], [2.5]], dtype=np.float32)
predictions = model.predict(new_data)
print("Predictions:", predictions)
```

```

model = Sequential()
model.add(Dense(64, input_shape=(1,), activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))

model.compile(optimizer=SGD(learning_rate=0.01),
loss='mean_squared_error')

model.fit(X_train, y_train, epochs=100, verbose=0)

loss = model.evaluate(X_train, y_train)
print("Mean Squared Error:", loss)

predictions = model.predict(new_data)
print("Predictions:", predictions)

```

OUTPUT:

```

1/1 ————— 0s 31ms/step
Predictions: [[0.50131893]
 [1.5008863 ]
 [2.5004535 ]]
1/1 ————— 0s 92ms/step - loss: 0.0019
Mean Squared Error: 0.0018834971124306321
1/1 ————— 0s 42ms/step
Predictions: [[0.5617491]
 [1.5415802]
 [2.5216813]]

```

5. Build a deep neural network model start with linear regression using multiple variables.

DESCRIPTION:

To build a deep neural network model starting with linear regression using multiple variables, define a Sequential model with several layers. Begin with an input layer that matches the number of features, followed by multiple hidden layers with activation functions such as ReLU. Conclude with an output layer

using a linear activation function for regression.

PROGRAM:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

X_train = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]],
dtype=np.float32)
y_train = np.array([3, 5, 7, 9, 11], dtype=np.float32)

model = Sequential()
model.add(Dense(64, input_shape=(2,), activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))

model.compile(optimizer=Adam(learning_rate=0.01),
loss='mean_squared_error')

model.fit(X_train, y_train, epochs=100, verbose=0)

loss = model.evaluate(X_train, y_train)
print("Mean Squared Error:", loss)

new_data = np.array([[2.5, 3.5], [3.5, 4.5]], dtype=np.float32)
predictions = model.predict(new_data)
print("Predictions:", predictions)
```

OUTPUT:

```
1/1 ————— 0s 62ms/step - loss: 0.0027
Mean Squared Error: 0.0027128790970891714
1/1 ————— 0s 56ms/step
Predictions: [[6.0486217]
 [8.023477 ]]
```

6. Write a program to convert speech into text.

DESCRIPTION:

To convert speech into text, use the **SpeechRecognition** library. Initialize a **Recognizer** object and capture audio from a microphone using **sr.Microphone**. Use the **recognizer** to convert the captured audio to text with **recognize_google**. Handle exceptions for cases where speech is not recognized or where there are network issues. This setup allows for real-time speech recognition and conversion to text.

PROGRAM:

```
import speech_recognition as sr

recognizer = sr.Recognizer()

with sr.Microphone() as source:
    print("Listening...")
    audio = recognizer.listen(source)

try:
    text = recognizer.recognize_google(audio)
    print("You said:", text)
except sr.UnknownValueError:
    print("Sorry, I could not understand.")
```

OUTPUT:

```
Listening...
You said: hello how are you
```

Write a program to convert text into speech.

7. Write a program to convert text into speech.

DESCRIPTION:

To convert text into speech, use the gTTS (Google Text-to-Speech) library. Define the text you want to convert and create a gTTS object with the text and desired language. Save the generated speech to an audio file and optionally play it using system commands. Alternatively, use pyttsx3 for offline conversion, which speaks the text directly through the system's speakers.

PROGRAM:

```
import pyttsx3
text = "Hello, I am Sanjith Vikraman"
engine = pyttsx3.init()
engine.setProperty('rate', 150)
engine.setProperty('volume', 1)
engine.say(text)
engine.runAndWait()
print("Speech Recognized sucessfully!!!")
```

OUTPUT:

The speech is recognized as Sanjith Vikraman

1

Speech Recognized sucessfully!!!

RESULT:

To implement vector addition in TensorFlow, create and add tensors using `tf.constant` and `tf.add`. For a regression model in Keras, build a `Sequential` model with a linear output layer, compile with mean squared error, and train the model. For an SVM model in TensorFlow/Keras, create a custom model with a dense layer and hinge loss, and train it with SGD. For deep neural networks, start with single-variable linear regression by adding hidden layers and use a linear output, and similarly, handle multiple variables by adapting the input layer. Convert speech into text using `SpeechRecognition` by capturing and recognizing audio, and convert text into speech with `gTTS` or `pyttsx3` to generate and optionally play audio.