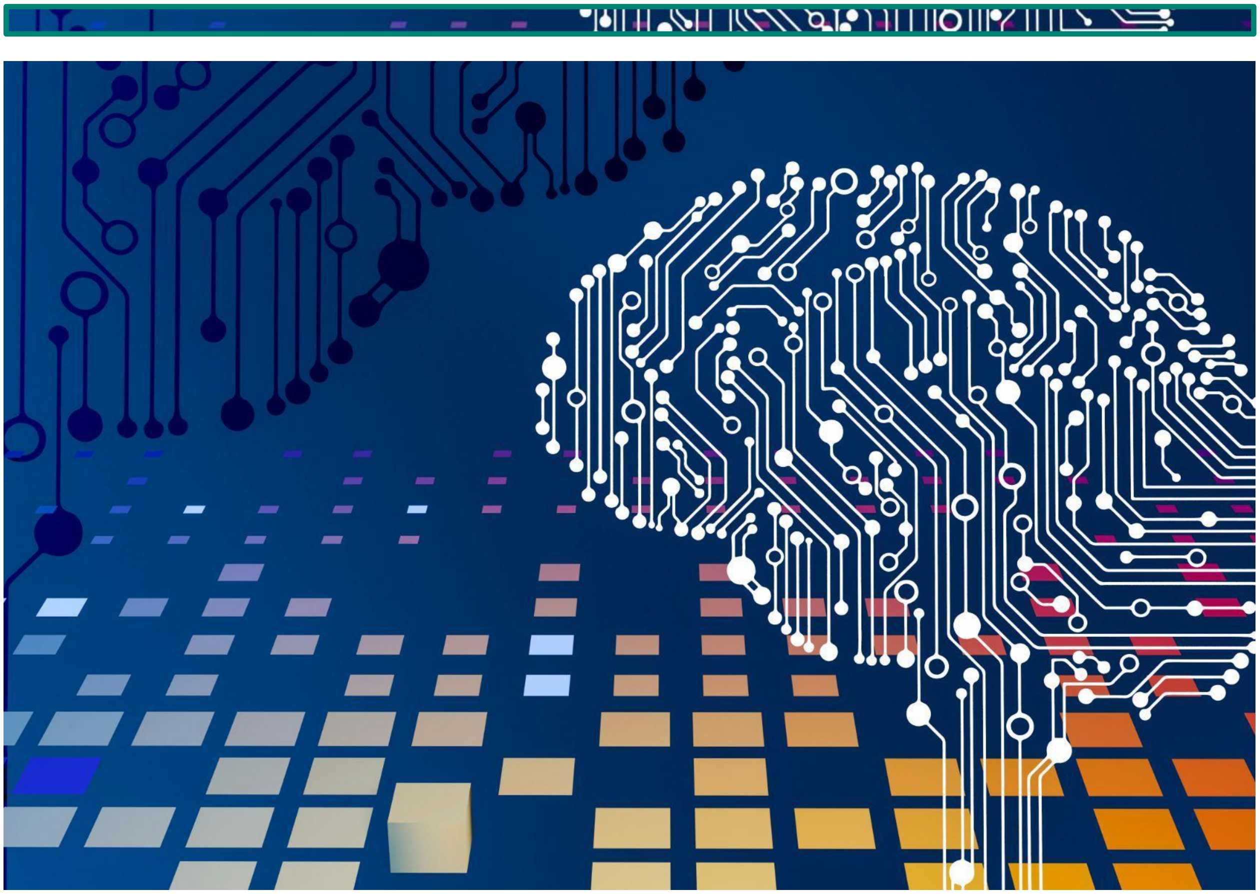

TENSORFLOW PRACTICALS

28.06.2024

SHARON P. A

Activity 1

create a 3*3 matrix of ones and a 3*3 matrix of zeros.

add them together using tensorflow

Required Hardware/Software's: Personal computer or laptop, Google collab

Procedure

```
import tensorflow as tf

# Create a 3x3 matrix of ones
matrix1 = tf.ones(shape=(3, 3), dtype=tf.float32)

# Create a 3x3 matrix of zeros
matrix2 = tf.zeros(shape=(3, 3), dtype=tf.float32)

# Add the matrices together
result = tf.add(matrix1, matrix2)

# Print the matrices and their sum
print("Matrix of Ones:")
print(matrix1.numpy())
print("\nMatrix of Zeros:")
print(matrix2.numpy())
print("\nSum of matrices:")
```

```
print(result.numpy())
```

Output

Matrix of Ones:

[[1. 1. 1.]

 [1. 1. 1.]

 [1. 1. 1.]]

Matrix of Zeros:

[[0. 0. 0.]

 [0. 0. 0.]

 [0. 0. 0.]]

Sum of matrices:

[[1. 1. 1.]

 [1. 1. 1.]

 [1. 1. 1.]]

Activity 2

implement a custom layer in tensorflow that performs a specific operations (eg.custom activation, function)

use this layer as a simple model

Required Hardware/Software's: Personal computer or laptop, Google collab

Procedure

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

# Step 1: Define the custom activation function
def custom_relu(x):
    return tf.minimum(tf.maximum(0.0, x), 6.0)

# Step 2: Create a custom layer that applies this activation function
class CustomActivationLayer(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super(CustomActivationLayer, self).__init__(**kwargs)

    def call(self, inputs):
        return custom_relu(inputs)

# Step 3: Build a simple model using the custom layer
model = Sequential([
```

```
    Dense(10, input_shape=(5,)),  # Input layer
    CustomActivationLayer(),      # Custom activation layer
    Dense(1)                      # Output layer
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Print the model summary
model.summary()

# Step 4: Generate some random data and train the model
import numpy as np

X_train = np.random.rand(100, 5)
y_train = np.random.rand(100, 1)

# Train the model
model.fit(X_train, y_train, epochs=5)
```

Output

Model: "sequential_1"

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 10)                60

 custom_activation_layer_1   (None, 10)                0
```

```
(CustomActivationLayer)

dense_3 (Dense)          (None, 1)               11

=================================================================
Total params: 71 (284.00 Byte)
Trainable params: 71 (284.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
4/4 [==============================] - 1s 4ms/step - loss: 0.1169
Epoch 2/5
4/4 [==============================] - 0s 4ms/step - loss: 0.1110
Epoch 3/5
4/4 [==============================] - 0s 4ms/step - loss: 0.1090
Epoch 4/5
4/4 [==============================] - 0s 6ms/step - loss: 0.1073
Epoch 5/5
4/4 [==============================] - 0s 4ms/step - loss: 0.1067
<keras.src.callbacks.History at 0x7eb4de26ac20>
```

Activity 3

implement a simple example of distributed training using tf.distribute strategy to train a model on multiple GPUs

Required Hardware/Software's: Personal computer or laptop, Google collab

Procedure

```
import tensorflow as tf

# Step 1: Set up the distribution strategy
strategy = tf.distribute.MirroredStrategy()

print('Number of devices: {}'.format(strategy.num_replicas_in_sync))

# Step 2: Prepare the dataset
# Load and preprocess the MNIST dataset
def preprocess(x, y):
    x = tf.cast(x, tf.float32) / 255.0
    y = tf.cast(y, tf.int64)
    return x, y

batch_size = 64

(train_images, train_labels), (test_images, test_labels) =
tf.keras.datasets.mnist.load_data()
train_images = train_images[..., tf.newaxis]
```

```
test_images = test_images[..., tf.newaxis]


train_dataset = tf.data.Dataset.from_tensor_slices((train_images,
train_labels))
train_dataset =
train_dataset.map(preprocess).shuffle(60000).batch(batch_size)


test_dataset = tf.data.Dataset.from_tensor_slices((test_images,
test_labels))
test_dataset = test_dataset.map(preprocess).batch(batch_size)


# Step 3: Create the model inside the strategy scope
with strategy.scope():
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28,
28, 1)),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D(64, 3, activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10)
    ])


    # Step 4: Compile the model
    model.compile(
        optimizer=tf.keras.optimizers.Adam(),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )
```

```python
# Step 5: Train the model
model.fit(train_dataset, epochs=5, validation_data=test_dataset)

# Step 6: Evaluate the model
test_loss, test_acc = model.evaluate(test_dataset)
print(f'Test accuracy: {test_acc}')
```

Output

Number of devices: 1

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

11490434/11490434 [==============================] - 0s 0us/step

Epoch 1/5

938/938 [==============================] - 62s 62ms/step - loss: 0.1534 - accuracy: 0.9538 - val_loss: 0.0471 - val_accuracy: 0.9853

Epoch 2/5

938/938 [==============================] - 61s 62ms/step - loss: 0.0468 - accuracy: 0.9860 - val_loss: 0.0382 - val_accuracy: 0.9884

Epoch 3/5

938/938 [==============================] - 53s 54ms/step - loss: 0.0332 - accuracy: 0.9900 - val_loss: 0.0375 - val_accuracy: 0.9880

Epoch 4/5

938/938 [==============================] - 62s 63ms/step - loss: 0.0249 - accuracy: 0.9921 - val_loss: 0.0352 - val_accuracy: 0.9888

Epoch 5/5

938/938 [==============================] - 55s 56ms/step - loss: 0.0180 - accuracy: 0.9942 - val_loss: 0.0293 - val_accuracy: 0.9901

157/157 [==============================] - 3s 17ms/step - loss: 0.0293 - accuracy: 0.9901

Activity 4

write a tensorflow function to calculate precision,recall and F1 Score for a multi class classifdication problem

Required Hardware/Software's: Personal computer or laptop, Google collab

Procedure

```
import tensorflow as tf

def simple_multi_class_metrics(y_true, y_pred, num_classes):
    # Convert predictions and true labels to one-hot encoding
    y_pred_onehot = tf.one_hot(y_pred, depth=num_classes)
    y_true_onehot = tf.one_hot(y_true, depth=num_classes)

    # Calculate true positives, false positives, and false negatives
    tp = tf.reduce_sum(y_true_onehot * y_pred_onehot, axis=0)
    fp = tf.reduce_sum((1 - y_true_onehot) * y_pred_onehot, axis=0)
    fn = tf.reduce_sum(y_true_onehot * (1 - y_pred_onehot), axis=0)

    # Calculate precision, recall, and F1-score for each class
    precision = tp / (tp + fp + tf.keras.backend.epsilon())
    recall = tp / (tp + fn + tf.keras.backend.epsilon())
    f1_score = 2 * precision * recall / (precision + recall +
tf.keras.backend.epsilon())

    # Return metrics as a dictionary
```

```
    metrics = {
        'Precision': precision.numpy(),
        'Recall': recall.numpy(),
        'F1-Score': f1_score.numpy()
    }

    return metrics

# Example usage:
y_true = tf.constant([0, 1, 2, 0, 1, 2])  # Example true labels
y_pred = tf.constant([0, 2, 1, 0, 0, 1])  # Example predicted labels
num_classes = 3  # Number of classes in the classification problem

metrics = simple_multi_class_metrics(y_true, y_pred, num_classes)
print("Precision: ", metrics['Precision'])
print("Recall: ", metrics['Recall'])
print("F1-Score: ", metrics['F1-Score'])
```

Output

Precision: [0.6666667 0.   0.   ]

Recall: [1. 0. 0.]

F1-Score: [0.79999995 0.   0.   ]