tensorflow-mnist-complete

February 29, 2024

1 MNIST Classification with Deep Neural Network

[ ]:
```
import numpy as np
import tensorflow as tf

import tensorflow_datasets as tfds
```

1.1 Data

[ ]:
```
mnist_dataset, mnist_info = tfds.load(name='mnist', with_info=True,␣
 ↪as_supervised=True)

mnist_train, mnist_test = mnist_dataset['train'], mnist_dataset['test']

num_validation_samples = 0.1 * mnist_info.splits['train'].num_examples
num_validation_samples = tf.cast(num_validation_samples, tf.int64)

num_test_samples = mnist_info.splits['test'].num_examples
num_test_samples = tf.cast(num_test_samples, tf.int64)


def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255.
    return image, label

scaled_train_and_validation_data = mnist_train.map(scale)

test_data = mnist_test.map(scale)


BUFFER_SIZE = 10000

shuffled_train_and_validation_data = scaled_train_and_validation_data.
 ↪shuffle(BUFFER_SIZE)
```

```
validation_data = shuffled_train_and_validation_data.
 ↪take(num_validation_samples)
train_data = shuffled_train_and_validation_data.skip(num_validation_samples)


BATCH_SIZE = 100

train_data = train_data.batch(BATCH_SIZE)
validation_data = validation_data.batch(num_validation_samples)
test_data = test_data.batch(num_test_samples)

validation_inputs, validation_targets = next(iter(validation_data))
```

1.2 Model

1.2.1 Outlining the model

[9]:
```
input_size = 784
output_size = 10
hidden_layer_size = 50

model = tf.keras.Sequential([
                            tf.keras.layers.Flatten(input_shape=(28,28,1)),
                            tf.keras.layers.Dense(hidden_layer_size,␣
 ↪activation='relu'),
                            tf.keras.layers.Dense(hidden_layer_size,␣
 ↪activation='relu'),
                            tf.keras.layers.Dense(output_size,␣
 ↪activation='softmax')
                            ])
```

1.2.2 Optimizer and the loss function

[10]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

1.2.3 Training the model

[11]:
```
NUM_EPOCHS = 5

model.fit(train_data, epochs = NUM_EPOCHS, validation_data=(validation_inputs,␣
 ↪validation_targets), verbose=2)
```

```
Epoch 1/5
540/540 - 2s - loss: 0.4131 - accuracy: 0.8825 - val_loss: 0.1981 -
val_accuracy: 0.9417 - 2s/epoch - 4ms/step
```

```
Epoch 2/5
540/540 - 1s - loss: 0.1774 - accuracy: 0.9472 - val_loss: 0.1459 -
val_accuracy: 0.9562 - 1s/epoch - 2ms/step
Epoch 3/5
540/540 - 1s - loss: 0.1327 - accuracy: 0.9601 - val_loss: 0.1196 -
val_accuracy: 0.9650 - 1s/epoch - 2ms/step
Epoch 4/5
540/540 - 1s - loss: 0.1090 - accuracy: 0.9674 - val_loss: 0.1003 -
val_accuracy: 0.9695 - 1s/epoch - 2ms/step
Epoch 5/5
540/540 - 1s - loss: 0.0910 - accuracy: 0.9727 - val_loss: 0.0894 -
val_accuracy: 0.9733 - 1s/epoch - 2ms/step
```

[11]:
```
<keras.src.callbacks.History at 0x7f6150120650>
```

1.3 Testing

[12]:
```
test_loss, test_accuracy = model.evaluate(test_data)
```

```
1/1 [==============================] - 0s 270ms/step - loss: 0.1028 - accuracy:
0.9661
```

[13]:
```
print('Test loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss,␣
↪test_accuracy*100.))
```

```
Test loss: 0.10. Test accuracy: 96.61%
```

[ ]:

[ ]:

[ ]: