MNIST Deep Net

April 15, 2020

1 Deep Neural Network For MNIST Classification

1.0.1 Import the relevant packages

[7]:
```
import numpy as np
import tensorflow as tf

import tensorflow_datasets as tfds
```

1.1 Data

[41]:
```
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)

#To shuffle the data, so it can be as random as possible

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 = 500

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 Outline the Model

[42]:
```
input_size = 784
output_size = 10
hidden_layer_size = 500

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)), # Input Layer
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'), # 1st hidden␣
 ↪layer
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'), # 2nd hidden␣
 ↪layer
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'), # 3rd hidden␣
 ↪layer
    tf.keras.layers.Dense(output_size, activation='softmax') # output layer
])
```

1.2.2 Optimizer and Loss Function

[43]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

1.3 Training

[45]:
```
NUM_EPOCHS = 10

model.fit(train_data,
          epochs = NUM_EPOCHS,
          validation_data = (validation_inputs, validation_targets),
          validation_steps = 10,
          verbose = 2)
```

```
Epoch 1/10
Epoch 1/10
108/108 - 4s - loss: 0.0265 - accuracy: 0.9915 - val_loss: 0.0278 -
val_accuracy: 0.9908
108/108 - 4s - loss: 0.0265 - accuracy: 0.9915 - val_loss: 0.0278 -
val_accuracy: 0.9908
Epoch 2/10
Epoch 2/10
108/108 - 4s - loss: 0.0197 - accuracy: 0.9939 - val_loss: 0.0257 -
val_accuracy: 0.9922
108/108 - 4s - loss: 0.0197 - accuracy: 0.9939 - val_loss: 0.0257 -
val_accuracy: 0.9922
Epoch 3/10
Epoch 3/10
108/108 - 4s - loss: 0.0167 - accuracy: 0.9948 - val_loss: 0.0228 -
val_accuracy: 0.9937
108/108 - 4s - loss: 0.0167 - accuracy: 0.9948 - val_loss: 0.0228 -
val_accuracy: 0.9937
Epoch 4/10
Epoch 4/10
108/108 - 5s - loss: 0.0140 - accuracy: 0.9954 - val_loss: 0.0177 -
val_accuracy: 0.9955
108/108 - 5s - loss: 0.0140 - accuracy: 0.9954 - val_loss: 0.0177 -
val_accuracy: 0.9955
Epoch 5/10
Epoch 5/10
108/108 - 5s - loss: 0.0142 - accuracy: 0.9954 - val_loss: 0.0140 -
val_accuracy: 0.9960
108/108 - 5s - loss: 0.0142 - accuracy: 0.9954 - val_loss: 0.0140 -
val_accuracy: 0.9960
Epoch 6/10
Epoch 6/10
108/108 - 5s - loss: 0.0118 - accuracy: 0.9961 - val_loss: 0.0117 -
val_accuracy: 0.9967
108/108 - 5s - loss: 0.0118 - accuracy: 0.9961 - val_loss: 0.0117 -
val_accuracy: 0.9967
Epoch 7/10
Epoch 7/10
108/108 - 5s - loss: 0.0108 - accuracy: 0.9964 - val_loss: 0.0104 -
val_accuracy: 0.9972
108/108 - 5s - loss: 0.0108 - accuracy: 0.9964 - val_loss: 0.0104 -
val_accuracy: 0.9972
Epoch 8/10
Epoch 8/10
108/108 - 5s - loss: 0.0087 - accuracy: 0.9972 - val_loss: 0.0106 -
val_accuracy: 0.9967
108/108 - 5s - loss: 0.0087 - accuracy: 0.9972 - val_loss: 0.0106 -
val_accuracy: 0.9967
```

```
Epoch 9/10
Epoch 9/10
108/108 - 5s - loss: 0.0103 - accuracy: 0.9964 - val_loss: 0.0175 -
val_accuracy: 0.9945
108/108 - 5s - loss: 0.0103 - accuracy: 0.9964 - val_loss: 0.0175 -
val_accuracy: 0.9945
Epoch 10/10
Epoch 10/10
108/108 - 5s - loss: 0.0082 - accuracy: 0.9972 - val_loss: 0.0168 -
val_accuracy: 0.9950
108/108 - 5s - loss: 0.0082 - accuracy: 0.9972 - val_loss: 0.0168 -
val_accuracy: 0.9950
```

[45]:
```
<tensorflow.python.keras.callbacks.History at 0x1afa3006108>
```

[45]:
```
<tensorflow.python.keras.callbacks.History at 0x1afa3006108>
```

1.4 Test the Model

[46]:
```
test_loss, test_accuracy = model.evaluate(test_data)
```

```
1/Unknown - 1s 700ms/step - loss: 0.0923 - accuracy: 0.9805
```

[47]:
```
print('Test Loss : {0:.2f}. Test Accuracy : {1:.2f}%'.format(test_loss,␣
↪test_accuracy * 100.))
```

```
Test Loss : 0.09. Test Accuracy : 98.05%
Test Loss : 0.09. Test Accuracy : 98.05%
```

Our Model has a Test Accuracy of ~ 98%