COMP2002 - Artificial Intelligence

Week 4 - Deep Learning Exercises - Model Answer

Introduction

This set of model answers is intended to show you possible ways of solving the deep learning exercises in week 4 – there are others. Please attempt the exercises before looking at the answers.

Activities

Your task is to go through the following tasks. Please note, that you are expected to complete some work on this outside of the timetabled sessions.

Exercise 1 - Mnist digit classification

The full code for this exercise was provided within the workshop sheet and will therefore not be repeated here. Similar code is also provided in the solution to Exercise 3

Exercise 2 - Deep neural network confusion matrix

Your task here is to produce a confusion matrix - in exactly the same way as was done for the machine learning examples and weeks 2 and 3. The only difference is that you're producing with a deep neural network rather than, for example a kNN classifier. So, you'll need to do the following:

- Import the relevant Scikit and Matplotlib packages.
- Produce a prediction for the test data (you could also do this for the training data if you are keen).
- Construct the confusion matrix.
- Plot the result.

Here is the code to achieve these steps:

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Make the predictions for the test set
predictions = model.predict(x_test).argmax(axis=1)

# Construct the confusion matrix (C) and plot it
C = confusion_matrix(y_test, predictions)
plt.imshow(C)
```

Exercise 3 - Fashion data

The next task is to refactor the training code so that is can be used with different datasets. We're not going to totally generalise the code, as the two datasets we're going to test have the same dimensions, but you could make it more general by taking out the hardcoded numbers and computing them from the data. Here is the function, which I'll then describe:

```
def train(x_train, y_train, x_test, y_test):
  # Define the inputs.
  inputs = tf.keras.Input(shape=(784, ))

  # Define the first layer.
  dense = layers.Dense(64, activation="relu")
  x = dense(inputs)

  # Define the second layer and outputs.
  x = layer.Dense(64, activation="relu")(x)
  outputs = layers.Dense(10)(x)

  # Complie the model.
  model.complie(
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer = tf.keras.optimizers.RMSprop(),
    metrics = ["accuracy"],
  )

  # Train the model.
  history = model.fit(x_train, y_train, batch_size=64, epochs=2, validation_split=0.2)

  # Evaluate the model.
  test_scores = model.evaluate(x_test, y_test, verbose=2)
  print("Test accuracy:", test_scores[1])

  # Return the model.
  return model
```

The function does everything that your code in Exercise 1 did, with the exception of loading and normalizing the data. If we're going to have a generic training function, we'll do that prior to calling the function (as you'll see shortly). Other than returning the model, nothing else has changed from Exercise 1.

Once we've got our function, we can call it as follows:

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(60000, 784).astype("float32")/255
x_test = x_test.reshape(10000, 784).astype("float32")/255

model = train(x_train, y_train, x_test, y_test)
```

This repeats the training that was done in Exercise 1. What we can do now is change the dataset and repeat as follows:

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

x_train = x_train.reshape(60000, 784).astype("float32")/255
x_test = x_test.reshape(10000, 784).astype("float32")/255

model = train(x_train, y_train, x_test, y_test)
```

Exercise 4 - Another confusion matrix

here we're going to do the same thing as in Exercise 3 - refactor the confusion matrix code so that it can

be reused for different problems. Here is a refactored version of the confusion matrix cose, put into a Python function called **plot_cf**:

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

def plot_cf(model, x_test, y_test):
  predictions = model.predict(x_test).argmax(axis=1)
  C = confusion_matrix(y_test, predictions)
  plt.imshow(c)
```

Finally, call the function - pass it the model and test the data and you will see the corresponding confusion matrix.

```
plot_cf(model, x_test, y_test)
```

Note, this works with whichever version of the model you have from the two datasets in Exercise 3.