COMP2002 - Artificial Intelligence

Week 4 - Deep Learning Exercises

Introduction

The aim of this sheet of exercises is to get you started with implementing machine learning software. You should complete the exercises ahead of the week 5 seminar session. The model answers will be published shortly after, giving you enough time to re-attempt the exercises after the demonstration in the seminar.

You should complete the exercises in a Jupyter Notebook. For this week, I suggest you use Google's Co-Lab to complete the exercises – we'll be using TensorFlow and associated libraries, and they have them all installed. You are of course welcome to use your own Jupyter installation and install TensorFlow yourself.

Activities

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

Exercise 1 - Mnist digit classification

The first task this week is to classify the Mnist digit dataset using a deep neural network. You will use the **Keras** interface, which sits between you and TensorFlow.

This exercise is taken from a TensorFlow tutorial, which provides an excellent introduction to building deep neural networks – you can find it here.

You should begin by entering the following code (I would use a different cell for each block of code shown here). First, import the modules you'll need:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
```

Having done that, define the input layer. This will be taking the raw pixel values, and there are 784 of them (the images have 28x28 pixels).

```
inputs = tf.keras.Input(shape=(784,))
```

We can now define our first layer of hidden neurons. These will start to convolve the inputs to extract features. It will process the **inputs** variable defined in the previous step and produce an output called **x**. We will go from 784-dimensional inputs to 64-dimensions.

```
dense = layers.Dense(64, activation="relu")
x = dense(inputs)
```

We then add the next layer, which further extracts features. This will process **x** from the previous step and result in a new **x** that is also 64-dimensional, before passing those outputs to a final layer that is 10-dimensional – the outputs of that layer go into a variable called **outputs**.

```
x = layers.Dense(64, activation="relu")(x)
outputs = layers.Dense(10)(x)
```

Each of those outputs represents one of the digits. Having defined the layers of the network we can now construct the model:

```
model = tf.keras.Model(inputs=inputs, outputs=outputs, name="mnist_model")
model.summary()
```

The model.summary call produces some useful summary information about the model, and should look like this:

Model: ”mnist_model”

Layer (type)	Output Shape	Param #
input 2 (Input Layer)	[(None, 784)]	0
dense (Dense)	(None, 64)	50240
dense_1 (Dense)	(None, 64)	4160
dens_2 (Dense)	(None, 10)	650

Total params: 55,050
Trainable params: 55, 050
Non-trainable params: 0

Like **Scikit**, Keras comes with some test datasets – one of which is the Mnist digit classification task we've spoken about previously. The images' pixels are on the range 0 to 255 – we want them between 0 and 1, so normalise them by dividing by 255

```
(x_train, y_train),(x_text, y_test)=tf.keras.datasets.mnist.loaddata()

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

Having loaded the data, compile it as follows. You will need to specify a loss function (to determine how good the predictions are), an optimiser, and accuracy metrics to report on how well the predictions do. You can then train the model.

```
model.complie(
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=TRUE),
  optimizer=tf.keras.optimizers.RMSprop(),
  metrics=["accruacy"],
)

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

test_scores = model.evalute(x_test, y_test, verbose=2)
print("Test accuracy:", test_scores[1])
```

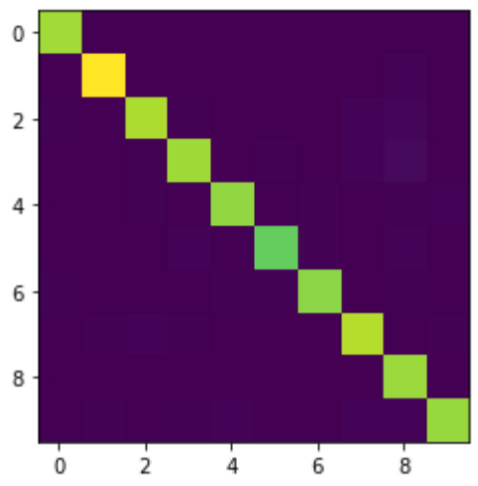
Exercise 2 - Deep neural network confusion matrix

Having trained the model you can use its **predict** method to make predictions for a dataset. This is done as follows:

predictions = model.predict(x_test).argmax(axis=1)

The result of the predict method is 10 outputs for each input image. These represent the likelihood that the image represents each image, so there is the likelihood that the image is a 0, the likelihood that it's a 1, and so on. What you need to do is find the largest of these, which will give you the most likely prediction – the **argmax** method will do this for you. The **axis=1** argument will return the highest value in each row, so you can do the whole lot in one go.

Using the Sklearn **confusion_matrix** function and Matplotlib's **imshow** produce and plot a confusion matrix for the test data. The predictions are in the **predictions** variable, and the true values are in **y_test**. You should get something that looks like this:



Exercise 3 - Fashion data

Another dataset you'll find in Keras is similar to the Mnist digits – it describes images of fashion items. These are also 28x28 pixels, and there are also ten classes.

The similarity between the two problems means you can put most of the code above into a function, and pass it the data to predict. Refactor your training code (up to calling the end of exercise 1) into a function called **train**. Then pass it the two datasets and assure yourself that you can use the function to train either a model for type of data. Your function should return the **model** so that it can be used elsewhere.

Exercise 4 - Another confusion matrix

Refactor the code from Exercise 2 so that you can also reuse your confusion matrix. You should call the function **plot_cf**, and it should take the model, the true class (**y_test**), the model, and the test inputs (**x_test**) so that it can produce predictions and then the corresponding confusion matrix.