7CCSMPNN: Pattern Recognition, Neural Networks and Deep Learning

KING'S College LONDON

Coursework 1: Deep Neural Networks

Instructors: Oana Cocarascu, Jeroen Berrevoets

Instructions

This part of the coursework requires you to build and train a deep neural network for classifying handwritten digits. Your network must be built using KERAS. Please see the module's KEATS page for a tutorial on using KERAS. This tutorial describes how to build networks for classifying the MNIST dataset. This dataset consists of 70000 28-by-28 pixel images of handwritten digits, which are by convention split into 60000 images for training and 10000 for testing. The tutorial provides instructions on how to build both a MLP and a CNN for MNIST digit classification. You can use either of these as a starting point and experiment with making modifications to improve performance. You may also use pre-built networks from the model zoo, or you may choose to build your own deep network based on your own or someone else's design. Please note that in all cases **you must train your own neural network**: anyone who is found to have submitted a model that they have not trained themselves (e.g., a pre-trained network obtained from the internet) will receive a mark of zero.

- **Submission:** Your trained neural network must be saved **as a .h5 file and submitted via KEATS**. You can give this file any name you wish, but it must end with a .h5 extension. Note it should be possible to test your neural network to predict class labels on a standard PC running Linux without the need for special hardware, such as a GPU or RAM in excess of 16GB. 16GB is sufficient to test a model with around 1 billion parameters, however, KEATS places a limit of 500MB on the size of any file that you can upload which limits the size of any network you can submit to around 40 to 50 million parameters.
- **Evaluation:** Your neural network will be assessed by testing the accuracy with which it classifies another, unseen, test data ("`my testset`"). This testing data consist of about 12000 samples and is distinct from the testing and training data provided as part of the MNIST dataset, but in common with MNIST (after applying the pre-processing steps described in the tutorial) consists of images of handwritten digits that: have pixel values between 0 and 1, large values correspond to pen strokes while small values correspond to the background (i.e. images contain white digits on a black background when shown with a gray colormap), have been scaled and cropped (but not otherwise augmented) to be 28-by-28 pixel images with the digits approximately in the centre. The primary difference is that the digits in `my testset` have been written by different people, so may have different characteristics.

The reasons for evaluating performance on this new dataset are two fold. Firstly, because the MNIST test set is freely available it is not possible to ensure that submitted networks have not been trained with this data. If a network is assessed with the same data it was trained on, this does not provide a fair assessment of how well the network can classify unseen data. Secondly, this scenario is more consistent with the sort of problem you might be faced with in the real-world, where you need to develop a system that will work "in production", with new data that may not have even been generated yet. It is difficult to know if a classifier will generalise well to unseen data, but to have a chance of succeeding it is important for you to ensure that your model is not over-fitted to the MNIST data and that it can can cope with digits that may vary in appearance compared to the MNIST data.

- **Marking:** Marks will be awarded based on how accurately your neural network performs the classification of `my testset`. Note that the tutorial on using KERAS explains how to build two networks for classifying the MNIST data: the MLP can classify the standard MNIST test set with an accuracy of about 98%; the CNN can classify the standard MNIST test set with an accuracy of about 99%. The MLP can classify `my testset` with an accuracy of about 90%; the CNN can classify `my testset` with an accuracy of about 95% (these scores are very achievable).

Accuracy (%) on `my testset`	**Score**
$[97, 100)$	20
$[96.5, 97)$	18
$[96, 96.5)$	16
$[95.5, 96)$	14
$[95, 95.5)$	13
$[94.5, 95)$	12
$[94, 94.5)$	10
$[90, 94)$	9
$[85, 90)$	8
$[80, 85)$	7
$[70, 80)$	6
$[60, 70)$	5
$[50, 60)$	4
$(0, 50)$	3
0	0

Table 1: How scores will be awarded.

Frequently asked questions

Q: How different is `my testset` from the MNIST dataset? Should we expect rotations of the digits, and if so, are these small tilts (e.g. 0-15 degrees) or more pronounced rotations (e.g. 45/90 degrees / upside down)? Are there any other differences apart from the different handwriting styles, such as shifts, scale changes, flips, shears etc?

A: The test set is another sample of handwritten digits, with all the variation you might expect to find "in the wild". That means handwriting may differ, slight tilts may be present, but every digit is fully recognizable (i.e. there are no 45 or 90 degree flips).

Q: Can I also use some of the image data I found or generated combined with the MNIST dataset you gave me to train my model?

A: You can use any dataset you like. But if you choose to go beyond MNIST data, I would recommend ensuring that the extra samples are very "MNIST-like"; the more dissimilar they are, the more it could hurt results on the test set. Make sure to apply the correct pre-processing steps.

Q: Does model complexity impact the marks awarded (i.e., 2 models with 95% on `my testset`, 1 has 100,000 parameters and the other has 1 million parameters)? Will highly complex models be penalised?

A: We are evaluating accuracy exclusively. Complexity is not penalised. That said, the model must be uploadable to KEATS which imposes a 500MB cap on submissions.

Test script: To help you check that your network is compatible with our test environment, here is the code we will be using to test its performance on the MNIST dataset:

```
import numpy as np
import os
from tensorflow.keras.models import load_model

# Load .h5 file of arbitrary name for testing (last if more than one)
print(os.getcwd())
for file in os.listdir(os.getcwd()):
    if file.endswith(".h5"):
        print(file)
        net = load_model(file)
net.summary()

# Determine what type of network this is
input_dims = net.input_shape
netType = 'CNN' if len(input_dims) > 2 else 'MLP'

# Test with MNIST data
from tensorflow.keras.datasets import mnist
(x_train, labels_train), (x_test, labels_test) = mnist.load_data()
x_test = x_test.astype('float32') / 255
if netType == 'MLP':
    x_test = x_test.reshape(10000, 784)
else:
    x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# Evaluate
outputs = net.predict(x_test)
labels_predicted = np.argmax(outputs, axis=1)
correct_classified = np.sum(labels_predicted == labels_test)
print('Percentage correctly classified MNIST =', 100 * correct_classified / labels_test.size)
```

Tensorflow version: To test the submissions, we will be using tensorflow 2.18.0 (tf 2.18). We recommend to use this version, but if you can only use an older version, please put the version number as part of the submitted file name (e.g., xy_submission_tf213). This will help us to identify it and resolve any potential issues during testing. Good luck!

Plot misclassified samples: To further evaluate your results, you can also add the following script to the test script above to plot a few misclassified samples:

```
# Find and plot some misclassified samples
misclassified_indices = np.where(labels_predicted != labels_test)[0]
n_plot = min(len(misclassified_indices), 8)
import matplotlib.pyplot as plt

if n_plot > 0:
    fig, axes = plt.subplots(2, n_plot, figsize=(12, 3))
    fig.subplots_adjust(hspace=0.5)

    for i in range(n_plot):
        idx = misclassified_indices[i]
        axes[0, i].imshow(x_test[idx].reshape(28, 28), cmap='gray_r')
        axes[0, i].set_title(f'True: {labels_test[idx]}', y=1.05)
        axes[0, i].get_xaxis().set_visible(False)
        axes[0, i].get_yaxis().set_visible(False)

        if netType == 'MLP':
            output = net.predict(x_test[idx:idx+1].reshape(1, 784))
        else:
            output = net.predict(x_test[idx:idx+1].reshape(1, 28, 28, 1))

        output = output[0, 0:]
        axes[1, i].bar(range(10), output)
        axes[1, i].set_xticks(range(10))
        axes[1, i].set_title(f'Pred: {np.argmax(output)}', y=1.05)
        axes[1, i].get_yaxis().set_visible(False)

        if i == 0:
            axes[1, i].get_yaxis().set_visible(True)
            axes[1, 0].set_ylabel('Probability')
            axes[1, 0].set_yticks([0, 0.25, 0.5, 0.75, 1])

        fig.subplots_adjust(bottom=0.2)
        fig.text(0.51, 0.05, 'Classes', ha='center', va='center')
    plt.savefig("misclassified.pdf")
else:
    print("No misclassified samples found. No plot generated.")
```