Artificial Intelligence and Data Analytics for Engineers

Exercise 9

Introduction

Solving exercises is not mandatory, and exercises will not be graded or corrected by the lecture team. However, we strongly advise you to do as many exercises as possible to prepare for the final exam. Some of the tasks will be discussed with a presentation of the solutions during the exercise session on Thursday. If you want to do exercises at home, we suggest installing Anaconda (https://www.anaconda.com/distribution/ - you will want to download the Python 3.7 version).

In case you have any questions, feel free to send us an e-Mail to: aidae@ima-ifu.rwth-aachen.de.

Task 0) Setting up your Python environment

Since you created a Python environment the first week, we only need to enable the environment from now on. This needs to be done **every time** you open a new terminal, for instance after a system restart or if you accidentally closed the Terminal. Open a terminal and enter:

```
source activate aidae
```

Now you are ready to start programming in Python. Remember to ensure that spyder is installed in your environment (`conda install spyder`) and launched from the terminal where you activated the environment.

Task 1) Tensorflow basics

a) For the following tasks we will use the alpha version of Tensorflow 2, which you need to install first. Ensure that you have activated the environment and enter the following commands into your terminal:

```
pip install tensorflow==2.0.0-alpha0
```

b) We want to (again) build a classifier that, given an image of a handwritten number, figures out which number that is. This time, we will use a neural network to do that.

First, please import tensorflow, and load the MNIST digit dataset:

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

The tf.keras.datasets.mist API provides you with n different pictures of handwritten numbers. These are formatted in (n, 28, 28) numpy arrays of unsigned integers in the range 0-255 as x values, where each value represents a pixel's greyscale value.
The target value is a numpy array of shape (n, 10), where each value is in the range 0 – 1. This is a one hot encoding that represents the digit depicted in the associated image.
Let's start by having a look at the images: Use matplotlib to visualize the first image of the test set x_train[0].

c) Let's build a simple classifier. Our model will consist of a preprocessor that converts a (28, 28) 2-dimensional input tensor into a (28*28) 1-dimensional output tensor, and a single layer of 10 densely connected neurons which will consume that (28*28) tensor. For the dense layer we use the softmax activation function, which will lead to an output tensor that adds up to one and has no values below zero. This is helpful because we can interpret it as a probability distribution.

```
model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),
      tf.keras.layers.Dense(10, activation='softmax')
      ])
model.compile(loss='sparse_categorical_crossentropy', metrics=['accuracy'],
optimizer='adam')
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

This will train the classifier for 5 epochs and print its accuracy on both, train and test set. Note that the fit function expects numpy arrays or tensors, where the first dimension differentiates individual training samples. In other words, if you have n individual training samples and each has an m dimensional feature vector, you would provide a (n, m) shaped array, similar to the way scikit learn handles this.
Which accuracies do you achieve?

d) We will be able to improve the accuracy by either adding additional layers, pre/postprocess our data or training the network for a longer time. Let's start with the latter. Increase the number of epochs to 20. Before you start the training and evaluation process again, we want to add a way to easily see how the accuracies change during the process. For this we need to retrieve the return value of `model.fit`. That return value is a dict, in which you can find a field named `history` which contains another dict. This history dict contains the fields `accuracy` and `val_accuracy`. Each of these is a list, containing the accuracies after each epoch for the training and validation datasets, respectively. Plot these using matplotlib such that both line graphs end up in the same diagram, where the number of the epoch is on the x-axis, and the corresponding accuracy is on the y-axis. Which effect does increasing the number of epochs have?

e) To improve the convergence behavior, you can preprocess the input data, such that the values are not between 0 and 255 but between 0 and 1 (i.e. 1 becomes 1/255, 2 becomes 2/255) etc. Apply this transformation to the `x_train` and `x_test` arrays prior to training and see what happens.

f) Experiment and see what happens if you add one or multiple Dense layers of different size between the existing Flatten and the Dense layers to the sequential model. Also feel free to try different activation functions. A list of all available activation functions can be found here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/activations
Attention: The existing dense layer of 10 neurons needs to stay the last within the sequential model, because otherwise the output will have the wrong format.

g) To improve the accuracy on the test set even further, you can add dropout layers (`tf.keras.layers.Dropout(p)`) between the individual Dense layers. p is the dropout probability, start with `p=0.2` and see what significantly higher and lower values result in.

Task 2) Regression

There is a very good regression tutorial available in the Tensorflow documentation: https://www.tensorflow.org/alpha/tutorials/keras/basic_regression

We highly recommend that you read it and implement this yourself.