Homework 03

IANNwTF

November 12, 2022

This week's deadline is *19.11., 23:59*.

Submit your homework via https://forms.gle/ApAZ5ubY8ewgNmJA9

Remember that you now have to review another group's homework. More on that can be found further down.

Contents

1 Reviews & How to do them

Welcome back to the third homework for IANNwTF. This week you will get to properly use and play around with Tensorflow! Before we get to the fun though we have an organisational matter to discuss.

Starting this week, in addition to handing in your homework, you will have to review last week's homework of two groups and note down which group you reviewed on the homework submission form. This requires you to find two other groups and have a short (10 min) meeting where you go over each others homework submission, discuss it and then write a short summary of what you discussed on each others forum pages. We recommend using the Q&A timeslots for this purpose, but you can meet however and whenever you like. The main review part of this should be the discussion you have together. The written review in the forum should merely be a recap of the main points you discussed so that we can see that you did something and the reviewed group has access to a reminder of the feedback they received.

As to how a discussion could look like, you could for example have the group being reviewed walking the other two groups through their code. The other two groups should try to give feedback, e.g. "I like how you designed your data pipeline, looks very efficient.", "For this function you implemented there actually exists a method in e.g. Numpy or Tensorflow you could've used instead." or "Here you could've used a slightly different network architecture and higher learning rate to probably achieve better results.", and note down their main points.

Important! Not every member of every group has to be present for this. You could for example implement a rotation where every group member of yours only has to review every third homework.

2 Assignment: MNIST classification

This time we are going to train a MLP to classify handwritten digits. But instead of building the MLP from scratch like we did last week, we are going to make use of TensorFlow's pre-built layers and functions.

2.1 Loading the MNIST dataset

The MNIST dataset consists of seventy thousand labelled images, each depicting a single handwritten digit. This may sound like a lot of data, but as you'll see for yourselves in a bit, the images are rather small.

The MNIST dataset is included in TensorFlow, so you getting access to it is actually pretty easy. You can load it directly into your code like this:

```python
import tensorflow_datasets as tfds
import tensorflow as tf

(train_ds, test_ds), ds_info = tfds.load('mnist', split=['train', 'test'], as_supervised=True, with_info=True)
```

Note that the dataset is already split into training data and testing data. Additionally we get some information about the dataset in the form of the ds_info object. You can take a look at it by simply using print(ds_info). It will also be helpful later when we want to take a look at a sample of images.

When working with data, always make sure that you understand what you are dealing with first! How many entries are there in the dataset? What is the format? You must consider questions like these every time you are designing a network. Answer the following:

- How many training/test images are there?
- What's the image shape?
- What range are pixel values in?

Most of the information can be found in ds_info, and you can take a closer look at the data type of the images (uint8) and it's maximum value to answer the third question (hint: It is somewhere between 250 and 260).

It can also be helpful to visualize the data. You can sample a few images with their corresponding labels using tfds.show_examples, like this:

```
tfds.show_examples(train_ds, ds_info)
```

2.2 Setting up the data pipeline

After getting to know the dataset it makes sense to create a data pipeline function that prepares your data for use in your model. You can use this pipeline to prepare the training and test data one after another for use in your network. You can obtain the dataset from the tensorflow-datasets package as showcased in the lecture[1]. In your pipeline you will need to do a number of things. You may follow the example code from the lecture again closely here, but let's summarize the steps:

The MNIST handwritten digits images come in uint8 datatype. This refers to unsigned 8-bit integers (think numbers 0-255). As the network requires float values (think continuous variables) as input rather than integers (whole numbers), we need to change the datatype: (**map**[2] in combination with **lambda** expressions[3] can be really useful here). In your first lambda mapping you want to change the datatype from uint8 to tf.float values [4]. To feed your network the 28x28 images also need to be flattened. Check out the reshape function [5], and if you want to minimize your work, try and understand how it interacts with size elements set to the value -1 (infering the remainder shape). In order to improve the performance you should also normalize your image values. Generally this

[1] https://www.tensorflow.org/datasets/catalog/mnist
[2] https://www.tensorflow.org/api_docs/python/tf/data/Dataset#map
[3] https://docs.python.org/3/tutorial/controlflow.html?highlight=lambda#lambda-expressions
[4] https://www.tensorflow.org/api$_{d}$*ocs/python/tf/cast*
[5] https://www.tensorflow.org/api$_{d}$*ocs/python/tf/reshape*

means bringing the input close to the standart normal (gaussian) distribution with $\mu = 0$ and $\sigma = 1$, however we can make a quick approximation as that: Knowing the inputs are in the 0-255 interval, we can simply divide all numbers by 128 (bringing them between 0-2), and finally subtracting one (bringing them into -1 to 1 range). Additionally you need to encode your labels as one-hot-vectors[6]. Remember a very similar example for the data preparation can be found in the lecture contents.

2.3 Building a deep neural network with TensorFlow

Now that you have your data pipeline built, it is time to create your network. Check out the courseware for how to go about building a network with TensorFlow's Keras. Following that method, we want you to build a fully connected feed-forward neural network to classify MNIST images with. To do this, have a look at 'Dense' layers[7]; they basically provide you with the same functionality as the 'Layer' class which you have implemented last week. TensorFlow also provides you with every activation function you might need for this course[8]. A good (albeit arbitrary) starting point would be to have two hidden layers with 256 units each. For your output layer, think about how many units you need, and consider which activation function is most appropriate for this task.

2.4 Training the network

Define a training loop function which receives

- The number of epochs
- The model object
- The training dataset
- The test dataset
- The loss function
- The optimizer
- Different arrays for the different values you want to track for visualization

It should return the filled arrays after your model is done training. Before you call the function you will have to define your hyperparameters and initialize everything. To start off you can use 10 epochs, a learning rate of 0.1, the categorical cross entropy loss[9] and the optimizer SGD[10].

[6] https://www.tensorflow.org/api_docs/python/tf/one_hot
[7] https://www.tensorflow.org/api$_{d}$ocs/python/tf/keras/layers/Dense
[8] https://www.tensorflow.org/api$_{d}$ocs/python/tf/keras/activations
[9] https://www.tensorflow.org/api$_{d}$ocs/python/tf/keras/losses
[10] https://www.tensorflow.org/api$_{d}$ocs/python/tf/keras/optimizers/SGD

2.5 Visualization

After traing visualize the performance of your model using matplotlib and the values that you collected during training and testing. Here is just one example that you could use.

```python
def visualization(train_losses, train_accuracies, test_losses, test_accuracies):
  """ Visualizes accuracy and loss for training and test data using the mean of each epoch.
      Loss is displayed in a regular line, accuracy in a dotted line.
      Training data is displayed in blue, test data in red.
  Parameters
  ----------
  train_losses : numpy.ndarray
    training losses
  train_accuracies : numpy.ndarray
    training accuracies
  test_losses : numpy.ndarray
    test losses
  test_accuracies : numpy.ndarray
    test accuracies
  """
  plt.figure()
  line1, = plt.plot(train_losses, "b-")
  line2, = plt.plot(test_losses, "r-")
  line3, = plt.plot(train_accuracies, "b:")
  line4, = plt.plot(test_accuracies, "r:")
  plt.xlabel("Training steps")
  plt.ylabel("Loss/Accuracy")
  plt.legend((line1, line2, line3, line4), ("training loss", "test loss", "train accuracy", "test accuracy"))
  plt.show()
```

3 Adjusting the hyperparameters of your model

At this point you should have a working model. Now we want you to start adjusting all the parameters you can think of and see how it affects your models performance. The main hyperparameters that you could adjust are the learning rate, batch size, the number and size of layers of your model and the optimizer that you are using (and e.g. in SGD's case the momentum hyperparameter). You could try adjusting these one by one, or in combinations (e.g. lower learning rate combined with a higher momentum).

We want you to note down at least 4 deviations from your initial setup that you found interesting and try to interpret the results that you got with those setups. One idea here could be to see how small you can make your network while still achieving comparable results.