Homework 03

IANNWTF 20/21

Submission until 18th Nov 23:59 via https://forms.gle/H31ckx5251Qg3Ndm6

Welcome back to the third homework for IANNWTF. Now we already know how to make small Multilayered Perceptrons ourselves and understood how we train them - by backpropagating the error through the network. Now, while this might still not be enough to find a cure for the Corona virus, we will try if we can already identify bacteria populations from their genome sequence. To do so, we will implement a Multi-layer Perceptron using keras and train it on the genomics data set provided by tensorflow.

As always, feel free to ignore the following instructions and try doing it by yourself, if you're motivated. In case you are struggling with the instructions, there are hints (denoted by the superscript numbers) at the end of the document. Do not read them immediately, but think about it first, and only go to them if you're stuck. Sometimes instructions might be a bit vague and that might be intentional. We leave most things up to you specifically to enable the respective learning experiences which we deem necessary. But sometimes there are also mistakes from our side, so don't be shy to ask if sometimes is unclear for you or doesn't make sense.

1 Data set

In this homework we want to classify bacteria based on their genome sequences. The 'genomics_ood' data set is already inbuilt in Tensorflow. The input will be a genomic sequence, which (short biology recap) consists of the characters {A, C, G, T} (also known as nucleotides). Each genomic sequence will be 250 characters long, with each char one of the 4 options. We want to classify each genome sequence to 1 of 10 bacteria populations. (Technically, the data set also contains 120 more 'out of distribution' classes which are stored in `test_ood` and `validation_ood`, but you can disregard them for this homework.) Learn more about the data set here and have a look at example elements: https://www.tensorflow.org/datasets/catalog/genomics_ood

First you will need to load the data set from the module `tensorflow_datasets`. [1]

Then we will need some preparations on the data:

Take out a bunch of examples for our network. The whole dataset contains 1 million training examples and 100.000 test examples. It will be sufficient to use 100.000 examples

for training and 1000 for testing. (If you are not sure what is the purpose of having a training and testing data set, check the chapter 'Gradient Tape' on Courseware Week 03 again. But we will also talk about that next week in more detail.)

The genomic sequences come as string-tensor which are not so handy to work with. Instead, we would like to have the genomic sequence with one-hot-vectors for encoding the nucleotides (A, C, G, T). But because the sequences are string tensors, we cannot simply call `tf.one_hot(x,4)` on them. Therefore we need a function which converts the string tensor into a usable tensor that contains the one-hot-encoded sequence. You can try to build this function yourself (which might potentially be tricky, but we will give you some instructions in the hints[2]). Otherwise we also provide it for you in the end of the document. Also apply one-hot-encoding on the labels. [3]

For an outstanding we would like to see you using an input pipeline, including batching and (ideally) prefetching, but otherwise you can also do it without one.

2 Model

We will implement a simple fully connected feed forward neural network like the last time. Our network will have the following layers:

- Hidden layer 1: 256 units. With sigmoid activation function.
- Hidden layer 2: 256 units. With sigmoid activation function.
- Output: 10 units. With softmax activation function.

Instead of implementing our own layer we directly implement the network using pre-built layers from TensorFlow (keras). [4]

3 Training

Then train your network for 10 epochs using a learning rate of 0.1. As a loss use the categorical cross entropy.[5] As an optimizer use SGD.

For the training loop you can use the MNIST notebook on Courseware as orientation or try to build one yourself. You can, but you don't have to, compute the loss of each step with the running average, as we did it in the MNIST notebook.

For this task, an accuracy of 35 - 40% is sufficient.

4 Visualization

Visualize accuracy and loss for training and test data using matplotlib. [6]

Notes

[1]Check out `tfds.load()` for that.

[2] First replace the character with a number from 0-3. You might have to translate the "string" numbers into int numbers - check out `tf.cast()` for that. Split after each number. And only then call `tf.one_hot(x,4)` on them. It is also important that your inputs have the shape (batchsize, 1000) in the end.

[3]You can use the standard `tf.one_hot()`. Your labels should have the shape (batchsize, 10) in the end.

[4]For that check out 'tf.keras.layers.Dense(units= , activation=)'. It is basically the same layer that we implemented by hand last time. For activations functions check out 'tf.keras.activations'.

[5]Check out 'tf.keras.losses'.

[6]Use the notebooks uploaded by us as orientation, e.g. the MNIST notebook.

```
def onehotify(tensor):
  vocab = {'A':'1', 'C': '2', 'G':'3', 'T':'0'}
  for key in vocab.keys():
    tensor = tf.strings.regex_replace(tensor, key, vocab[key])
  split = tf.strings.bytes_split(tensor)
  labels = tf.cast(tf.strings.to_number(split), tf.uint8)
  onehot = tf.one_hot(labels, 4)
  onehot = tf.reshape(onehot, (-1,))
  return onehot
```