Homework 2

Implementing ANNs with TensorFlow

05.11.2018

This exercise will be your first implementation of a neural network with TensorFlow. Submit deadline is 11.59 am on Monday, 05.11.2018. Please run your notebook in the end once from top to bottom (restart kernel & run all) `and` download it as notebook/*.ipynb and as HTML/*.html. Upload both files to your homework folder on Stud.IP. You will be assigned a group to correct in the next week on the next homework sheet.

You will build a neural network that will learn to classify the MNIST dataset. As this is your first implementation I will give you step for step instructions. I will not give you all the code, but I will state which TensorFlow functions you will have to use. Look either at the TensorFlow introduction lecture to see how to use them (a lot of the code will be very similar/the same as in the main example from the lecture). Otherwise you can always have a look at the TensorFlow documentation for further explanation.
I guess that this first programming homework might be hard for some of you, so please do not hesitate to write me a mail if you have questions.

1 Preparing the dataset

1.1 Download the MNIST dataset

We already briefly talked about the MNIST dataset. Please download it on the following website: http://yann.lecun.com/exdb/mnist/ .
You have to download all four files and extract them. Put them wherever you want. I recommend to put them into a folder called 'MNIST' and to put this folder into the same one as your jupyter notebook for this homework.

1.2 Jupyter Notebook

Now you can open a jupyter notebook in the specified folder. You need to import four modules:

```
import numpy as np
import tensorflow as tf
import struct
import matplotlib.pyplot as plt
```

1.3 Create NumPy arrays out of the data

The data comes in the idx format, but we would like to use it as numpy arrays. On Stud.IP you find a python file "read_idx.py" in which you will find a function `read_idx(filename)`. Copy paste this function into your jupyter notebook.
Now use this function to read in the training input, training labels, validation input and validation labels (the validation dataset is called 't10k...').
E.g.: `training_data = read_idx('./MNIST/train-images-idx3-ubyte')`

1.4 Investigate the data

Before you work on building a classifier for a dataset you should always have a look at it yourself. Plot the first 10 images from the training data. You can do this however you want.

I recommend you start by creating a subplot figure:
`fig, ax = plt.subplots(1,10,figsize=(10,10))`.
Then you can use a for-loop to loop through the components of `ax`.
E.g.: `for i in range(10):  ax[i].imshow(...)`

1.5 Create the dataset

Functions: `tf.reset_default_graph, tf.data.Dataset.from_tensor_slices`,
Dataset method: `batch, shuffle`.

Before you use any TensorFlow function once reset the default graph. If you want to rerun later cells, you might need to start here, because the graph has to be cleared.
Now create the training and the validation dataset.
Then specify the batch sizes for both datasets. For the training dataset we will use batch size 128. For the validation dataset we can use all 10000 validation samples at once.
Shuffle the training dataset.

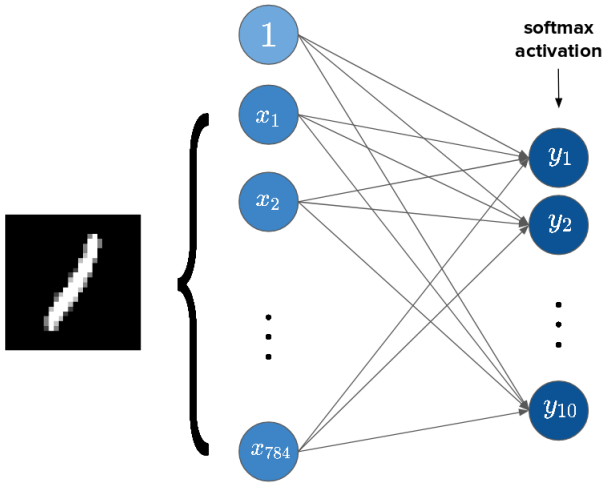
1.6 Create an iterator

Functions: `tf.data.Iterator.from_structure`, Dataset attributes: `output_types, output_shapes`, Iterator methods: `get_next, make_initializer`.

First create an iterator. After that you can name the operation that will give you the next batch from iterator like `next_batch`. We will use the same iterator for both the training dataset and the validation dataset. So we need two initializer operations, that you can also name already (e.g.: `training_init_op`).

2 Building the model

We will use a very simple no hidden layer feed forward neural network with 784(=28*28) input nodes and 10 output nodes. As an activation function we will apply the softmax activation. As loss we use cross entropy.



2.1 Formatting the data

Functions: `tf.reshape, tf.cast, tf.one_hot`.

If you called the operation for getting a new item from the iterator `next_batch` you can now define:
`input_data = next_batch[0]`
`labels = next_batch[1]`
Our input data has the following shape: [batch_size, 28, 28]. We want the input to be only one vector. So we want to reshape to [batch_size, 784]. To keep the first dimension as it is call `input_data = tf.reshape(input_data,shape=[-1,784])`.

Our labels come as single values 0,1,2,... . But for our model we need them as one hot vectors, so you have to transform them.
Additionally you might need to cast the input_data and/or the labels to other datatypes (e.g. tf.float32). You will figure this out yourself after getting errors.

2.2 Define the forward step

Functions: `tf.Variable, tf.random_normal, tf.zeros, tf.matmul, tf.nn.softmax`.

You need to define two variables. One for the weights and one for the biases. Initialize your weights randomly drawn from a normal distribution with mean 0 and standard deviation 2e-06. Contrary to the formal definition in the lecture we will not multiply the weights with the input but the input with the weights. Initialize your biases with zeros. Lastly define the drive and the output. In TensorFlow the drive of the last layer is often denoted as `logits`. As output function we use the softmax activation function.

2.3 Metrics

Functions: `tf.nn.softmax_cross_entropy_with_logits_vs, tf.reduce_mean, tf.equal, tf.argmax`.

First we define the loss: we use cross entopy loss. If you use the above mentioned function be careful to use the logits as an input and **not** the output. If you print the shape of the resulting tensor you will see that it has as many entries as the batch size; one cross entropy value for each training sample. To obtain one value as a loss you take the mean of the cross entropies (`loss = tf.reduce_mean(cross_entropy`).
Now we define the accuracy. You can use the `tf.argmax` function to obtain the guess of the model from the output (specify the correct axis for tf.argmax). With `tf.equal` you can compare it to the labels (the original ones not the one-hot labels). Again you take the mean to obtain one value.

2.4 Optimizer

Functions: `tf.train.GradientDescentOptimizer`, Optimizer methods: `minimize`

First we define the learning rate as 1e-5. Now define optimizer and define a training step as minimizing the above defined loss.

2.5 Summaries

Functions: `tf.summary.scalar, tf.summary.merge_all, tf.summary.FileWriter`

Add summaries for the loss and for the accuracy. Merge them and define two file writers. One for your training metrics and one for the validation metrics. Set the parameter `flush_secs` for the validation file writer to something small like 2, otherwise you might see only one value for the validation loss/accuracy.

3 Training

3.1 Prepare the training

Functions: `tf.Session, tf.global_variables_initializer`,
Session method: `run`.

First define the number of epochs as 5. Afterwards start a TensorFlow session. Initialize all variables once. For saving the summaries we need a global step counter. (`global_step = 0`)

3.2 Define the epoch loop

Use a for loop to run through the cycle of training and validation as often as it is defined by the number of epochs.

3.3 Training Part

Session method: `run`, File Writer method: `add_summary`.

First initialize the iterator with the training data for the upcoming epoch. Afterwards use the `while True, try, except` construct (see the notebook 03 from the lecture) to go through all batches once. For each batch first run the training step and read out the summaries. Then save the summaries using the training file writer. Remember to include the global step number here. And increment the global step with one.

3.4 Validation Part

Session method: `run`, File Writer method: `add_summary`.

Initialize the iterator. Read out the summaries (do **not** train here) and save them using the validation file writer.