EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATH. - NATURWISS. FAKULTÄT
FACHBEREICH INFORMATIK
KOGNITIVE SYSTEME · PROF. A. ZELL

Deep Neural Networks

Assignment 6

Assignment due by: 19.06.2018, Discussions on: 26.06.2018

This week's assignment will serve as an introduction to *tensorflow*, which is a framework for deep learning developed by Google. As you have probably seen last week, computing gradients by hand is somewhat tedious. We chose *tensorflow* because it is very versatile and has its high-level interface primarily in python, which we have used so far in this course.

While developing something with tensorflow is mostly done in python, none of the numerically intensive computations are done by python itself as that would be far too slow (even with a library like numpy which is already much better than native python maths). Instead tensorflow operates on the principle of a computational graph. The code written in python only describes a series of computations to be applied to some given input data, but to run this computation it is handed off to some much more optimized code (potentially running on a GPU) which will complete the full computation and hand the result back to python. This has the benefits of the ease of development in python without sacrificing performance.

In this course, we want to use a low level interface of *tensorflow* because it makes it easier to see the mechanics behind the networks. With this knowledge it should be easier to pick up the higher level interfaces *tensorflow* provides.

Question 1 Getting started with tensorflow (10 points)

Your first task is to set up tensorflow on your own pc/laptop. You can find the instructions on how to do this at `https://www.tensorflow.org/install`; we recommend that you install the cpu-only version, since setting up CUDA can take much more time and can have many problems.

The low level API tutorial (`https://www.tensorflow.org/programmers_guide/low_level_intro`) gives an overview on how to construct and run a computation and simple models. Also read through the two subsequent tutorials on variables and tensors. We have prepared a file (`tf_hands_on.py`) with some small exercises to get you used to tensorflow's conventions. For each section, you should create the necessary operations to have tensorflow perform the desired computation and then test that it is performed correctly.

Question 2 A simple neural network in tensorflow (4+2+2+2 points)

We have prepared a file (`nn.py`) which uses tensorflow to implement a simple linear model for classifying the MNIST which we have already use in Question 2 of the last assignment. Please make sure you understand what the code is doing (or ask on ILIAS if something is unclear). The accuracy of this simple linear model is not very good, so we will try to improve it in this question by modifying the code:

(a) Introduce a hidden layer into the neural network. This means that we create a second set of weights and biases and our network should compute the following function:

$$\boldsymbol{S}(f(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2)$$

instead of:

$$\boldsymbol{S}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$$

which the linear model used ($\boldsymbol{S}$ is the softmax function). In addition to the extra bias and weights, the new classification function also has a function f in it. f is called an activation function and is there to prevent the model from degenerating to a linear model (think about why). The usual choice for f is the rectified linear unit (ReLU, $f(x) = \max(0, x)$, `tf.nn.relu()`) or historically $\tanh$/sigmoid and is applied element-wise.
We have to make a choice about how large the hidden layer should be (the size of $\boldsymbol{b}_1$ and the second/first dimension of $\boldsymbol{W}_{1/2}$) since it is not determined by the input or output size. For now choose the size to be 100.
Lastly, two training details: due to the increased complexity, you should increase the number of training iterations and the weight variables $\boldsymbol{W}$ should no longer be initialized to zero, but instead with small random values (e.g. using `tf.truncated_normal`).

(b) Vary the size of the hidden layer and see how this affects the accuracy of the model.

(c) Introduce a second hidden layer (or even more) into the model and see how high you can get the accuracy while keeping the training time below one minute.

(d) Now see how accurate you can make the model with the restriction that the total number of neurons in all hidden layers can be no more than 200. Try and document at least five different configurations.