Models of Higher Brain Function
Computer Course
Week 1

Lecturer: Prof. Dr. Henning Sprekeler

Assistants: Mathias Schmerling
(mathias.schmerling@bccn-berlin.de)

Visual perception

April 19, 2018

The solutions for these exercises (comprising source code, discussion and interpretation as an IPython Notebook) should be handed in before **April 26, 2018 at 10:15 am** through the Moodle interface (in emergency cases send them to mathias.schmerling@bccn-berlin.de).

Exercise: Building translation-invariant networks in Tensorflow

Today's Deep Learning software packages enable the easy construction and training of translation-invariant hierarchical neural networks. In this exercise, we will use Tensorflow for its simplicity, easy installation and documentation.
Have a look at https://www.tensorflow.org for installation instructions. Install the appropriate Python3 CPU-only version. We highly recommend using an Anaconda-based installation. If you run into problems installing Tensorflow, we can assist you in the first tutorial on **April 23**.

Part 1: Simple MNIST

1. Familiarize yourself with the package by going through the MNIST tutorial[1] on convolutional neural networks. The code of that tutorial implements a convolutional feedforward neural network to classify handwritten digits.

2. Adapt the function `cnn_model_fn` to instead implement a linear classifier called `linear_model_fn`. A linear classifier is a neural network with one dense layer instead of multiple convolutional layers. Given a vector $x \in \mathbb{R}^{784}$ which represents the pixel intensities of an input image of size 28×28, it should compute its output $y \in \mathbb{R}^{10}$ as

   $$\mathbf{y} = \mathbf{W} \cdot \mathbf{x} - \mathbf{b}$$

   where $\mathbf{W} \in \mathbb{R}^{10 \times 784}$ and $\mathbf{b} \in \mathbb{R}^{10}$ are learnable weights and biases. The existing code will take care of learning these parameters as well as some other functionalities such as evaluating performance. Train the model for 1000 steps with `tf.train.GradientDescentOptimizer`, a learning rate of 0.5 and a batch size of 100.

3. Plot a few sample MNIST images (`matplotlib.pyplot.imshow`). Plot the class-dependent weight vectors (receptive fields) after learning as 28×28 images. Calculate how many parameters the model has. Alternatively extract the number of parameters from the code (e.g. with `mnist_classifier.get_variable_names()` and `mnist_classifier.get_variable_value()`). Finally, report the classification accuracy on the test set.

Part 2: Translated MNIST

We are going to test our models on a more complicated dataset now. Look at the code snippet provided with this exercise, which transforms the original dataset by placing the original digits at random locations in a wider image frame of size 40×40.

1. Plot a few sample images from the new dataset.

2. Train the linear model from Part 1 on the new dataset. Keep everything else fixed. Report the final test accuracy and plot the weight vectors as before. Discuss differences to the weight vectors from before. *(Hint: Use* `tf.reset_default_graph()` *in-between models/experiments).*

3. Next, we are going to adapt the model to include a hidden layer and call it `mlp_model_fn`. The hidden layer should have 500 units and a rectifying nonlinear activation function (`tf.nn.relu`). Use the following learning parameters: learning rate 10^{-4}, batch size 50, the `tf.train.AdamOptimizer`, 10000 update steps. Report the final classification accuracy on the test set. How many parameters does the new model have?

4. We now want to use a convolutional architecture as in `cnn_model_fn`. The model proposed in the tutorial is already quite big and would take on the order of hours to train, depending on your hardware. Therefore, you will need to simplify the model. You can e.g. drop one of the convolutional/max_pooling layers, drop the dropout layer, reduce the number of training steps, reduce the number of classes you want to classify and/or reduce the batch size. Keep the learning rate at 10^{-4} and use `tf.train.AdamOptimizer`. Report the classification accuracy on the test set.

5. How many parameters does your model have?

6. Choose a few sample input images and compute the activations of (one of) the convolutional layers when feeding these images into the network. To this end, you should use `mnist_classifier.predict()` which is the last of the three modes that the model can operate in. Figure out which parts of the code need to be modified to accomplish this. Find an interpretable way to plot the activations.

7. Regardless of whether you actually managed to successfully train the convolutional model, which model, the densely connected model or the convolutional model, do you expect to perform better on the translated MNIST dataset? In your own words, what is the main difference between the models and why does one of them have an advantage over the other for translated MNIST?

8. *(Optional)* Many of today's state-of-the-art models require **a lot** of training samples. Maybe you ran into this problem when training the convolutional model as well. While you are training a model, how could you decide if you should stop the training or if the model is still improving? Hypothesize about a stopping criterion.

[1] `https://www.tensorflow.org/tutorials/layers`