Introduction to Neural Networks 67103

Assignment 1: CNN

Due: 3/12/2017

In this assignment you will implement a simple Convolutional Neural Network (CNN), train it to perform a simple classification task, and experiment with different architectures and hyper-parameters. Your implementation and experiments should be done using TensorFlow. As a basis you should start from the simple CNN-based MNIST classifier described in this tutorial.

A. **Train and Test Error.** The provided simple CNN has two convolution layers (each followed by pooling) and two fully connected layers, and it is able to achieve an accuracy above 99.2% after 20000 iterations. How many free parameters does this network have? The training process is often visualized/monitored by plotting the training accuracy and the test accuracy as a function of the number of training iterations. Plot these accuracy measures (perhaps using a data point every 500 iterations).

B. **Linear vs. Non-Linear**. Identify what layers/components of the network are non-linear and remove them (except for the logits/softmax in the loss function, also assume the max-pooling is a uniform stride - or change it to one if you wish). Compare the performance of this linear network to its original non-linear counterpart. What is the effect on the training and the final test accuracy of the network?

C. **Deep vs. Shallow**. Construct a shallow CNN consisting of a single convolutional layer, followed by a fully connected layer, such that it is able to achieve accuracy above 95% using the smallest number of free parameters. How many parameters does your network have (explain how you count them), and how quickly did it reach the target accuracy? Plot the training of this network. Next, reduce the number of parameters of the original deeper network as much as you can, while still being able to achieve 95% accuracy. How many parameters does this network have (detail the count), and how quickly did it reach the target accuracy? Plot the training of this network.

D. **Overfitting.** Train the original network (without dropout, i.e., keep_prob in the code should always be set to 1.0) over the complete set of the examples as well as a reduced set of only 250 examples. Plot the training and testing accuracies as well as their ratio. In order to reduce the over fitting of the reduced training dataset, reduce the network's size (fewer parameters) until a higher ratio and test accuracy is obtained. Report the reduction and size and gain in performance in this experiment. As an alternative insert **dropout layers** to the outputs of all the hidden layers. Report which keep-probability resulted in the best ratio/test accuracy. **Note:** the dropout layer randomly blocks neurons by setting them to zero and transfers the rest at probability *keep_prob*. It is used to cause the network to learn the filters with redundancies (let one neuron compensate for the possible loss of others), thus effectively reducing the number of

neurons in the network, or alternatively reducing the number of "free" parameters. We will discuss this layer in a future class.

E. **Two Digits Sum.** Design, implement and train a CNN architecture that takes as input two MNIST images and infers their sum. Note that this should be solved as a pure classification problem, and not by running the single image MNIST classifier twice and adding the results. Describe the reasoning behind your choice of architecture. Plot the training of this network.

Theoretical Questions

1. Convolution. Show that convolution is indeed a translation-invariant operation, i.e., (f[k+t]*g[k])[n] = (f[k]*g[k])[n+t] and (f[k]*g[k+t])[n] = (f[k]*g[k])[n+t]. Express the convolution operation as a dot-product (and its translations). When convolving a signal of length n with a filter of length k<=n and discarding all the cases where the signal is not fully included in the support of the filter, what is the length of the output signal? What would that be when n=k?

2. ReLU. Assume the filter f and the bias b produce a negative response on an example datapoint I, i.e., (f*I)[n]+b<0 for all n. Assume the next layer is a RelU layer, followed by the remaining layers and loss, expressed together by F(.), i.e., the network's loss is given by F(RelU(f*I+b)). What would be the partial derivative of this loss term with respect to f and b? What would happen to these filters and biases in a gradient-descent process consisting solely on examples which produce negative responses.

3. Backpropagation. Compute the partial derivative of the convolution layer, f*I with respect to the filter f (express the complete matrix whose columns correspond to the filter elements, and rows to the output image pixels - row per pixel - big matrix!). Explain how this matrix is efficiently multiplied by the gradient vector of the previous layer being propagated through this layer. You may assume 1D signals, and that the convolution layer discards points in which the filter f is not fully included within the support of the signal I.

4. Parameters vs. Constraints. How many parameters are involved in a convolutional layer (+ its biases) consisting of 96, 5-by-5 pixel filters, that operate on an RGB image? How many parameters are involved in a fully-connected layer (plus its biases) mapping 4-by-4 images of 256 channels (reshaped into a vector) to the same dimension? Assuming each loss terms is fulfilled, i.e., rather than minimizing the loss, we set each loss term as an equation (or constraint). How many constraints a dataset of 10000 images with 1000 categories produce?

Submission and Grading

Submit (via moodle) a ZIP or RAR archive that contains your code, and a document (pdf only, no doc/docx!), which clearly describes your solutions and reports your results.

Grading: exercises which provide complete and correct answers and working solutions that satisfy the exercise requirements will result in a grade up to 95. A bonus of up to 5 points may be provided to

solutions that use the fewest parameters, or converge the quickest.