CS 447/547: Assignment 3

Due May 3rd, 11:59 PM.

(This document was last modified on Saturday, April 27, 2019 at 02:43:34 PM.)

For this assignment, you will implement a neural network (NN) in CUDA to recognize the MNIST handwriting data set. If you are feeling ambitious, you may attempt a convolutional neural network (CNN). If you are feeling less so, you can get by with a fully-connected NN.

Use floats instead of doubles. It will run faster, and the accuracy is just as good.

You may find the materials here here to be helpful.

You may also find the CUDA Thrust library to be helpful.

If you have a Nvidia card in your desktop, you can develop on that.

A sequential implementation is here.

A more straightforward, and hopefully easier to understand, example is here. This second example only computes the softmax and cross-entropy layer, but those are actually harder to understand than the linear layer. It also shows how to use discrete difference approximation to check your code. Start by looking at `main()`.

Convolutional Neural Network

The suggested CNN architecture is the one used in this TensorFlow tutorial. You are free, however, to go beyond the architecture in the tutorial, or tackle other datasets such as CIFAR-10. Check with me first, though.

You may find it handy to generate images of activation maps, etc. I suggest that you output them in PGM or PPM format, which is a simple ASCII format. There are lots of tools to either view images in this format, or convert to other lossless formats such as PNG (or lossy ones such as JPEG).

Layers

Input Layer (IL)

The input layer is a 28×28×1 grayscale image.

Convolutional Layer 1 (CL1)

The first convolutional layer contains 32 5×5×1 filters, with ReLU activation. Pad the borders with 2 pixel wide strip of white (0), so that the output of each filter is the same size as the input. The output of this layer is 28×28×32, because each filter creates a new plane in the 3rd dimension.

Pooling Layer 1 (PL1)

The first pooling layer will perform max pooling with a 2×2 filter and stride of 2. Pooling regions will thus not overlap, and this will downsample. The output of this layer will be 14×14×32.

Each plane is pooled independently. In other words, each new pixel is the max over a 2×2 region in a single plane. You can also think of this as pooling over a 2×2×1 region. This is then repeated for that same position, for each depth.

Convolutional Layer 2 (CL2)

The second convolutional layer contains 64 5×5×32 filters, with ReLU activation. Pad the same as CL1. The output of this layer is 14×14×64, because each filter creates a new plane in the 3rd dimension.

It's important to note that the convolution is performed over the entire depth.

Pooling Layer 2 (PL2)

Same as PL1, but it will again downsample. The output of this layer will be 7×7×64.

Dense Layer 1 (DL1)

This is just a fully connected layer with ReLU activation, but we will stick with the TensorFlow terminology and call it a “dense layer”. It consists of 1024 neurons with dropout rate of 0.4. This means that during training, any given neuron has a 0.4 probability of “dropping out”, which means that it is set to 0, regardless of the inputs.

Note that the TF example first reshapes the incoming tensor. Because you are implementing it directly, you do not necessarily need to do this, but may find that there are CUDA performance benefits.

Dense Layer 2 (DL2)

This is the final classification layer. It is a fully connected layer consisting of 10 neurons, one for each class. It will compute a softmax. Note that softmax may have overflow/underflow issues, so guard against that. See this. Scroll down to "Computing softmax and numerical stability".

A Simpler Network

The deep CNN is challenging. If you use just a fully-connected network (FCN), you can also get over 90% accuracy. The layers you need are given below.

Layers

Input Layer

The input layer is a 28×28×1 grayscale image.

Fully Connected Hidden Layer

This is just a fully connected layer with ReLU activation, but we will stick with the TensorFlow terminology and call it a “dense layer”. It consists of 1024 neurons with dropout rate of 0.4. This means that during training, any given neuron has a 0.4 probability of “dropping out”, which means that it is set to 0, regardless of the inputs.

Note that the TF example first reshapes the incoming tensor. Because you are implementing it directly, you do not necessarily need to do this, but may find that there are CUDA performance benefits.

Output Layer

This is the final classification layer. It is a fully connected layer consisting of 10 neurons, one for each class. It will compute a softmax. Note that softmax may have overflow/underflow issues, so guard against that. See this. Scroll down to "Computing softmax and numerical stability".

Training

For training, use cross-entropy on the softmax as the loss function, and stochastic gradient descent (batched backprop). The learning rate used in the TensorFlow tutorial was 0.001. The batch size was 100.