Reactive Research Software Engineer Assignment

Andrew Gambardella

May 3, 2016

For this assignment, I experimented with many CNN architectures for handwritten Chinese character recognition. My final model was a 7-layer network with gradually decreasing filter size and gradually increasing layer depth. Strided convolutions were used in place of max-pooling for dimensionality reduction. All training was done on the CPU. The model achieves 4.86% test error on unseen data.

Intuition behind CNNs

Each convolutional layer of the CNN has 3 dimensions – height, width, and depth. The convolutional layers consist of a set of learnable filters, which are convolved with the height and width dimensions of the input. This is accomplished through a weight-sharing scheme such that all neurons in a single "depth slice" share the same weights and biases. The network will learn filters that activate upon seeing certain spatial features. In the lower layers of a CNN, usually these turn out to be low-level features (e.g., lines, colors) whereas in the higher layers, they turn out to be high-level features (e.g., eyes for networks trained on faces or windows for networks trained on bedrooms).

Deciding on the structure and hyperparameters

I initially started with the CNN tutorial that is provided with TensorFlow. This is an implementation of LeNet-5, trained on MNIST. After having modified it to take the Chinese character dataset as input, the initial hyperparameter settings produced a model with 14.2% validation error. Random search over hyperparameters reduced this to 8.9% validation error.

Inspired by many recent papers such as Radford et al. (`http://arxiv.org/abs/1511.06434`) in which CNNs are trained with strided convolutions and no fully-connected layers, producing more stable training with fewer parameters, I next removed one of the fully connected layers. This gave a model that achieved 11.1% validation error, but with approximately one-half of the network parameters.

This model was underfitting horribly. From here on out, I kept making the network bigger, both in terms of depth and by adding more layers. My final model (validation error 5.3%) had 7 layers, 6 of which were convolutional. The final layer was a fully-connected layer, used to project the convolutional layer features onto the 100 labels. This model was still underfitting with a dropout rate of only 25%. I wanted to continue to make the model bigger and use more aggressive regularization, but did not have the time to do so (training on a CPU, my final model took several hours to reach convergence, making further prototyping on bigger networks difficult).

Hyperparameter selection was done at first by random search with a small network, and then adjusted as needed as I made the network bigger. In particular, as the network got bigger, I found it necessary to reduce the base learning rate several times. 32x32 pixel images, preprocessed with pixel values in the range [-0.5, 0.5] were used for all experiments. I used a batch size of 64 and the Adam optimizer. Weights for all layers were initialized from a gaussian distribution with standard deviation of 0.1, as was done in the MNIST tutorial. ReLU nonlinearities were used in all layers, and all convolutional layers (with the exception of the 1x1 convolutions) had a stride of 2 in both directions.

20% of the data was held out for the test set, 16% of the data was held out for the validation set, and 64% was used for training. Once the final model was selected, it was retrained on both the training and validation sets, and evaluated on the test set. The data was shuffled before splitting, to ensure that classes were approximately balanced in all three sets. During training, the training set was also shuffled once at the beginning of each epoch.

The hardest and easiest parts

I would say that the hardest part of this assignment was knowing whether or not my results were reasonable. If I had created a model that achieves 4.86% test error on MNIST, that would be horrible, but there are 10 times as many classes in this dataset and 1/10 of the data for each class. With that in mind, 4.86% does not seem too horrible, but my job would have been a lot easier were I provided with other benchmarks with which I could measure my progress.

Another difficult part was deciding what I should do to improve on the error rate. For example, if I'm underfitting it means I should make the network bigger, but there are many choices for where to insert new parameters (e.g., convolutional layer depth, more layers, completely different architecture), and some would be more effective than others.

The easiest part was getting a reasonable(-ish), working model in the first place. Most of my work training neural networks up until now has been with RNNs, which are much more difficult to even get training in the first place. For this task, however, using the MNIST tutorial with no changes at all produced a model that worked somewhat reasonably well.

What I enjoyed

This was actually my first experience training a CNN from scratch (I had always before used pre-trained networks as fixed feature extractors, or fine-tuned a pre-trained network) so it was nice to have the experience of deciding the structure of the network myself.

Other than that, I appreciated having an interview that tested more practical skills than, say, coding on a whiteboard.

Improvements

My final model only used a dropout rate of 25% – this is relatively weak. I am fairly confident that the error rate could be reduced by making the network bigger and using much more aggressive regularization (possibly somewhere between 50% and 75%). I had also tried using batch normalization instead of dropout briefly, but it did not seem to work as well (possibly because I had first attempted to use it on a very small network). I would have liked to try using batch normalization again with a bigger network to see how it performs.

I also could have removed the final fully-connected layer, which, according to Springenberg et al. (`http://arxiv.org/abs/1412.6806`) would have reduced my error rate further. After reshaping the final convolutional layer, however, I was unsure of how to project those features onto the 100 classes of the dataset without using a fully-connected layer (possibly, I could have arranged my convolutional layers such that in the final layer, depth*height*width was equal to 100, but I did not attempt this).

I also did not attempt to use anything other than a "feed-forward convolutional network" (e.g., I did not try using inception modules or anything like that). These could have potentially helped greatly.

Business applications and more interesting problems

At SoftBank Robotics, I was developing a robot that would interact with customers in stores. There I developed a CNN-based application that allowed our robot to identify objects with a camera image, rather than a barcode.

In general, CNNs could be used whenever a user is interacting with image data, and needs to do some automated process based on that data. For instance, automatic labelling of images, or finding similar images in a collection.

As for more interesting problems, I am looking forward to seeing the emergence of unsupervised learning. One of the papers I linked earlier is about an architecture called a "Deep Convolutional Generative Adversarial Network," which can learn to generate its own, completely original, images by training a generator network, which makes images, and

a discriminator network, which tries to classify images as real or fake, in parallel. They got really interesting results, and I'm very excited to see where things go from there.