CIFAR10 classification task

for the Neural Networks course, winter 2015

Maciej Pawlikowski

Motivation

The task required making a classifier that will achieve less than 25 percent errors on the CIFAR10 dataset. The dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The proposed solution is based on convolutional neural networks. This approach tends to be successful when it comes to image classification, because we can use convolutional layers to extract features that will tell us more about a picture than just single pixels. The way it works resides in a very nature of a convolution operation, as each point of a convolved image contains information about a neighborhood of a corresponding point in the base picture. This fact is very useful, because before that the network had no information about the location of each of the input points. Now it knows which ones are positioned next to each other, which can hopefully help it to treat the input more like an image and less like just a set of numbers.

The network

I used the starter code from the lecture about convolutions.

The architecture of the network is as follows:

1. Convolutional layer with 40 3x3 filters followed by 3x3 max pooling and ReLU activation (at this point each 32x32 input image is transformed into 40 10x10 feature maps);

2. Convolutional layer with 50 3x3 filters followed by 2x2 max pooling and ReLU activation (now we have 40*50 4x4 feature maps out of each input image);

3. Fully-connected layer with 500 neurons activated by ReLU;

4. Fully-connected layer with 10 neurons followed by Softmax.

In the original code from the lecture filters in the first two layers were 5x5. Since I didn't know why they were chosen to be 5x5, I tried to experiment with their sizes. I thought they can't be too big, because the input images are only 32x32, which means that a big filter could lead to losing some details by adding together pixels that are too far apart in the original image. Because the images were small, I decided to try out smaller filters in a hope for better recognition of the contents of the pictures. 3x3 filters gave slightly better accuracy than 5x5, so I stuck with them.

I also increased the sizes of the convolutional layers. I didn't get significantly better results by making them larger than about 50 filters each. Increasing them to 40-50 range caused accuracy to grow by about 2 percentage points though. The training time is obviously rising with each increase, so I used the sizes listed above.

The learning procedure uses Stochastic Gradient Decent with standard improvements:

- momentum (We distribute each update of the network's parameters among several learning steps, which helps in faster converging);

- weight decay (We penalize huge weights by adding the norm of parameters to the cost function, so that the network can hopefully find the most important parameters, as most of the weights at the end of learning process should be close to 0);

- learning rate scheduling (The longer the learning process goes the lower the learning rate is, which helps in more fine-grained searching for the local minimum during the later parts of learning process).

I didn’t change the momentum value or the weight decay parameter, after experimenting with them my results only got worse. Adjusting learning rate proven very helpful though. I got better accuracy by almost tripling the initial value and making it progressively decay after 5000 samples.

The above approach yielded slightly more than 75 percent accuracy on the CIFAR10 test set. Given the fact that the network is still relatively small and very few changes were made compared to the starter code, I think this is a nice result.

Total training time: 555 seconds (42 epochs)

Best result achieved after epoch 27.

Test set accuracy: 75,64 percent

Train set accuracy: 99,8 percent

However, the network suffered from enormous amount of overfitting. During the later stages of training process the accuracy on the train set was nearly 100 percent. Apparently, simple weight decay technique was not sufficient to properly regularize the network. I made some attempts to regularize it further.

Improvements

1. Dropout

My first thought was to add dropout. Dropout is a very popular and easy to implement trick for preventing network overfitting. We want to randomly disable some neurons (usually about 50 percent of them) during each training step (and enable them again afterwards). That will allow us to basically train a large amount of models at the same time and average them out into a final solution. It is computationally efficient; in fact, we have to work less, because half of the neurons are turned off.

When I added 50 percent dropout after the first fully-connected layer, the accuracy raised by about 2 percentage points. I also experimented a bit with dropping neurons in other (convolutional) layers, but it didn't do any good, the accuracy didn’t improve at all.

Using dropout usually means that the network will take much longer to train, as the weight updates are more chaotic (we train different subset of neurons each time). In our case the number of epochs necessary for the net to learn increased significantly. It is worth noting that the lengths of training sessions started to vary greatly after applying dropout after the first fully-connected layer. Results of the two training sessions I've done:

Total training time: 969 seconds (73 epochs)

Best result achieved after 48 epochs.

Test set accuracy: 77,31 percent

Train set accuracy: 97,04 percent

Total training time: 2882 seconds (148 epochs)

Best result achieved after 98 epochs.

Test set accuracy: 77,91 percent

Train set accuracy: 99,16 percent

While increasing accuracy on the test set, dropout didn't quite solve the overfitting problem. The train set error was still very small.

2. Artificial data

Simple way of decreasing overfitting is to just get more data. More data means that more complex model is necessary to remember all of it. In case of pictures we can easily create artificial data by rotating of shifting existing images: neither rotate nor shift changes what object is in the picture, so the labels will stay the way they are.

I tried to get less overfitting by doubling the training data. Every picture in the data has been duplicated and then shifted by a small random number of pixels on both axis. So half of the new data consists of the original dataset and the other half is generated. This resulted in an increase of both accuracy and training time.

Total training time: 5034 seconds (manually stopped after 205 epochs)

Best result achieved after 205 epochs.

Test set accuracy: 80,59 percent

Train set accuracy: 95,61 percent

I also tried a different transformation of data. This time the second half of data was created by flipping images horizontally. This approach gave very similar results, so I decided to merge it with the previous one. To cut on the training time, I took the original dataset and flipped 30% of images chosen randomly. Then I shifted another random 30% of the original data (like in the previous attempt). So now we have the new dataset, 40% of which is original data and 60% consists of transformed images. Unfortunately I wasn't able to compare training time to the ones above, because I couldn't find a free lab computer. Each epoch should be shorter though, as the training set is now of the original size instead of being doubled. Results:

Total training time: manually stopped after 200 epochs

Best result achieved after epoch 168.

Test set accuracy: 81,17%

Train set accuracy: 91,75%

Conclusions

Regularization plays a very important role in the neural network training. In our case we see that to make the network generalize better we didn't really need more training examples. Just adding some noise to the part of the dataset caused much better accuracy and a major decrease of overfitting. The disparity between the effectiveness of the network on the training and testing data is not so extreme anymore (although it certainly can be decreased even further). Very nice improvement to the network would be the shortening of training time, something that the current method clearly has trouble with.