1 Regularization of Neural Networks using DropConnect

This paper from 2013 describes the current lowest error rate performance on the MNIST data set (0.21%).

DropConnect is the generalization of Dropout in which each connection, rather than each output unit, can be dropped with probability $1 - p$. It randomly drops weights rather than activations. This typically converges slower than Dropout, but improves results in the end.

Paper found here.

Explanation and code found here.

Their best results on the MNIST dataset is achieved by first creating more data with the following steps:

1. Scale pixel values to [0, 1] range.
2. Take cropped 24×24 image patches from the original 28×28 images at random.
3. Flip images horizontally.
4. Create rotated and scaled versions of these cropped images (15% scaling and rotation variations).

They used an initial learning rate of 0.01 with a 700-200-100 epoch schedule, no momentum and preprocess by subtracting the image mean. They manually decreased the learning rate if the network stopped improving according to a schedule determined on the a validation set described in this paper. Here's an MLM guide on learning rate schedules in Keras.

All experiments use mini-batch SGD on batches of 128 images. Five independent networks with random permutations of the training sequence are then trained on the images:

1. Convolutional layer with 32 feature maps, ReLU activation.
2. Convolutional layer with 64 feature maps, ReLU activation.
3. Fully connected layer with 150 neurons, ReLU activation, 0.5 DropConnect rate.
4. Fully connected layer with 10 neurons, softmax activation.

The precise network is not described in detail, this summary was the best I could make of it. They do not mention max-pooling, which is often used after convolutional layers. They mention elastic distortions, but do not use them. Elastic distortions of images create more data, and reportedly improve performance on some neural networks. Dropout layers are typically used after fully connected layers, but I've read somewhere that using them after convolutional layers might improve performance.

2 Fractional Max-Pooling

This paper from 2015 describes the current 8th lowest error rate performance on the MNIST data set (0.32%).

The most popular pooling technique is with 2×2 filter with stride $= 2$, which has no overlap. In some networks results have been improved with overlapping pooling, such as 3×3 with stride $= 2$. In general, pooling layers have been overlooked: not much research has been done.

Fractional max-pooling (FMP) introduces a degree of randomness to the pooling process. The pooling regions are chosen (pseudo-)randomly, and the re-scaling factor can be a fraction.

The paper is very short, and it is hard to pinpoint how exactly they implemented the pooling layer. I think it is interesting nonetheless, because they do have good results, especially on the CIFAR-10 data set.

Paper found here.

To summarize, this is what they found:

1. Overlapping pooling regions seem to perform better in general.
2. Dividing the original image into rectangular pooling regions using pseudo-randomness seems to perform better when data-augmentation is used.
3. Dividing the original image into rectangular pooling regions using randomness seems to work better when no data-augmentation is used.
4. Random pooling shows elastic distortion, pseudorandom does not.
5. Max-pooling seems to work better than average pooling.
6. The result image is not twice as small as with a 2×2 filter and stride $= 2$, but a fractional factor, or $\sqrt{2}$ times as small.

Not sure how hard this is to implement. If we do not manage to do this, then we should at least take a look at overlapping max-pooling (3×3, stride $= 2$), because it looks promising.

3 Other improvements

Some general improvements to CNN's from the ISMI course:

1. Trying different activation functions than ReLU such as PReLU, LeakyReLU, etc.
2. Use valid convolutions (no padding).
3. Initialize the kernel with Glorot or He to speed up learning.
4. Try L1 or L2 regularization. This might not work too well combined with Dropout/DropConnect because both go against overfitting: we might end up underfitting.
5. Add a batch normalization layer before or after the activation function of a dense/convolutional layer, see what works better. (I've read that adding it after the activation function works better in practice)
6. If we use SGD as an optimizer we should play around with the (Nesterov) momentum parameter. However, I expect we are going to use ADAM.
7. There have been some great results with fully convolutional networks (without dense layers) on the CIFAR10 data set. I'm not sure if it's any good for MNIST because small CNN's seems to give great results, but it might be worth trying.
8. Some networks (such as AlexNet) augment images even when testing, resulting in a small improvement. For example: take 5 patches from the original image, and mirror them left-to-right. This creates 10 images from the original image. We can let the network score each image individually and aggregate the scores to classify the original image.