

# Problem Set 4 (due Nov 7)

## Convolutional Neural Networks

October 26, 2006

In this problem set we will implement a basic convolutional neural network and train it on some artificial data.

### 1. Batch learning on artificial data

Implement a two-dimensional convolutional neural network that has an input layer, a convolutional hidden layer, and an output layer. The output layer should also be convolutional, so that each output unit looks at a patch within each feature map of the hidden layer. Implement the network such that you can easily vary the number of feature maps, the size of the 2-d patch in both the hidden and output layer, and the overall size of the input. While you may implement this using any technique you wish, we strongly recommend that you follow the suggested implementation in the lecture, using MATLAB's built in functions for convolution and correlation.

Use the provided script `fake_circles.m` to generate an image with circles randomly placed on it. Be sure to binarize the image that this script produces. The input to the network will be this image with varying amounts of gaussian noise added to it (i.e., `image + n*randn(size(image))` where `n` may be between 0 and 1). The desired output will simply be the original image, unmodified. Transforming a corrupted input back into itself is a task known as denoising.

For the batch learning case, process the entire image, compute the backprop gradient updates, and then update the weights. Monitor the mean-squared error (MSE) of your network output. If your implementation is correct, this error should go down after each learning update (epoch). Continue learning until the MSE of the network output stops changing.

Vary the amount of noise, patch size, and hidden units. How much noise causes the network to completely fail? Does the size of the patches or the number of hidden units change the results?

### 2. Stochastic online learning on artificial data

Repeat the denoising experiment using online instead of batch learning. One relatively easy way to implement this is to select a random portion of the input (a 10x10 patch, for example) and then update the weights based on the backprop gradient of just that patch. This is called online learning with minibatches, because instead of updating based on just a single example, you are averaging from several examples at once (though not the entire image). This is often a more stable learning procedure than updating after a random chosen single example.

Monitor the MSE of each minibatch. Due to the stochastic nature of this learning algorithm, the MSE may actually increase from one iteration to the next, but on average in the long run it should go down and eventually converge with only small fluctuations. Is learning slower, faster, or the same speed as the batch learning case? Is the final MSE lower than that in the batch learning experiment?