# Notes on neural networks — CIFAR material

Michael Nielsen[1,2]

September 6, 2013

[1]Email: mn@michaelnielsen.org
[2]Web: http://michaelnielsen.org/ddi

# Chapter 1

# Introduction

**Working notes, by Michael Nielsen:** These are rough working notes, written as part of my study of neural networks, especially work on CIFAR. Note that they really are *rough*, and I've made no attempt to clean them up, nor do I plan to. They contain misunderstandings, misinterpretations, omissions, and outright errors. As such, I don't advise others to read the notes, and certainly not to rely on them!

# Chapter 2

# Papers

## 2.1 KSH configuration

Note that the bias initialization parameter initB was not set anywhere in the KSH configuration. That means it defaults to 0.

**Layer 1**

- Convolutional
- 3 channels.
- 32 filters
- Padding of 2. Pads the images on the outside with a 2-pixel border.
- Stride length of 1.
- Filter size is 5 by 5.
- initW=0.0001. The initial standard deviation. I'm surprised by how low this is — much lower than I would have guessed. I wonder if there's any benefit to increasing it?
- partialSum=4. No idea what this means. The docs don't really say.
- sharedBiases=1. According to the docs, "indicates that the biases of every filter in this layer should be shared amongst all applications of that filter." This is a little unclear. Does it mean that all filters have the same bias?

- Fully linear layer.

**Layer 2**

+ Pooling layer + Uses maxpooling + start=0. Where to start pooling. This is just the default, which is to start pooling where you'd expect (the top left). + sizeX=3. Pool 3 x 3 regions. + stride=2. The stride length. + outputsX=0. This is an unimportant default; if not equal to 0 the output would only cover part of the image. + channels=32. Presumably to correspond to the filters in the last layer. + neuron=relu

Layer 3

+ Convolutional layer + 32 filters output, 32 channels input. + 5 by 5 filters. + Stride length of 1 + Initial weight SD = 0.01 + Rectified linear units + sharedBiases=1 + partialSum=4

Layer 4: + Pooling layer + Average pooling + 3 x 3 pooling windows + Stride length 2

Layer 5: + COnvolutional layer. + 32 input channels, 64 output filters + 5 x 5 filters + Padding by 2 pixel border + Stride length of 1 + Initial weight SD = 0.01 + Recitified linear units + sharedBiases=1 + partialSum=4

Layer 6: + Pooling layer, 64 input channels, 64 outputs + Average pooling + 3 x 3 poling windows. + Stride length 2

Layer 7: + Fully connected layer + 64 outputs + Initial weight SD = 0.1 + Rectified linear units

Layer 8: + Fully connected layer + 10 outputs + Initial weight SD = 0.1. + Linear neurons

Layer 9: + SOftmax layer, producing 10 outputs

Cost function: logistic regression on the Softmax outputs.

Learning parameters

Layer 1 (first convolutional layer): + Weight learning rate: 0.001 + Bias learning rate: 0.002 + Weight and bias momentum: 0.9 + Weight decay 0.004. Note there is no bias decay.

Note that in the docs Krizhevsky explicitly gives the update rule:

w' = (weight momentum) * w - (weight decay) * (weight learning rate) * w + (weight learning rate) * gradient

The bias rule is the same, but there is no bias weight decay.

Layer 3 (second convolutional layer)

Same as layer 1.

**Layer 5 (third convolutional layer):** Same as layer 1.

**Layer 7 (first fully connected layer):** Learning rates as for convolutional layers, and weight decay of 0.03

**Layer 8 (final layer)**: Same as first fully connected layer.

Krizhevsky notes that rescaling the overall cost function has the effect of changing the effective overall learning rate.

## 2.2 Wan et al (2013) – "Regularization of Neural Networks using DropConnect

### 2.2.1 Summary of the main points

- Dropout means randomly deleting half the neurons when training.
- DropConnect means randomly deleting half the connections when training.
- Note that the output is defined as the *average* output over the sampled networks, not the full network.
- There is a nice linear algebraic way of representing DropConnect and Dropout, using Hadamard products, which no doubt helps in implementations.
- In actual fact, they don't literally implement DropConnect. Rather, they analyse what the distribution of weighted sums would be, and approximate by a Gaussian, before sampling. I don't see why they do this (it may be faster), but in some sense we can use this as a definition. I'd probably prefer just to sample. No idea why they don't.
- They claim that the regularization is greatly helped by using small mini-batches, ideally mini-batch size 1 (online learning).
- The code is available. They used cuda-convnet for convolutional and softmax steps. The DropConnect implementation is a bit convoluted — worth reading about the problems they had, though. It certainly seems worth storing the masks as bits or ints, not floats.

- Used mini-batch SGD with momentum on batches of 128 images, and momentum fixed at 0.9. Not clear how this relates to the above comments about online learning. They augment the dataset (cropping, flipping, scaling and rotation); train 5 independent network with random permutuations; manually decrease the learning rate using a validation set; train using Dropout, DropConnect, or neither. Use 1,000 samples. Use a bias learning rate twice the weight learning rate. Weights are N(0, 0.1) for fully connected layers, and N(0, 0.01) for convolutional layers.

- The learning schedule is fascinating. "We report three numbers of epochs, such as 600-400-200 to define our schedule. We multiply the initial rate by 1 for the first such number of epochs. Then we use a multipler of 0.5 for the second number of epochs followed by 0.1 again for this second number of epochs. The third number of epochs is used for multipliers of 0.05, 0.01, 0.005, and 0.001 in that order, after which point we report our results. We determine the epochs to use for our schedule using a validation set to look for plateaus in the loss function, at which point we move to the next multiplier."

- CIFAR-10: Subtract per-pixel mean computed over the training set. Then use KSH's 3-layer convolutional net. Follow by 64-unit fully connected layer to which DropConnect etc may be applied. No data augmentation. 150-0-0 epochs, a single model, with an initial learning rate of 0.0001, and KSH's weight decay (0.995, I believe). DropConnect prevents overfitting a little better than Dropout.

- CIFAR-10: More advanced results. Using 2 conv layers, 2 locally connected layers, per KSH. 128 neuron fully connected layer with ReLU activations between softmax and feature extractor. Images are cropped to 24 by 24 to get more data. Initial learning rate: 0.001, and train for 700-300-50 epochs with KSH's weight decay. Model voting helps a *lot*, getting error rate 9.41 percent. This can be improved to 9.32 percent by using 12 networks.

Add a note: data agmentation works nearly as well. We should push on that.

### 2.2.2 Other notes

"When training with Dropout, a randomly selected subset of activations are set to zero within each layer. DropConnect instead sets a randomly selected subset of weights within the network to zero."

As with Dropout, DropConnect is essentially a method of regularization, to prevent the network from overtraining. "In practice, using these [regularization] techniques when training big networks gives superior test performance to smaller networks trained without regularization."

On Dropout: "Although a full understanding of its mechanism is elusive, the intuition is that it prevents the network weights from collaborating with one another to memorize the training examples."

"Like Dropout, [DropConnect] is suitable for fully connected layers only."

I don't really see why. Does something go wrong if we apply it to a convolutional net? I don't see why something analogous couldn't be done.

We can rewrite Dropout as $a \to \sigma(m \odot (wa+b))$, where $\odot$ is the Hadamard product, and $m$ is a binary mask vector, chosen according to an appropriate Bernoulli distribution. A similarly nice expression can be obtained for DropConnect. (This seems likely to help in implementations.)

**Architecture:** A CNN, followed by a DropConnect layer, followed by a SoftMax, and a cross-entropy loss.

Note that the output value can be viewed as the result of sampling a large number of different (though overlapping) neural networks.

"A key component to successfully training with DropConnect is the selection of a different mask for each training example. Selecting a single mask for a subset of training examples, such as a mini-batch of 128 examples, does not regularize the model enough in practice."

They define the output as the result of averaging over all DropConnected networks. Note that this seems likely to be superior to using the entire network (i.e., with no weights deleted).

They do some odd things involving Gaussian moment matching to sample. I don't see *why* they need to do this, I must admit. But it does give a reasonably nice way of approximating the network. Alternately, one could view it as the definition of DropConnect.

**Q: How do Dropout and DropConnect fare in a sparse network?** My guess is that they'll show very interesting behaviour.

# Chapter 3

# Short reviews: what do we know about nonlinearities?

In this chapter I take a very quick and not in-depth look at what is known about various nonlinearities.

DasGupta and Schnitger (1994): They want to compare activation functions as a function of size and number of layrs. And they want to figure out when two activation functions have essentially the same approximating power. "Our results show that good approximation performance... hings on two properties, namely efficient approximation of polynomials and efficient approximation of the binary threshold." I have a lot of troubel believing the former; I wonder if it is an artifact of their analysis. The latter seems interesting.

Jarrett et al (2009): "We show that using non-linearities that include rectification and local contrast normalization is the single most important ingredient for good accuracy on object recognition benchmarks." ALso, "[H]ow do the non-linearities that follow the lter banks inuence the recognition accuracy. The surprising answer is that using a rectifying non-linearity is the single most important factor in improving the performance of a recognition system. This might be due to several reasons: a) the polarity of features is often irrelevant to recognize objects, b) the rectication eliminates cancellations between neighboring lter outputs when combined with average pooling. Without a rectication what is propagated by the average down-sampling is just the noise in the input. Also introducing a local normalization layer improves the performance. It appears to make supervised learning considerably faster, perhaps because all variables have similar variances (akin to the

advantages introduced by whitening and other decorrelation methods)"

Karlik and Olgac (2009): Investigated a few special cases.

Nair and Hinton (2010): Done in the context of Boltzmann machines. They consider noisy rectified linear units (NReLUs), which have output $\max(0, x + N(0, \sigma(x))$, where $N$ denotes a Gaussian random variable, as per usual. Not so clear that it's relevant to us.

Tan, Teo, and Anthony (2011): link Investigated a few special cases.

Question: What should we bound? What class of nonlinearities should we allow?

# Chapter 4

# Queue

LeCun 2013.

Snoek, Larochelle, Adams.

Model voting.

Hinton Dropout paper.

Bengio's dropout paper.

ReLU.