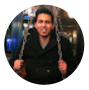
Soumith Chintala -2014-12-13T13:47:48-0500- Updated: 2014-12-13T13:51:34-0500

A few months ago we had a small post here discussing different weight initializations, and I remember +Sander Dieleman and a few others had a good discussion . It is fairly important to do good weight initializations, as the rewards are non-trivial.
For example, AlexNet, which is fairly popular, from Alex's One Weird Trick paper, converges in 90 epochs (using alex's 0.01 stdv initialization).
I retrained it from scratch using the weight initialization from Yann's 98 paper, and it converges to the same error within just 50 epochs, so technically +Alex Krizhevsky could've rewritten the paper with even more stellar results (training Alexnet in 8 hours with 8 GPUs).
In fact, more interestingly, just by doing good weight initialization, I even removed the Local Response Normalization layers in AlexNet with no drop in error.
I've noticed the same trend with several other imagenet-size models, like Overfeat and OxfordNet, they converge in much lesser epochs than what is reported in the paper, just by doing this small change in weight initialization.
If you want the exact formulae, look at the two links below:
https://github.com/torch/nn/blob/master/SpatialConvolution.lua#L28
https://github.com/torch/nn/blob/master/Linear.lua#L18
And read yann's 98 paper Efficient Backprop: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

On that note, Surya Ganguli's talk this year at NIPS workshop wrt optimal weight initializations triggered this post. Check out his papers on that side, great work.

Shared to the community Deep Learning - Public

+1'd by: Chaeun Lee, Ishan Nigam, Kerawit, Aruni RoyChowdhury, Marcin Dahlen, Jean-Marc Beaujour, Ed Chi, Dmitriy Salnikov, Ravi Kiran, Jie Zhang, vigneshwer dhinakaran, Heikki Arponen, Brian McFee, David N. Sanchez, Akash Mehra, Harsh Hemani, Dhanar Adi Dewandaru, Kai Arulkumaran, Jae Hyun Lim, Mat Kelcey, Albert Swart, Vasile Rotaru, Yusuke Watanabe, Victor Escorcia, Andrew Greene, Artem S, Miguel Eduardo Gil Biraud, Michal Hradiš, José Carlos Méndez de la Torre, Anuroop Sriram, Mehdi Mirza, Olivier Grisel, Alexander Matyasko, Viet Nguyen, Liu Liu, mu Takagiwa, Xiao Zhang, Dan Brickley, Federico Pernici, Fabio Kepler, Madison May, Akshay Kumar, Jason Rolfe, Min Lin, John Tran, Junbo Zhang (Lucktroy), Eric Battenberg, Jan Jannink, Saining Xie, Dan Farmer
Reshared by: Chaeun Lee, Balamurugan V R, Haoliang Li, Gaurav K, Zhiming Luo, Oleg Zabluda, Shaoxiang Chen, Sisu Alexandru, Debasish Ghosh, José Carlos Méndez de la Torre, jian chen, Adam Polyak, Emre Safak, Sergey Matyunin, Emad Barsoum, Rupesh Kumar Srivastava, Min Lin, Zhouhan Lin, Junbo Zhang (Lucktroy), Jie Shao, Saining Xie, Yushu Gao, Mikkel Frimer-Rasmussen, Kyle Kastner, Joshua Susskind, Yang Chen, Hailin Jin, Sander Dieleman, Federico Pernici, Soumith Chintala

---

Alex Dalyac - 2014-12-13T13:53:46-0500
it really sucks that previous posts are not easy to find and that there's no topic level organisation, would be much more contributive. should we be hanging out on reddit?

Soumith Chintala - 2014-12-13T13:57:05-0500
https://plus.google.com/117584495470836419167/posts/f3tPKjo7LFa That's the previous post. I found it easily using the search bar, and entering the word "weight".

But I agree, reddit would be way better, I like it much more. But communities cant be moved forcefully, it is what it is :)

Sander Dieleman - 2014-12-13T14:00:13-0500 - Updated: 2014-12-13T14:19:49-0500
I've actually been wanting to do some experiments. I recently retrained some models from my Galaxy Challenge entry, with a more 'standard' initialization than the one I used for the competition (it was very ad hoc and I wasn't too happy about that). Turns out I haven't been able to achieve the same level of performance with these.

So by chance, there is something particular about the initialization I used for my best network, and I still haven't figured out what it is. The differences in performance aren't huge, but they are significant (and for Kaggle competitions, every fraction of a percent matters). One interesting thing I noted was that I always get better results with initial weights drawn from a Gaussian than with weights drawn from a uniform distribution (same variance).

I'll have a look at Surya Ganguli's work. Unfortunately I couldn't make it to NIPS this year :(

EDIT: I realized I should clarify that the 'standard' initialization I was talking about is not the same as the one you link to. I used the version that also takes into account both the fanin and the fanout of each unit, as in Glorot & Bengio 2010, formula (16): http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

I also tried a Gaussian initialization based on this idea, and it seemed to work better than uniform. Still not as good as my 'custom' initialization though. I suspect it might be an interaction with dropout, which I only applied in the fully connected layers of my models.

I should probably compare with using only the fanin (as you did) as well.

Sander Dieleman - 2014-12-13T14:22:56-0500
Also, provided that the biases are initialized to zero, increasing the initialization stddev of a ReLU layer by a factor of sqrt(2) seems to help to get things going in very deep networks, in my experience. This factor compensates for the fact that approx. half of the units will be inactive initially, which reduces the variance by a factor of 2.

Soumith Chintala - 2014-12-13T14:40:15-0500
To tl;dr surya's approach: initialize your weights using auto encoders

Sander Dieleman - 2014-12-13T14:42:13-0500
Actually, after reading the code that +Soumith Chintala linked more carefully, the distribution that the initial values are drawn from doesn't have the specified stddev at all, so it's not actually implementing the initialization from LeCun '98.

In the code, the distribution is Uniform[-stddev, stddev], but to get the desired standard deviation, this should actually be Uniform[-sqrt(3)*stddev, sqrt(3)*stddev].

With that factor of sqrt(3) missing there isn't really a theoretical justification for this initialization. But clearly it seems to work well, from Soumith's experience, so what's going on there?

Soumith Chintala - 2014-12-13T18:22:34-0500
+Sander Dieleman section 4.6 equation 15. Exactly the formula that I am using.

Sander Dieleman - 2014-12-13T18:34:22-0500
Yes, but a uniform distribution on the interval [-a, a] has a standard deviation of a / sqrt(3), not a.

So to get a uniform distribution with an stddev of a, you actually need to specify the interval [-sqrt(3)*a, sqrt(3)*a], and as far as I can see this doesn't happen in the code you linked.

Soumith Chintala - 2014-12-13T19:46:14-0500
Ah yes its a good bug. Apparently it was a bug to not multiply by sqrt(3) but it helped with some gradient scaling issues across layers, so it was left in there as it works slightly better in practice

Sander Dieleman - 2014-12-13T20:13:20-0500
See, that's very interesting :) I want to know why that is!

Xiang Zhang - 2014-12-14T01:12:34-0500
I though that initialization was intended for the case when you use sigmoid or tanh, for which the network will be stationed at a position where gradients are least saturated/vanished at back-propagation. I guess not using sqrt(3) could work because rather than using sigmoid or tanh nowadays, we are using Threshold (reglu).

Alec Radford - 2014-12-14T01:23:07-0500 - Updated: 2014-12-14T01:24:17-0500
Has anyone tried or seen benefits from the random orthogonal initialization technique? Curious how it applies to conv layers.
Paper: http://arxiv.org/abs/1312.6120
Presentation: https://www.youtube.com/watch?v=Ap7atx-Ki3Q&list=PL4VIKuMdJoBETL2PAx14pvmVDLjGuIC1_&index=5

Sander Dieleman - 2014-12-14T04:52:58-0500 - Updated: 2014-12-14T04:56:13-0500
+Xiang Zhang: actually, afaik the reasoning behind using a distribution with an std of 1/sqrt(fanin) uses the assumption that the layer is linear (or that the nonlinearity will preserve variance if it is close to 1). By using this distribution, the variance of the layer output will be about the same as the variance of the input.

If the variance of the input is around 1, and you use f(x) = 1.7159*tanh(2/3 * x), as suggested in LeCun '98, then the variance of the output after the nonlinearity will also be around 1.

If you use tanh(x), sigmoid(x) or max(x, 0), this obviously no longer holds, but people use it anyway because it just works. The reason why is probably that the missing 'correction' factor to bring the variance back to 1 is usually fairly close to 1 anyway, so if your network is not too deep, this doesn't cause huge problems.

For example, tanh(x) with x ~ N(0, 1) has a variance of about 0.63. Note that this actually depends on the shape of the distribution now: tanh(x) with x ~ U[-sqrt(3), sqrt(3)] has a variance of about 0.67.

With today's 20+-layer networks though (ImageNet 2014), I think finding good initializations is going to gain importance. All these missing scale factors would compound into a pretty big scale factor at the top of the network.

I don't think dividing by sqrt(3) helps because of ReLUs. In fact, in my experience what you want to do instead is multiply the variance by sqrt(2) to compensate for the fact that approximately half the units will be off at the start of training (provided that the biases are initialized to zero, which is not always done).

There must be some other reason why making the target output variance smaller than 1 somehow works a bit better, and the ideal scale factor probably isn't sqrt(3). I really want to know what's going on here, "it just works a bit better" is rather unsatisfying :)

+Alec Radford: awesome, thanks for the reference! I'll check this out and probably give it a try as well.

Andrew Saxe - 2014-12-14T13:19:55-0500
Here's our initialization suggestion from http://arxiv.org/abs/1312.6120 as a matlab function:

```
function W = ortho_init(sz,g)
% sz: list of layer sizes (any sizes allowed)
% g: scale factor to apply to each layer
%
% g should be set based on the activation function:
% linear activations     g = 1 (or greater)
% tanh activations       g > 1
% ReLU activations       g = sqrt(2) (or greater)
%
% Author: Andrew Saxe, 2014

Nl = length(sz);
for i = 1:Nl
   [R{i},tmp] = qr(randn(sz(i),sz(i)));
end

for i = 1:Nl-1
   dim = min(size(R{i+1},2),size(R{i},1));
   W{i} = g*R{i+1}(:,1:dim)*R{i}(:,1:dim)';
end
```

It's basically an initialization to orthogonal matrices, scaled by an overall gain g. This g should be set for different activation functions:

Activation	g
Linear	1
Tanh	greater than 1
ReLU	sqrt(2)

(our paper shows how to do this for other activation functions)

A couple papers have reported some luck with this scheme, eg, http://arxiv.org/pdf/1406.1078v3.pdf and http://papers.nips.cc/paper/5350-learning-to-discover-efficient-mathematical-identities.pdf.

The scaling factor seems to be more important than the orthogonal initialization in many cases, so you may be able to get away with just scaling the Glorot initialization suggestion by the above values.

This initialization uses orthogonal matrices, but there's a bit of subtlety when it comes to undercomplete layers—basically you need to make sure that the paths from the input layer to output layer, through the bottleneck, are preserved. This is accomplished by reusing parts of the same orthogonal matrices across different layers of the network.

In general, scaling up the weights can cause training of even very deep networks to be fast. See for example the plot here http://youtu.be/Ap7atx-Ki3Q?t=15m39s. That shows train and test error after a fixed number of epochs for a 30 layer tanh network as a function of the scaling factor g. Two things to note: first, larger gains make the optimization go super fast (low train error); second, larger gains at some point start hurting generalization error. So that's the downside… and I think a really interesting candidate for more analysis!

Also keep in mind our paper only analyzes the training error dynamics. From an optimization perspective, we should strive to optimize training error as fast as possible, adding regularization to achieve good generalization—in present practice, we mix these two things. Our models are implicitly regularized because of the way we optimize them by starting with small weights.

So to summarize what I think is a more accurate set of intuitions for why deep learning is hard,

1) Small random initializations are slow because they start you near a saddle point, in particular, the saddle point in which all weight matrices are zero.
2) The time spent near this saddle point is proportional to 1/epsilon where epsilon is the "initial mode strength" defined precisely in the paper, but to get the general idea, it's a product of the strengths in each layer: epsilon = a_1 * a_2 * … a_n. To get the intuition, just think of a deep linear network with only one neuron per layer—then the initial mode strength a_i just is the initial weight value for that layer, and epsilon is the product of them all. With small random weights you're multiplying lots of little numbers and so the overall epsilon is going to be exponentially small in depth. Learning is slow since 1/epsilon is huge. If the a_i are > 1, then the product is going to explode—also a problem. A good initialization scheme makes these a_i about equal to one, because then epsilon is about equal to one regardless of depth and learning proceeds quickly.
3) In fact, if epsilon is about 1, then very deep networks are only a *constant factor* slower than shallow networks.
4) Everything I just described is from deep *linear* networks. In a nonlinear network, you can get a good initialization by autoencoder pretraining (still improves training speed even now in the era of second order methods—our paper has a tiny lit review in the appendix). This establishes epsilon = 1. Or, you can compensate for your typically compressive nonlinearity by using slightly larger than orthogonal matrices. That's the gain factor proposed above, and for different nonlinearities you can compute the gain factor that will cause activity to propagate indefinitely, rather than decay away in a deep network.

Sander Dieleman - 2014-12-14T15:34:24-0500 - Updated: 2014-12-14T15:57:25-0500
Very cool, thanks a lot for chiming in! Looks like I wasn't crazy scaling up my initializations by sqrt(2) for ReLU layers :)

This part of your comment is intriguing, and I think I don't fully understand it: "basically you need to make sure that the paths from the input layer to output layer, through the bottleneck, are preserved. This is accomplished by reusing parts of the same orthogonal matrices across different layers of the network."

Could you elaborate on this a bit? What does this look like in practice? Many practical neural nets have undercomplete layers ("bottleneck layers" were a huge buzzword at ICASSP this year), so I'm wondering what's so special about them and how your "reuse parts of the same orthogonal matrices"-trick works.

EDIT: also, I'm curious if any special care needs to be taken with convolutional layers. I guess those will typically be 'undercomplete' in practice. For example, if a convolutional layer has 32 3x3 filters, and the input has 32 feature maps, then the number of filters is 9 times smaller than the fanin of a filter.

Eric Battenberg - 2014-12-14T18:20:34-0500
I'm really glad this is being discussed again. This is exactly the kind of stuff I was hoping to see when I asked about it a couple months ago. If only someone would've pointed us to the ICLR paper then. :) I was very interested in all the work Surya presented in his talk. We definitely need more of these theory-based recommendations for training deep networks.

Alec Radford - 2014-12-14T20:23:45-0500

+Sander Dieleman It works for convnets :) Just had a VGG-D network learn using orthogonal initalization.

To my knowledge the paper claimed they couldn't train beyond 11 weight layers with standard random initalization and this was 16 weight layers.

Sander Dieleman - 2014-12-15T04:44:57-0500
+Alec Radford cool! Did you use a variant of the MATLAB code that +Andrew Saxe posted, or something else? From his comment it seems that not all orthogonal initializations are created equal :)

The paper says "We note that it might be possible to appropriately initialise learning without pre-training, e.g. by adjusting the random initialisation procedure [19]."

So guess that's a yes then!

Andrew Saxe - 2014-12-15T07:00:32-0500 - Updated: 2014-12-16T12:04:33-0500
Yeah the over/undercomplete issue is really a minor point--I can't imagine it makes a big difference in practice--but here it is:

Suppose you have two weight matrices, W_1 and W_2, and W_1 goes to a tiny layer of neurons, e.g., it has very few rows. In the linear case, the overall function the network computes is just y = W_2W_1x, which you can rewrite as y = Wx for some W. But note that you can't represent *any* possible W with this network. The small W_1 layer imposes a rank constraint on the overall function. If you have 10 neurons in the W_1 layer, you can't represent more than a rank 10 linear function. If you took the SVD of W, i.e. W=USV^T, there could only be 10 nonzero elements in S.

Now about initializations: take a net with three weight matrices, W_1 through W_3, with two bottleneck layers, one between W_1 and W_2 (so W_1 has few rows), and one at the output of W_3 (so W_3 has few rows). To be concrete let's say the undercomplete layers are size 10 and the others are size 100. We can break down each weight matrix using the SVD to get W_i = U_i S_i V_i^T. Then the overall map is y = U_3 S_3 V_3^T U_2 S_2 V_2^T U_1 S_1 V_1^Tx. At the undercomplete layers, the singular values S_1 and S_3 have the form [D_i 0] where D_i is a 10x10 matrix with the singular values on the diagonal, and the 0 there is a 10x90 matrix of zeros to fill out the rest. This means that inputs aligned with the first 10 input singular vectors of V_1 or V_3 will get passed through, but everything else won't make it. At the overcomplete layer, S_2 has the form [D_2; 0] where the 0 is 90x10 now. All 10 input directions make it to the output, but there are 90 unused output directions in the overcomplete layer that aren't hooked up to any input.

To be a good initialization, you want the overall function to have singular values of about one. You might think that you could just set all these D_i matrices equal to the identity to get this. This is sort of like an "orthogonal" initialization but our matrices aren't square and so can't be orthogonal in the standard sense that U^TU=I=UU^T. Now here's the crucial bit: suppose you are quite unlucky and the 10 output singular vectors U_2 with nonzero singular values are aligned with 10 of the 90 input singular vectors of V_3 with zero singular values. Then W_3W_2 = 0. And so the overall function y = W_3W_2W_1x=0, regardless of the input x. The first bottleneck has funneled all of the information into directions that the second bottleneck wipes out. Or said another way, after the first bottleneck all of the activity lies in one 10 dimensional subspace of the 100 dimensional overcomplete layer, but the second bottleneck is only listening to information in some *other* 10 dimensional subspace.

The fix is to tie together the input singular vectors and output singular vectors across layers to make sure this doesn't happen. I.e. recall that y = U_3 S_3 V_3^T U_2 S_2 V_2^T U_1 S_1 V_1^Tx. Choose the initialization such that V_3=U_2 and V_2=U_1. Then the overall function is y = U_3

S_bottleneck V_1^Tx where S_bottleneck has 10 nonzero elements (or whatever the dimension of the smallest bottleneck is) all equal to one, as desired. The code I posted does that (you can see each matrix R gets used twice, once as an output singular vector matrix U and once as an input singular vector matrix V^T).

+Alec Radford that's quite encouraging to hear it helps with convolutional nets! I haven't been sure whether this analysis would apply. As +Sander Dieleman noted, the solution we found isn't clearly applicable to convolutional nets (a side note about that, it seems conv layers can be overcomplete in practice too, typically the very first layer. For example the first layer of Krizhevsky's net goes from a 150k-dimensional input to a 250k-dimensional output). Alec's approach of using orthogonal filters would seem to give independence between filters, which apparently is helpful, but you might also be able to reduce coupling between a single filter and itself at nearby locations. It'd be great to see some solutions of the gradient descent dynamics specialized for linear conv nets if that's possible. I should try that :).

Sander Dieleman - 2014-12-15T09:41:59-0500
Thanks for the very clear and detailed explanation. It would be interesting to see how much it actually matters to do this, as a function of depth.

As you mentioned convolutional layers are sometimes overcomplete, but in most cases they seem to be undercomplete, so if you stack 20 of them I guess the probability of running into this problem becomes nonnegligible. I'd love to see some quantitative results :)

Alec Radford - 2014-12-15T15:12:57-0500 - Updated: 2014-12-15T15:15:47-0500
+Sander Dieleman +Andrew Saxe Yep, I used Andrew's code (just ported to numpy). Thanks for sharing! I agree the independence of filters is the most likely reason for helping. I also wonder if using orthogonal on the output layer might be helping, need to do more comparisons.

Side interesting note, very deep nets can learn using tanh and small update sizes / nesterov even with standard initalizations. 30 layer mnist learns when you just use uniform(-0.05, 0.05) but is slow compared to orthogonal. ReLU with uniform/normal don't ever seem to learn in this setup.

Seems to line up with observations that ReLU RNNs are difficult to train - begs the question that maybe they'll work now if orthogonal initialization is used.

Alex Rothberg - 2014-12-16T10:35:50-0500
+Alec Radford Would you be able to share this numpy code?

Kyle Kastner - 2014-12-30T03:21:25-0500
+Alec Radford I think the problems of unbounded ReLU in the sequence part of RNNs is more due to the unlimited dynamic range above 0. Clipped ReLU can work (dynamic range 0-20 perhaps, I am experimenting with this but had some luck previously), but to me that still feels like a hack. Tanh (or better yet, LSTM/GRU) has worked fine for me, but the orthogonal initialization was very useful here as well, though I used a different though perhaps equivalent formulation.

Between layers, there is no problem - just like a feedforward net. ReLU, dropout, etc. is all helpful from http://arxiv.org/abs/1409.2329 .

Dunno what to comment on the deep deep MNIST thing, but my ongoing RNN work is here: https://github.com/kastnerkyle/net/blob/master/net.py

Eric Battenberg - 2014-12-30T03:43:36-0500
+Kyle Kastner, what's the reasoning behind specifically choosing 20 as the upper limit in a clipped ReLU?   I did see that range in the Baidu "Deep Speech" paper. Have others been using that value as well?

Is this another case of someone originally chose an arbitrary number that seemed to work, and others stuck with it? I'm thinking of the 0.01 stdev gaussian weight initialization from the AlexNet paper.

Kyle Kastner - 2014-12-30T03:51:44-0500 - Updated: 2014-12-30T03:52:03-0500
+Eric Battenberg I chose it from Baidu :) so definitely arbitrary. Don't know about any others using it. Hoping to try different values or find a nice heuristic to set it at some point.

Soumith Chintala - 2014-12-30T12:25:31-0500
I've played with Saturated linear units (ReLU with an upper bound as well).
It depended on problem to problem, but for mnist-style problems i got away with upper bounds of 2.0, but for larger networks, i had to up the bound to about 8.0. This got me within 1% error difference vs ReLU nets

Eric Battenberg - 2014-12-30T13:09:14-0500
+Kyle Kastner, I just remembered I used 20 for the upper limit of these bounded softplus units I used in some old work on CRBMs.

(cf. Fig 4)
http://ericbattenberg.com/school/drum_patterns_ismir2012.pdf

I chose 20 to get a specific noise level on the reconstructions, but I'm thinking 20 is some sort of universal magic number. Might as well stick with it. ;)

Sergey Ten - 2015-01-04T03:21:06-0500
Is 20 related to -128..127 range of input images?

Alex Rothberg - 2015-01-05T23:30:41-0500
Interesting paper on the subject of weight initilization: http://arxiv.org/pdf/1412.6558.pdf

Urko Sanchez - 2015-02-14T11:02:13-0500
I guess it is worthy to update this discussion with this paper http://arxiv.org/pdf/1502.01852.pdf since many people end up here when looking for initialization tricks.

Yannis Assael - 2015-08-04T10:59:19-0400 - Updated: 2015-08-04T11:02:48-0400
+Soumith Chintala in case anybody is interested I implemented an nn.Linear module with Orthogonal Weight Initialisation for torch.

https://github.com/iassael/torch-linear-orthogonal

thank you for your constant support!

Oleg Zabluda - 2015-10-17T21:50:12-0400 - Updated: 2015-10-17T22:18:25-0400
+Soumith Chintala> more interestingly, just by doing good weight initialization, I even removed the Local Response Normalization layers in AlexNet with no drop in error.

Do you understand why AlexNet even used LRN at all? Overfeat "fast model" didn't, and it's a slight variation on 1-col Alexnet (with more featuremaps) and "Krizhevsky initialization". In fact, it's very rare to see LRN elsewhere. In the paper they do say that it helps, but why it helps AlexNet, but not OverFeat "fast".

Soumith Chintala - 2015-10-18T00:22:05-0400
+Oleg Zabluda The broad idea of the local contrast or response normalization is to make sure that the input to the next layer is nicely centered and scale invariant. With proper weight initialization, you make sure that the inputs to the next layer will naturally have these properties (at least at the

beginning of training) and hence these layers are less important. Batch Normalization (a landmark paper from this year http://arxiv.org/abs/1502.03167) also tackles the same problem in a nice way.

Oleg Zabluda - 2015-10-18T00:32:43-0400 - Updated: 2015-10-18T00:44:18-0400
But why LRN helps AlexNet, but not nearly identical (to 1-col) OverFeat "fast" (table 1 of http://arxiv.org/pdf/1312.6229v4.pdf), which didn't use BN either, and did use identical "Krizhevsky initialization" N(0, .01)? Maybe that initialization accidentally was a better fit for the larger number of featuremaps?

Urko Sanchez - 2015-10-18T09:28:41-0400
+Soumith Chintala +Oleg Zabluda I may add also the transformer modules (http://arxiv.org/abs/1506.02025) to the set of things that tackle the problem. For invariances in a bit more general case Domingo's Deep Symmetry Networks may be interesting too (http://homes.cs.washington.edu/~pedrod/papers/nips14.pdf). As for LRN, I've never seen improvement using them in my experiments, Im also curious about why they did work in Krizhevsky's net.

Oleg Zabluda - 2015-10-18T19:22:20-0400
What may help in figuring it out would be to pinpoint where LRN stopped improving things. Was it during the transition AlexNet(2-col)=>AlexNet(1-col), without Alex realizing it? Or was it during the transition AlexNet(1-col) => Overfeat(fast)? AlexNet(2-col). AlexNet(1-col), Overfeat(fast) are otherwise nearly the same: topology, ReLU, 0.5 Dropout, N(0, .01) initialization, no BN.

Oleg Zabluda - 2015-10-18T20:02:41-0400
+Soumith Chintala>The broad idea of the local contrast or response normalization is to make sure that the input to the next layer is nicely centered and scale invariant. With proper weight initialization, you make sure that the inputs to the next layer will naturally have these properties (at least at the beginning of training) and hence these layers are less important.

Krizhevsky et al are quite explicit in the paper that LRN does not help learning, but
""""
However, we still find that the following local normalization scheme aids generalization.
""""

and they tend to specify those things explicitly, for example:
""""
We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model’s training error.
""""

Dmytro Mishkin - 2016-03-03T09:15:14-0500
Hi,

Instead of deriving gain coefficients to ensure mean 0, var 1, of layer output, one could use data. Init with Saxe -> forward pass, minus mean / divide by std.

It is very simple and works quite well, as independently found by me http://arxiv.org/abs/1511.06422 and Berkeley guys http://arxiv.org/abs/1511.06856.

P.S.Sorry for necro-posting and self-PR :)

Long Ouyang - 2016-03-10T18:15:04-0500

Hi +Soumith Chintala,

I'm a little late to this party, but I'm interested in the Ganguli talk you mentioned.. I can't quite figure out which work of his you're referring to. Do you know if there is a paper / video of this online somewhere?

Soumith Chintala - 2016-03-10T18:31:34-0500
+Long Ouyang couldn't find a video of the talk, but here's the paper: http://arxiv.org/abs/1312.6120
And here's an earlier talk from Saxe on the same paper: https://www.youtube.com/watch?v=Ap7atx-Ki3Q