23 Residual Networks; Batch Normalization; AdamW

TRAINING DEEP NETWORKS

Most influential ideas: ResNets, batch normalization, layer normalization.
[These ideas enable deep networks to train. They reduce the likelihood of encountering the vanishing gradient or exploding gradient problems, but don't quite eliminate them.]

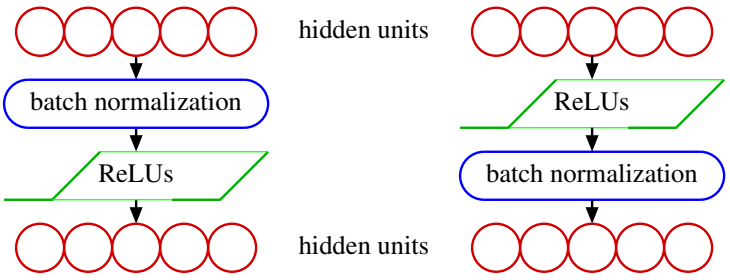
Batch Normalization

[Batch normalization has played a huge role in making it easier to train very deep neural networks since its introduction in 2015, and it's still a mainstay today. It seems to make the cost function uglier, though; when you can train a network without it, it might generalize better.]

Recall batch normalization from Homework 6: For a vector a of activations,

- a batch-norm layer learns parameters β_i and γ_i for each activation a_i;
- calculate the sample mean μ of a and the sample variances σ_i^2 of a_i over a minibatch;
- for some small ϵ, the layer outputs vector z with $z_i = \beta_i + \gamma_i \dfrac{a_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$.

[We didn't say much in the homework about where batch normalization layers are used. There is some disagreement over whether it's best to place them before or after a nonlinear activation function. Both ways are commonly used. The only clear rules are to never use batch normalization for outputs, and never use ReLUs for inputs.]



batchrelu.pdf [The original authors proposed the left version.]

[Batch normalization is often applied to image data in convolutional neural networks, but not quite like this. We do not normalize each pixel separately. Remember that in a CNN, we rely heavily on relationships between adjacent pixels. Normalizing each pixel separately could introduce spurious edges and ruin the network's ability to recognize images. But we can normalize an entire image as a whole, and we can normalize each channel separately.]

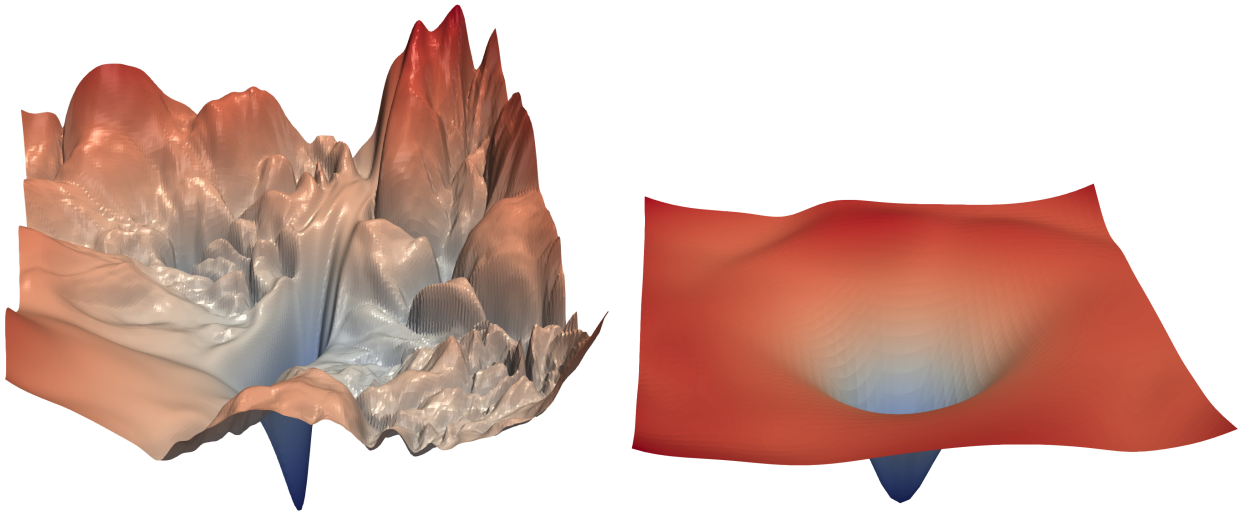
Batch-norm for images: compute mean & variance over all images in minibatch AND all pixels in each image. One β and one γ **per channel**.

Layer normalization: compute mean & variance over all hidden units (all pixels and all channels), but not (necessarily) over images. One β and one γ **per image**.

[Layer normalization ensures that for any one image and any one layer of hidden units, the hidden unit vector will lie on a sphere with center β and radius γ. Layer normalization is easier to parallelize than batch normalization, and it is particularly useful in recurrent neural networks.]

Residual Neural Networks (ResNets)

[Look at this famous figure depicting two-dimensional cross sections through the cost function of deep convolutional neural networks. At right is the cost for a residual neural network. At left is the cost if we remove the "residual connections" that characterize residual networks. You can guess which one is easier to optimize.]



(a) without skip connections (b) with skip connections

lossskip.pdf, (Hao Li et al., 2018)

Idea: design a network with layers that can easily represent the identity fn, e.g., when all weights are zero.
[There are two observations that help to motivate this idea.]

Motivation 1: Networks with nonlinear activations have great difficulty representing the identity. Let's fix that.
[If a hidden layer has negative unit values, a subsequent hidden layer with ReLUs cannot replicate those values. You might be able to replicate them at a linear output layer if you're very clever. Instead, let's redesign our networks to make it easy.]

Motivation 2: Consider a linear neural network $\hat{y} = W_L W_{L-1} \cdots W_1 x$ with square matrices W_i.

Given an $n \times d$ design matrix X and an $n \times d$ label matrix Y, solve this linear regression problem by gradient descent [batch or stochastic].

Find matrices that minimize $J(W_L, W_{L-1}, \ldots, W_1) = \|W_L W_{L-1} \cdots W_1 X^\top - Y^\top\|_F^2$.

The cost fn is very smooth around $(I, I, \ldots, I)$, but much more complicated in regions close to $(0, 0, \ldots, 0)$.
[There's a paper showing that near the identity matrices, every critical point of the cost function is a global minimum—much like in the figure at top right. So even a simple linear neural network suffices to show some of the behavior in the figure.]

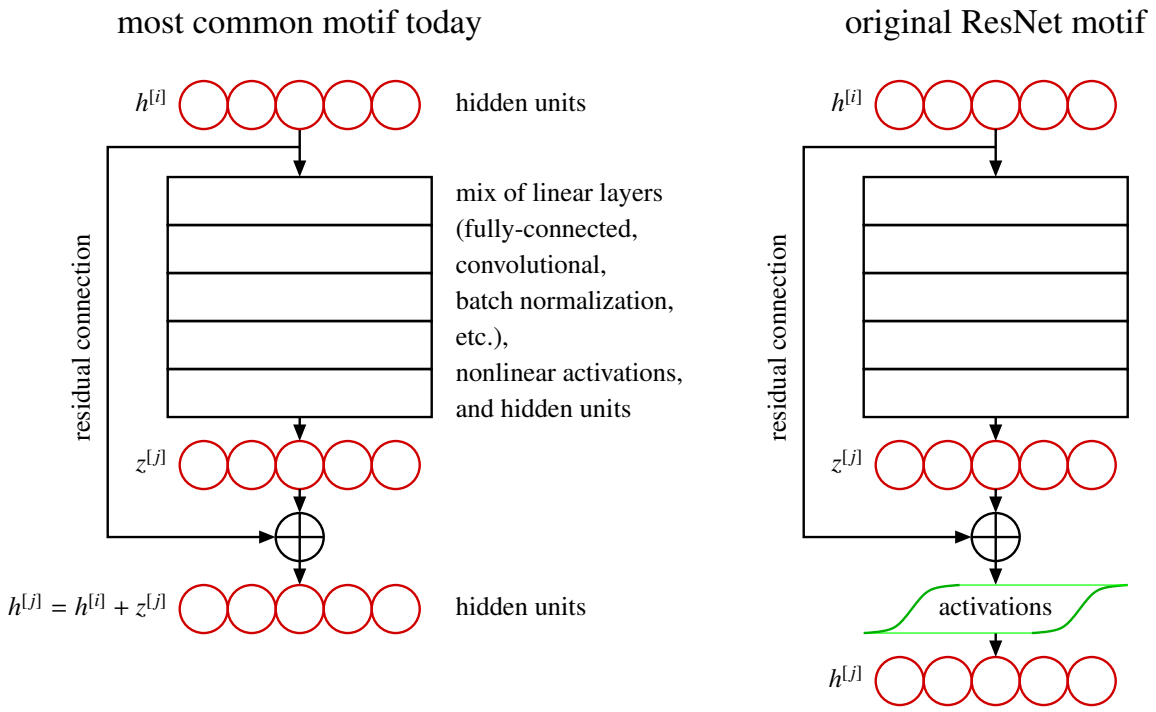
Any network satisfying $W_L W_{L-1} \cdots W_1 = Y^\top (X^\top)^+$ is a solution.
If L is sufficiently large, there is a solution or approximate solution where each W_i is "close" to I.
[If you multiply together enough matrices that are close to the identity matrix, you can obtain any square, invertible matrix, and you can get very close to any square matrix. So for sufficiently many layers, there are solutions in the "nice" region of the cost function. SGD starting from the identity network will find them.]

[Of course, you would never do linear regression this way. But when we add nonlinear activation functions such as ReLUs between the matrices, the phenomenon persists that the cost function is ugly near the origin and can be relatively nice where the network is computing an function near the identity function at each layer.]

Takeaways:

- Initializing near I (plus small random weights) is better than initializing near zero.
- More layers makes it more likely there's a solution in a "nice" part of the cost fn.

ResNets use residual connections aka skip connections to add hidden layer values to subsequent layers. Networks are constructed by repeating one of these motifs.



motifs.pdf

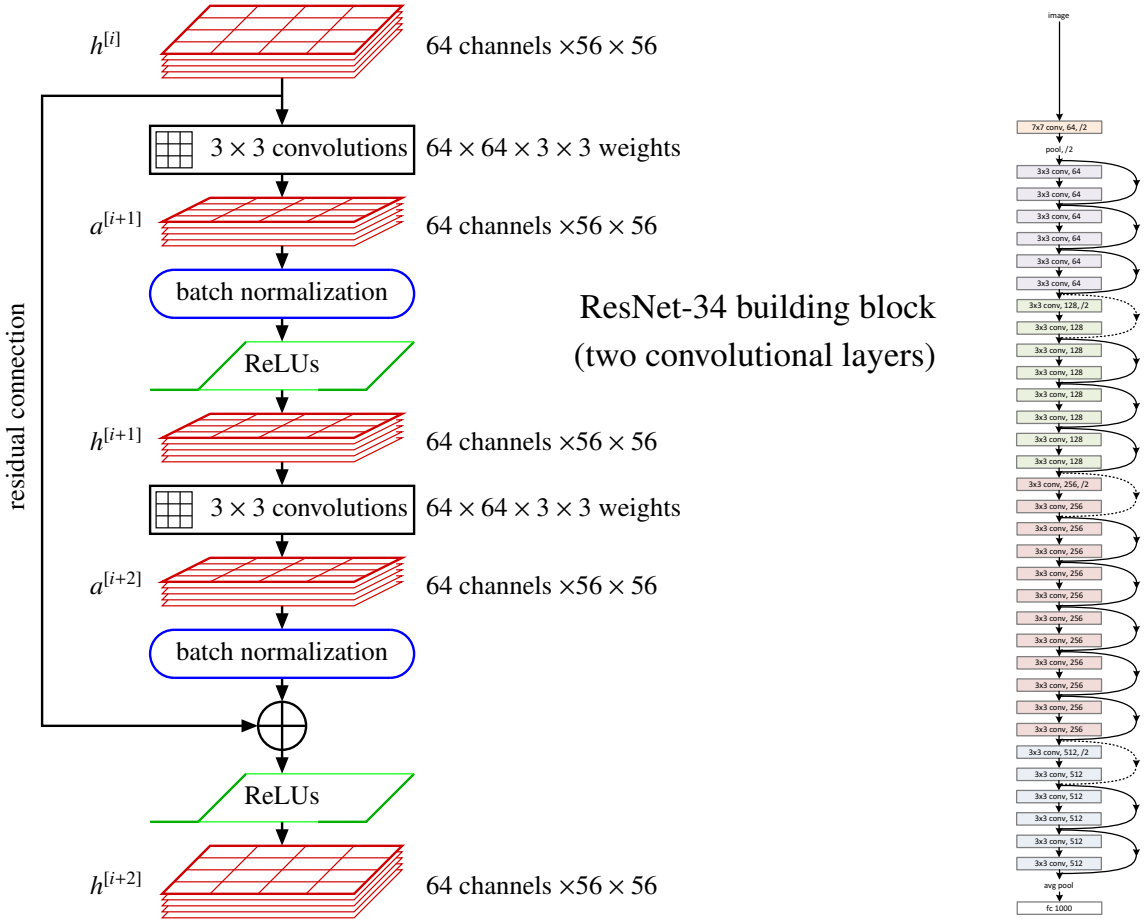
[The motif on the right was used by the original ResNet paper, with ReLU activations after the residual connection. But the motif on the left seems to be more popular now.]

If all weights are zero, left motif sets $h^{[j]} = h^{[i]}$ by default.
Right motif with ReLUs copies all positive units but zeros out negative ones.

[A big advantage of the motif on the left is that if it is advantageous to send some hidden unit values from an early layer to a later layer unchanged, the network has the opportunity to learn to do that. The motif on the right can do that with positive hidden unit values if it uses ReLU activations.]

If the "ideal" mapping from $h^{[i]}$ to $h^{[j]}$ is expressed by a function f, the left motif is trying to learn $f(h) - h$ (which we hope is small).

[The first ResNets were CNNs for image classification. The authors won first place in the 2015 ImageNet Large Scale Visual Recognition Challenge with an ensemble of six ResNets, two of which had 152 layers. This was the biggest advance in neural network vision performance since the AlexNet paper that changed modern computer vision in 2012. Here is the building block for one of their smaller ResNets, a 34-layer model. At right is the authors' schematic of the whole ResNet-34 network with normalizations omitted.]

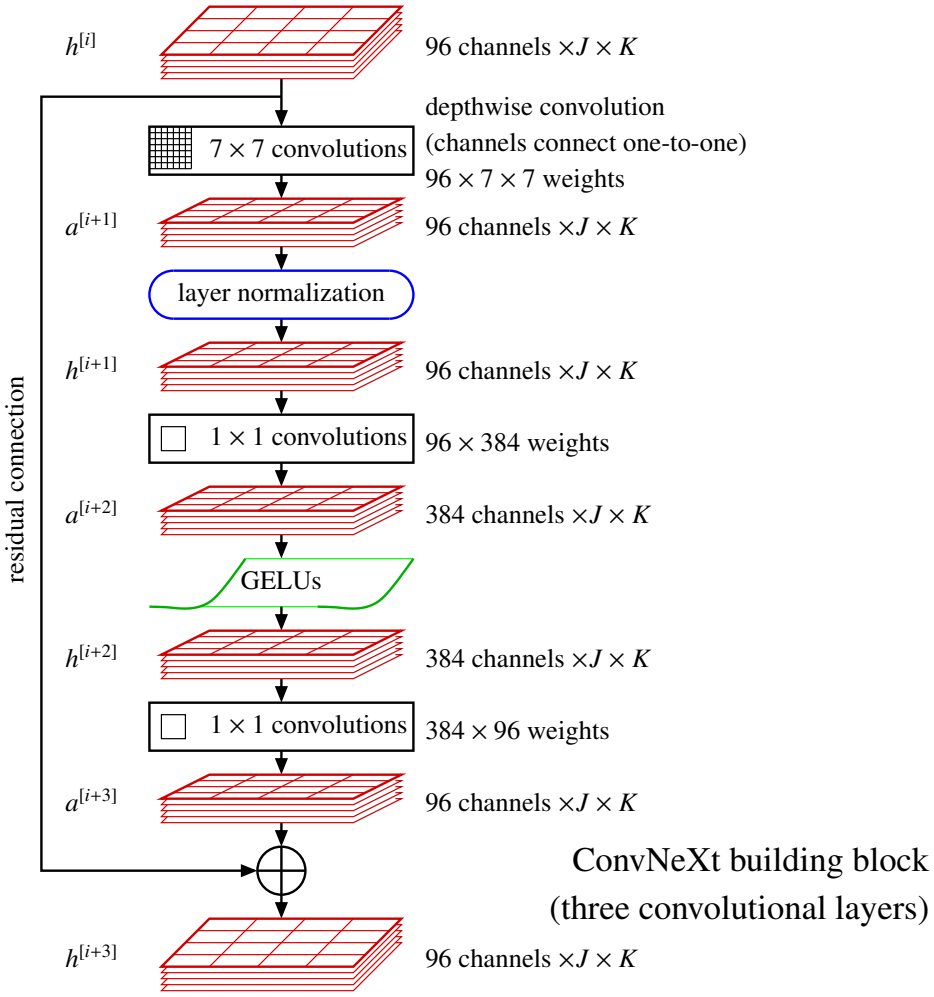


resnet34.pdf, resnet34he.pdf

[Observe that there is a layer of ReLU activations in the middle of the motif, with linear convolutions before and after it. Each convolution is followed by a batch normalization layer. The biggest ResNet in the competition-winning ensemble, with 152 layers, uses a motif with three convolution layers, three batch normalization layers, and three ReLU layers—two within the residual connection and one after.]

[Below is an example of the building block for a more modern convolutional ResNet, called ConvNeXt, produced by a collaboration between Facebook and Berkeley. Unlike the original ResNet, it does not place an activation function after the residual connection.]

[The authors found that one layer normalization step per three convolutional layers suffices, whereas the original ResNets used a batch normalization step after every convolutional layer. Instead of a ReLU, they use a GELU, which stands for Gaussian Error Linear Unit. It's similar in shape to a ReLU but it's smooth, with no discontinuity. It appears to give better test accuracy than ReLUs in some circumstances, though not all. It is popular in transformers for speech generation.]



[The authors found that they get better test accuracy if they use 7×7 convolution filters, not just smaller ones. But these filters use depthwise convolution, which means that each output channel receives input from only one output channel. By contrast, traditional convolutional layers connect every output channel to every input channel. The advantage of depthwise convolution is a big savings in weights and time.]

[Another interesting design choice is the use of an inverted bottleneck, where they temporarily increase the number of channels from 96 to 384 and then decrease it again to 96. This creates very wide layers that lead to better generalization to test data. To prevent the number of weights from exploding, they use the odd idea of a 1×1 convolution going into and out of the 384 channels. This simply means that there is an all-to-all connection from 96 channels to 384 channels, but there are no connections between adjacent "pixels" in the activation maps. Note that the 1×1 convolutions are not depthwise convolutions! A 1×1 depthwise convolution would not permit any mixing of information at all.]

[The combination of depthwise convolutions and 1×1 convolutions causes channel mixing to be separated from spatial mixing. Both kinds of mixing take place, but they take place in different convolutional layers. This permits us to greatly reduce the number of weights while still allowing information to eventually flow everywhere.]

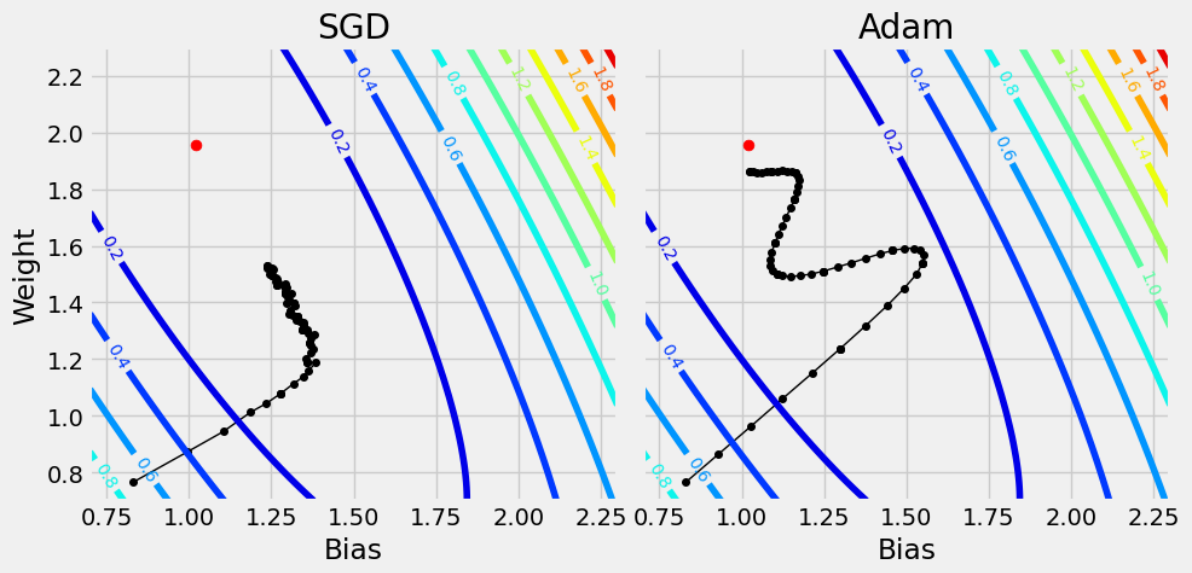
AdamW

"Adaptive moment estimation with weight decay."
An optimization method faster than SGD. Warning: may reduce test accuracy.
Let J be losses summed over a minibatch. Let w_i be a weight. Intuition:

- Sign of $\dfrac{\partial J}{\partial w_i}$ gives more useful information than its relative magnitude.
- We change w_i slowly if $\dfrac{\partial J}{\partial w_i}$ changes sign frequently; otherwise stay fast.
  [Roughly speaking, each weight has its own learning rate.]
- Keep exponential moving averages m_i of $\dfrac{\partial J}{\partial w_i}$ [first moment] and r_i of $\left(\dfrac{\partial J}{\partial w_i}\right)^2$ [second raw moment].
  Each step has $\Delta w_i \propto -m_i/\sqrt{r_i}$. Typically $m_i/\sqrt{r_i} \approx \pm 1$, but smaller if sign $(\partial J/\partial w_i)$ changes often.

$m \leftarrow 0$; $r \leftarrow 0$; $t = 0$
repeat
  $t \leftarrow t + 1$
  $g \leftarrow \nabla J(w)$
  $m \leftarrow \beta\, m + (1 - \beta)\, g$
  $r \leftarrow \tau\, r + (1 - \tau) g \odot g$ [elementwise multiplication; square each component of g]
  $\hat{m} \leftarrow m/(1 - \beta^t)$ [correcting bias, as m was initialized to zero]
  $\hat{r} \leftarrow r/(1 - \tau^t)$ [correcting bias]
  $\forall i, w_i \leftarrow w_i - \epsilon\left(\dfrac{\alpha \hat{m}_i}{\sqrt{\hat{r}_i} + \delta} + \lambda w_i\right)$

Typical parameters: $\alpha = 0.001, \beta = 0.9, \tau = 0.9999, \delta = 10^{-8}$.
Weight decay term λ regularizes; set by validation. If $\lambda = 0$, it's just called Adam.



sgdadamgodoy.png (Daniel Godoy) [Left: 50 steps of SGD (with 16-point minibatches) don't get very close to the minimum (red). Right: 50 steps with Adam.]