Building a working autoencoder – I Network tricks - B

D.A. Forsyth

University of Illinois at Urbana Champaign

Gradient problems

Imagine training many layers stacked on one another. Look at Section 17.3.4, and notice that the gradient of the first layer's parameters depends on a stack of derivative matrices connecting the output of the last layer to the result of the first. But the derivative matrices may not be particularly helpful, because the parameters for each layer are wrong (which is why we are training). In turn, the gradient of the first layer's parameters may not be helpful. Now think about the last layer. There are no intervening layers, so problems with derivative matrices don't apply. But the derivative is evaluated with that layer applied to a particular set of input values, and depends on these values. These values are wrong, because the previous layers are wrong, so the gradient update at the earliest layer is going to be poor as well. The argument applies to early and late layers, rather than just first and last. If there are few layers, this effect seems not to prevent training. But a deep stack of layers may be very hard to train. Gradient steps may diverge, or require an impractically small learning rate – and so a very large number of steps – to converge.

Residual connections

Residual connections

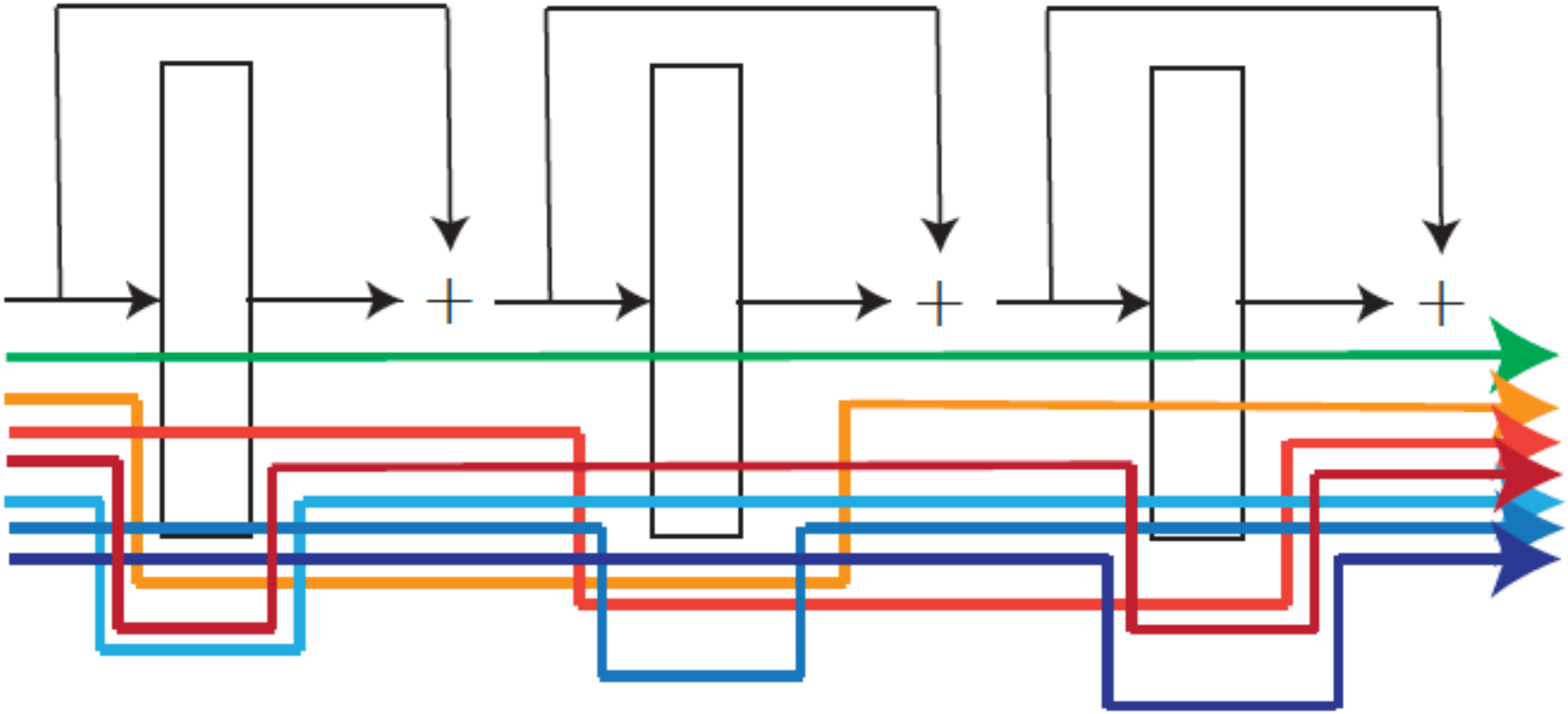
Start with a simple example. Assume that each layer is a convolutional layer (no ReLUs), and that the input feature block is the same size as the output feature block. Recall Section 17.3.4 wrote the w'th layer as $\mathcal{L}_w$. Now write $\mathcal{B}$ for some data block, and $\mathcal{R}_w = \mathcal{L}_w + \mathcal{I}$ (where $\mathcal{I}$ is the identity, so $\mathcal{R}_w(\mathcal{B}) = \mathcal{L}_w(\mathcal{B}) + \mathcal{B}$). Now stack these to get

$$\begin{aligned}
\mathcal{B}_4 &= \mathcal{R}_3(\mathcal{B}_3;\theta_3) \\
\mathcal{B}_3 &= \mathcal{R}_2(\mathcal{B}_2;\theta_2) \\
\mathcal{B}_2 &= \mathcal{R}_1(\mathcal{B}_1;\theta_1) \\
\mathcal{B}_1 &= \mathcal{I}.
\end{aligned}$$

Expand all this to get

$$\begin{aligned}
\mathcal{B}_4 &= \mathcal{L}_3(\mathcal{L}_2(\mathcal{L}_1(\mathcal{B}_1;\theta_1);\theta_2);\theta_3) + \\
&\quad \mathcal{L}_3(\mathcal{L}_2(\mathcal{B}_1;\theta_2);\theta_3) + \mathcal{L}_3(\mathcal{L}_1(\mathcal{B}_1;\theta_1);\theta_3) + \mathcal{L}_2(\mathcal{L}_1(\mathcal{B}_1;\theta_1);\theta_2) + \\
&\quad \mathcal{L}_3(\mathcal{B}_1;\theta_3) + \mathcal{L}_2(\mathcal{B}_1;\theta_1) + \mathcal{L}_1(\mathcal{B}_1;\theta_1) + \mathcal{B}_1 \\
\mathcal{B}_3 &= \mathcal{L}_2(\mathcal{L}_1(\mathcal{B}_1;\theta_1);\theta_2) + \\
&\quad \mathcal{L}_2(\mathcal{B}_1;\theta_1) + \mathcal{L}_1(\mathcal{B}_1;\theta_1) + \mathcal{B}_1 \\
\mathcal{B}_2 &= \mathcal{L}_1(\mathcal{B}_1;\theta_1) + \mathcal{B}_1 \\
\mathcal{B}_1 &= \mathcal{I} \text{ which is the input image.}
\end{aligned}$$

Residual connections



Batch normalization

- Large numbers in blocks cause problems

Numbers with large magnitude in a neural network cause problems. Imagine some input to some unit is big and the weight applied to that input is small. Then a single gradient step could cause the weight to change sign, and the ReLU might cause the corresponding output to swing between strongly positive and zero. This can cause training problems, because the gradient will be a poor predictor of what will actually happen to the output. Ideally, relatively few values at the input of

Batch normalization

Write $\mathcal{I}$ for the input of this layer, which is a $X \times Y \times F$ block of features, and $\mathcal{O}$ for its output, which is a block of features of the same dimension. The layer has two vectors of parameters, γ and β, each of dimension F. Write γ_i for the i'th component of γ, etc. Assume we know the mean (m_k) and standard deviation (s_k) of each feature in $\mathcal{I}$ computed over the whole dataset and over the spatial dimensions. Write ϵ for a small positive number chosen to avoid divide-by-zero. The data block $\mathcal{U}$, with ijk'th component

$$\mathcal{U}_{ijk} = \frac{(\mathcal{I}_{ijk} - m_k)}{(s_k + \epsilon)}$$

will tend to have small magnitude numbers in it, both positive and negative. The mean of each feature in this block should be about zero, because it is close to the mean over all blocks. The standard deviation of each feature in this block should be about one, because it is close to the standard deviation over all blocks. Now compute

$$\mathcal{O}_{ijk} = \gamma_k \mathcal{U}_{ijk} + \beta_k$$