

example, suppose we have a deep neural network that has only one unit per layer and does not use an activation function at each hidden layer: $\hat{y} = xw_1w_2w_3 \dots w_l$. Here, w_i provides the weight used by layer i . The output of layer i is $h_i = h_{i-1}w_i$. The output \hat{y} is a linear function of the input x , but a nonlinear function of the weights w_i . Suppose our cost function has put a gradient of 1 on \hat{y} , so we wish to decrease \hat{y} slightly. The back-propagation algorithm can then compute a gradient $\mathbf{g} = \nabla_{\mathbf{w}}\hat{y}$. Consider what happens when we make an update $\mathbf{w} \leftarrow \mathbf{w} - \epsilon\mathbf{g}$. The first-order Taylor series approximation of \hat{y} predicts that the value of \hat{y} will decrease by $\epsilon\mathbf{g}^\top\mathbf{g}$. If we wanted to decrease \hat{y} by .1, this first-order information available in the gradient suggests we could set the learning rate ϵ to $\frac{.1}{\mathbf{g}^\top\mathbf{g}}$. However, the actual update will include second-order and third-order effects, on up to effects of order l . The new value of \hat{y} is given by

$$x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2) \dots (w_l - \epsilon g_l). \quad (8.34)$$

An example of one second-order term arising from this update is $\epsilon^2 g_1 g_2 \prod_{i=3}^l w_i$. This term might be negligible if $\prod_{i=3}^l w_i$ is small, or might be exponentially large if the weights on layers 3 through l are greater than 1. This makes it very hard to choose an appropriate learning rate, because the effects of an update to the parameters for one layer depends so strongly on all of the other layers. Second-order optimization algorithms address this issue by computing an update that takes these second-order interactions into account, but we can see that in very deep networks, even higher-order interactions can be significant. Even second-order optimization algorithms are expensive and usually require numerous approximations that prevent them from truly accounting for all significant second-order interactions. Building an n -th order optimization algorithm for $n > 2$ thus seems hopeless. What can we do instead?

Batch normalization provides an elegant way of reparametrizing almost any deep network. The reparametrization significantly reduces the problem of coordinating updates across many layers. Batch normalization can be applied to any input or hidden layer in a network. Let \mathbf{H} be a minibatch of activations of the layer to normalize, arranged as a design matrix, with the activations for each example appearing in a row of the matrix. To normalize \mathbf{H} , we replace it with

$$\mathbf{H}' = \frac{\mathbf{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \quad (8.35)$$

where $\boldsymbol{\mu}$ is a vector containing the mean of each unit and $\boldsymbol{\sigma}$ is a vector containing the standard deviation of each unit. The arithmetic here is based on broadcasting the vector $\boldsymbol{\mu}$ and the vector $\boldsymbol{\sigma}$ to be applied to every row of the matrix \mathbf{H} . Within each row, the arithmetic is element-wise, so $H_{i,j}$ is normalized by subtracting μ_j