

and dividing by σ_j . The rest of the network then operates on \mathbf{H}' in exactly the same way that the original network operated on \mathbf{H} .

At training time,

$$\boldsymbol{\mu} = \frac{1}{m} \sum_i \mathbf{H}_{i,:} \quad (8.36)$$

and

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m} \sum_i (\mathbf{H} - \boldsymbol{\mu})_i^2}, \quad (8.37)$$

where δ is a small positive value such as 10^{-8} imposed to avoid encountering the undefined gradient of \sqrt{z} at $z = 0$. Crucially, *we back-propagate through these operations* for computing the mean and the standard deviation, and for applying them to normalize \mathbf{H} . This means that the gradient will never propose an operation that acts simply to increase the standard deviation or mean of h_i ; the normalization operations remove the effect of such an action and zero out its component in the gradient. This was a major innovation of the batch normalization approach. Previous approaches had involved adding penalties to the cost function to encourage units to have normalized activation statistics or involved intervening to renormalize unit statistics after each gradient descent step. The former approach usually resulted in imperfect normalization and the latter usually resulted in significant wasted time as the learning algorithm repeatedly proposed changing the mean and variance and the normalization step repeatedly undid this change. Batch normalization reparametrizes the model to make some units always be standardized by definition, deftly sidestepping both problems.

At test time, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ may be replaced by running averages that were collected during training time. This allows the model to be evaluated on a single example, without needing to use definitions of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ that depend on an entire minibatch.

Revisiting the $\hat{y} = xw_1w_2 \dots w_l$ example, we see that we can mostly resolve the difficulties in learning this model by normalizing h_{l-1} . Suppose that x is drawn from a unit Gaussian. Then h_{l-1} will also come from a Gaussian, because the transformation from x to h_l is linear. However, h_{l-1} will no longer have zero mean and unit variance. After applying batch normalization, we obtain the normalized \hat{h}_{l-1} that restores the zero mean and unit variance properties. For almost any update to the lower layers, \hat{h}_{l-1} will remain a unit Gaussian. The output \hat{y} may then be learned as a simple linear function $\hat{y} = w_l \hat{h}_{l-1}$. Learning in this model is now very simple because the parameters at the lower layers simply do not have an effect in most cases; their output is always renormalized to a unit Gaussian. In some corner cases, the lower layers can have an effect. Changing one of the lower layer weights to 0 can make the output become degenerate, and changing the sign