Where does $\frac{1}{\sqrt{d}}$ come from

Akash Garg

Version 2.0 - April 12, 2023

Among the many surprising things about optimizing neural networks, especially large/deep neural networks is the sensitivity to initialization. Having done a fair bit of computational science work in the past, I understand the need for good initializations. Most nonlinear optimization routines are sensitive to the quality of initialization to get to the ”correct” solution – mostly b/c most optimizers will locally linearize a nonlinear function; thus the closer you are to the solution the faster and more accurately you'll converge to the desired solution [3].

In neural network initialization, we linearize the problem using gradient descent and also run into problems with maintaining neuron activations to ensure that magnitude of the gradients is greater than zero. If gradients vanish to zero, no progress is made in the optimization/learning.

Common practice dictates that weights shouldn't be initialized to zero. This is mostly to avoid symmetry in the gradient propagation. Small random perturbations around zero is also not great and can lead to slow learning since gradients will be proportional to the weight values; which can be problematic for deep networks.

An interesting finding by Glorot and Bengio [1] is that in a simple initialization of weights based on $\frac{1}{\sqrt{n_{in}}+\sqrt{n_{out}}}$ results in deeper layers not saturating close to zero. The analysis is based on a linear deep neural network where we want the variances of the inputs to match variance of the weights. This is easier to see for a single linear layer. Consider a single layer like:

$$\boldsymbol{s} = z\boldsymbol{W} + \boldsymbol{b}$$

Then we would like to keep the same variance in the activations, like so:

$$\begin{aligned}
Var[\boldsymbol{s}] &= \sum_i Var[z_i w_i + b_i] \\
&= \sum_i Var[z_i w_i] \\
&= \sum_i E[w_i]^2 Var[z_{]} + E[z_{]}^2 Var[w_i] + Var[w_i]Var[z_i] \\
&= \sum_i Var[z_i]Var[w_i] \\
&= nVar[z]Var[w]
\end{aligned}$$

where $Var[z]$ and $Var[w]$ are the shared scalar variances for all weights in the layer. We can also assume that $E[z] = E[w] = 0$. Since we would want the variances to match, we want to satisy $Var[\boldsymbol{s}] = Var[z]$, which can be satisfied when $Var[w] = \frac{1}{n}$. This is achieved by scaling the values for $w = \frac{1}{\sqrt{n}}$, resulting in $Var[\frac{1}{\sqrt{n}}w] = \frac{1}{n}w]$ where n is dimensions in w and using the property that $Var[aX] = a^2Var[X]$.

However, this analysis ignores the ReLU activation function that is mostly used in training neural networks today. A similar analysis was followed up by Kaiming He, et. al. [2] but taking into account the ReLU activation function for very deep networks:

$$\begin{aligned} Var[s_l] &= n_l Var[w_l z_l] \\ &= n_l Var[w_l] E[z_l^2] \end{aligned}$$

Note that $E[z^2] \neq Var[z]$ when we use ReLU since $x = max(0, x)$ does not have zero mean.

Assuming w_{l-1} is symmetric around zero and $b_{l-1} = 0$, then the preactived layer $s_{l-1} = w_{l-1}z_{l-1} + b_{l-1}$ will also be symmetric around zero assuming z_{l-1} is also symmetric around zero.

Applying ReLU will set all negative values for s_{l-1} which breaks the symmetry in the post activation output. For s_{l-1} ReLU will only keep half the values (positive side) and sets the other half (negative side) to zero. The symmetry around zero implies that half the values are positive and half the values are negative. This truncation will modify the variance like so:

$$E[z_l^2] = \frac{1}{2}Var[s_{l-1}]$$

where,

$$z_l = max(0, s_{l-1}).$$

ReLU effectively eliminates the negative half, halving the variance because the expectation integrates only over the positive values (which cover half the original distribution).

Plugging this back into the above we get

$$Var[s_l] = \frac{1}{2}n_l Var[w_l]Var[s_{l-1}].$$

Recursively, with L layers, we get:

$$Var[s_L] = Var[s_1]\prod_{l=2}^{L}\frac{1}{2}n_l Var[w_l].$$

Using this Kaiming initialization takes this result and sets

$$\frac{1}{2}n_l Var[w_l] = 1.$$

as a sufficient condition to avoid reducing or magnifying the magnitudes of input signals exponentially. We can then solve for $Var[w_l] = \frac{2}{n_l}$, which then implies that w_l should be scaled by $\frac{2}{\sqrt{n_l}}$.

One wouldn't think that this subtle difference in initialization would make a difference but He et al.show that the Xavier initialization stalls during training of large networks while Kaiming initialization does converge even very deep models and is the common practice in weight initialization today.

There is one more place where this kind of initialization is seen, namely in Transformers. The attention layers are scaled by $\frac{1}{\sqrt{d_k}}$ where d_k is the dimension of the query and keys:

$$\text{Attention} = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V.$$

This scaling is again used here to avoid large magnitude vectors which will then cause softmax to peak. Larger dimension vectors mean that there are more terms in the dot product, which increases the variance of the logits. The scaling here will again attempt to normalize the effect of large variances similar to Xavier / Kaiming initialization.

References

[1] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[2] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[3] *Newton Fractal*. URL: https://en.wikipedia.org/wiki/Newton_fractal.

Revision	**Date**	**Description**
1.0	December 7, 2020	Initial draft
2.0	April 12, 2023	Adding bit about transformers