Chapter 1

Parameters Initialization

1.1 Weights

We normally want the weights to be close to zero because it is reasonable to think that half of the wights will be positive and half of them will be negative.

Warning: Its not necessarily the case that smaller numbers will work strictly better. For example, a Neural Network layer that has very small weights will during backpropagation compute very small gradients on its data (since this gradient is proportional to the value of the weights). This could greatly diminish the gradient signal flowing backward through a network, and could become a concern for deep networks. It would bring all the activations towards 0.

If the weight are initialized with a to high value the activations will explode and saturate to 1.

W=0

There is no balance break. If every neuron in the network computes the same output, then they will also all compute the same greadients during backpropagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if thir weights are initialized to be the same.

Small random numbers

Every neurons weight vector is initialized as a random vector sampled from a multi-dimensional gaussian, so the neurons point in random direction in the input space. It is also possible to use small numbers drawn from a uniform distribution, but this seems to have relatively little impact on the final performance in practice.

One of the problems of this method is to decide how small should the gradients be. Example assuiming tanh:

- `W = 0.01 * np.random.randn(D,H)` : might be too small, all activations become zero, therefore, no gradient accumulation.
- `W = 1* np.random.randn(D,H)` : might be too large, almost all neurons completely saturated, thus gradients will be zero.

Another problem is that the distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs.

Xavier initialization

eq: `w = np.random.randn(n) / sqrt(n)`

eq for ReLU: `w = np.random.randn(n) / sqrt(2n)`

It is possible to solve the distribution problem of the above method by normalizing the variance of each neurons output to 1 by scaling its weight vector by the square root of its fan-in (i.e. its number of inputs). This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

Mathematical reasoning

Consider the inner product $s = \sum_i^n w_i x_i$ between the weights and input , which gives the raw activation of a neuron before the non-linearity. We can examine the variance of s:

$$\begin{aligned}
\text{Var}(s) &= \text{Var}(\sum_i^n w_i x_i) \\
&= \sum_i^n \text{Var}(w_i x_i) \\
&= \sum_i^n [E(w_i)]^2 \text{Var}(x_i) + \text{E}[(x_i)]^2 \text{Var}(w_i) + \text{Var}(x_i)\text{Var}(w_i) \\
&= \sum_i^n \text{Var}(x_i)\text{Var}(w_i) \\
&= (n\text{Var}(w))\text{Var}(x)
\end{aligned} \tag{1.1}$$

where in the first 2 steps we have used properties of variance. In third step we assumed zero mean inputs and weights, so $E[x_i] = E[w_i] = 0$. Note that this is not generally the case: For example ReLU units will have a positive mean. In the last step we assumed that all x_i, w_i are

identically distributed. From this derivation we can see that if we want s to have the same variance as all of its inputs xx, then during initialization we should make sure that the variance of every weight w is $1/n$. And since $\text{Var}(aX) = a^2\text{Var}(X)$ for a random variable X and a scalar a, this implies that we should draw from unit Gaussian and then scale it by $a = \sqrt{1/n}$, to make its variance $1/n$. This gives the initialization `w = np.random.randn(n) / sqrt(n)`.

A similar analysis is carried out for ReLUs, reaching the conclusion that the variance of neurons in the network should be . This gives the initialization `w = np.random.randn(n) / sqrt(2n)`, and is the current recommendation for use in practice in the specific case of neural networks with ReLU neurons.

Sparse Initialization

Another way to address the uncalibrated variances problem is to set all weight matrices to zero, but to break symmetry every neuron is randomly connected (with weights sampled from a small gaussian as above) to a fixed number of neurons below it. A typical number of neurons to connect to may be as small as 10.

1.2 Biases

It is possible and common to initialize the biases to be zero, since the asymmetry breaking is provided by the small random numbers in the weights. For ReLU non-linearities, some people like to use small constant value such as 0.01 for all biases because this ensures that all ReLU units fire in the beginning and therefore obtain and propagate some gradient. However, it is not clear if this provides a consistent improvement (in fact some results seem to indicate that this performs worse) and it is more common to simply use 0 bias initialization.

Summary

The current recommendation is to use ReLU units and use the Xavier initialization `w = np.random.randn(n) * sqrt(2.0/n)` and all biases equal to 0 or 0.01. Also batch normalization helps having less headaches with weight initialization.