PERPETUAL ENIGMA

PERENNIAL FASCINATION WITH ALL THINGS TECH

Understanding Xavier Initialization In Deep Neural Networks

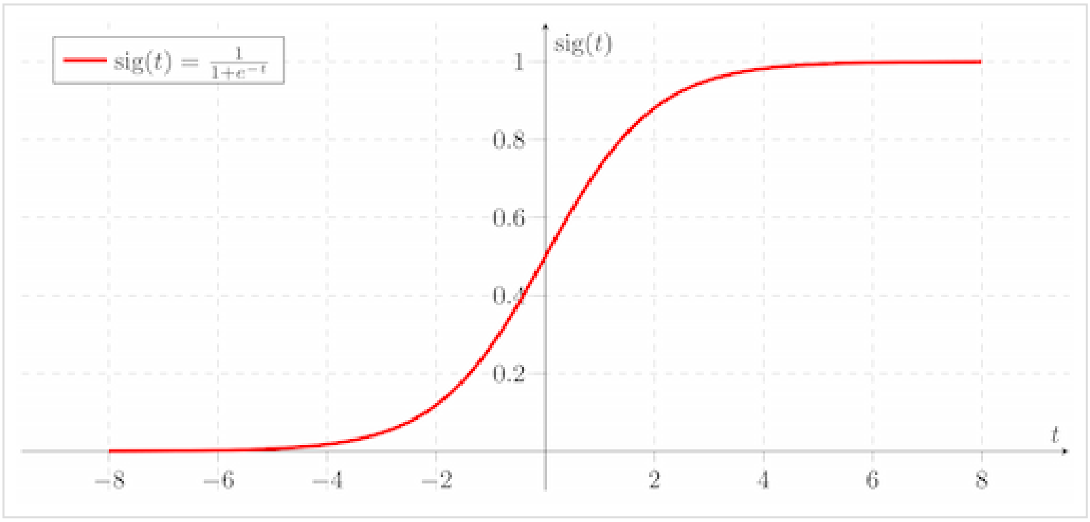
Posted on March 29, 2016

I recently stumbled upon an interesting piece of information when I was working on deep neural networks. I started thinking about initialization of network weights and the theory behind it. Does the image to the left make sense now? The guy in that picture is lifting "weights" and we are talking about network "weights". Anyway, when we implement convolutional neural networks, we tend to utilize all the knowledge and research available out there. A good number of things in deep learning are based on heuristics! It's worth exploring why we do things in a certain way whenever it's possible. This goes a long way in unlocking the hidden mysteries of deep learning and why it's so unbelievably accurate. Let's go ahead and understand how network weights are initialized, shall we?

Why do we need initialization in the first place?

When you are working with deep neural networks, initializing the network with the right weights can be the difference between the network converging in a reasonable amount of time and the network loss function not going anywhere even after hundreds of thousands of iterations.

If the weights are too small, then the variance of the input signal starts diminishing as it passes through each layer in the network. The input eventually drops to a really low value and can no longer be useful. Now why is that a problem? Let's consider the sigmoid function:



If we use this as the activation function, then we know that it is approximately linear when we go close to zero. This basically means that there won't be any non-linearity. If that's the case, then we lose the advantages of having multiple layers.

If the weights are too large, then the variance of input data tends to rapidly increase with each passing layer. Eventually it becomes so large that it becomes useless. Why would it become useless? Because the sigmoid function tends to become flat for larger values, as we can see the graph above. This means that our activations will become saturated and the gradients will start approaching zero.

Initializing the network with the right weights is very important if you want your neural network to function properly. We need to make sure that the weights are in a reasonable range before we start training the network. This is where Xavier initialization comes into picture.

What exactly is Xavier initialization?

Assigning the network weights before we start training seems to be a random process, right? We don't know anything about the data, so we are not sure how to assign the weights that would work in that particular case. One good way is to assign the weights from a Gaussian distribution. Obviously this distribution would have zero mean and some finite variance. Let's consider a linear neuron:

$$y = w_1x_1 + w_2x_2 + ... + w_Nx_N + b$$

With each passing layer, we want the variance to remain the same. This helps us keep the signal from exploding to a high value or vanishing to zero. In other words, we need to initialize the weights in such a way that the variance remains the same for x and y. This initialization process is known as Xavier initialization. You can read the original paper here.

How to perform Xavier initialization?

Just to reiterate, we want the variance to remain the same as we pass through each layer. Let's go ahead and compute the variance of y:

$$var(y) = var(w_1x_1 + w_2x_2 + ... + w_Nx_N + b)$$

Let's compute the variance of the terms inside the parentheses on the right hand side of the above equation. If you consider a general term, we have:

$$var(w_ix_i) = E(x_i)^2var(w_i) + E(w_i)^2var(x_i) + var(w_i)var(x_i)$$

Here, E() stands for expectation of a given variable, which basically represents the mean value. We have assumed that the inputs and weights are coming from a Gaussian distribution of zero mean. Hence the "E()" term vanishes and we get:

$$var(w_ix_i) = var(w_i)var(x_i)$$

Note that 'b' is a constant and has zero variance, so it will vanish. Let's substitute in the original equation:

$$var(y) = var(w_1)var(x_1) + ... + var(w_N)var(x_N)$$

Since they are all identically distributed, we can write:

$$var(y) = N * var(w_i) * var(x_i)$$

So if we want the variance of y to be the same as x, then the term "N * var(wi)" should be equal to 1. Hence:

$$N * var(w_i) = 1$$
$$var(w_i) = 1/N$$

There we go! We arrived at the Xavier initialization formula. We need to pick the weights from a Gaussian distribution with zero mean and a variance of 1/N, where N specifies the number of input neurons. This is how it's implemented in the Caffe library. In the original paper, the authors take the average of the number input neurons and the output neurons. So the formula becomes:

$$var(w_i) = 1/N_{avg}$$
$$\text{where } N_{avg} = (N_{in} + N_{out})/2$$

The reason they do this is to preserve the backpropagated signal as well. But it is more computationally complex to implement. Hence we only take the number of input neurons during practical implementation.

——

RELATED

Understanding Locally Connected Layers In Convolutional Neural Networks
In "Machine Learning"

Deep Learning For Sequential Data – Part I: Why Do We Need It
In "Machine Learning"

What Is Local Response Normalization In Convolutional Neural Networks
In "Machine Learning"

This entry was posted in **Machine Learning** and tagged **Artificial Intelligence**, **Convolutional Neural Networks**, **Deep Learning** by **Prateek Joshi**. Bookmark the **permalink [https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/]** .

6 THOUGHTS ON "UNDERSTANDING XAVIER INITIALIZATION IN DEEP NEURAL NETWORKS"

TR
on **November 17, 2016 at 4:18 AM** said:

Beautifully explained, thank you!

Vinh
on **January 22, 2017 at 10:01 AM** said:

Thanks for your explaination. This is really helpfull.

Wolf
on **February 22, 2017 at 9:28 PM** said:

“where N specifies the number of input neurons”. So to satisfy my OCD, a neuron whose inputs are two other neurons and a bias node from the previous layer”, N = 2?

focusedwolf
on **February 22, 2017 at 9:29 PM** said:

So for a neuron whose inputs are two other neurons and a bias node from the previous layer, N = 2?

Pingback: About xavier_initializer in tensorflow – program faq

Pingback: AI Starter– Build your first Convolution neural network in Keras from scratch to perform… – AI+ NEWS