


Machine Learning for Engineering and Science Applications
Professor Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology Madras
Learning Rate decay & Weight initialization


(Refer Slide Time: 0:14)



Learning Rate Decay

- Most techniques tend to decrease the learning rate as the number of iterations increase.
- High learning rates-parameter values would vary rapidly and by large amounts and would not settle down in a local minima
- A low learning rate would lead to small updates and convergence to a false minimum.

α alpha $\rightarrow \alpha = \alpha - \alpha \frac{\partial L}{\partial \alpha}$
 $W \leftarrow W - \alpha \frac{\partial L}{\partial W}$
Epoch / iteration



And the other important topic that we would like to look at before we go into, how to initialise weights is the learning rate decay, again here the idea is because the neural networks, it is a very complicated function of the weights, it is very difficult to say when we are near optimal, minima and we are some very poor minima, so that must be some systematic way of changing your way on the learning rate, if this is basically your alpha that you are pretty much used to that is, you must have seen X goes to X minus alpha delta L over delta X or in this case weights, your use to weights.

So your W minus alpha delta L over delta W right, so this alpha okay, so at this alpha in this many what we typically do is to where it is alpha with every iteration or epoch, so what you mean by epoch? Epoch is when you are gone through your entire data set what is? That is one epoch right, let say your seen stack so castigated in descend or many batch credit descend, let run through our entire dataset and that will be considered as one epoch, so from epoch to epoch you can vary your learning rate.

This is important because that this learning rate will dictate how much to change your parameter spiking right because that the magnitude of the update also depends on not only on the gradient of the last function of the respect to the weights but also on the learning rate

alpha, so by moderating alpha you can also moderate the magnitude of your updates. Okay, so what happens is that when you have very high learning rates right, so let say alpha is very large number it settles down is to 1, then your parameter values would vary rapidly okay, they would vary rapidly and by large amounts and would not settle down in local minima.

However lower learning rate would lead to slow learning which means that they will not be, parameters want change rapidly and it is quite possible that they can stuck in, out false minima okay, so then this, there is no good way to know when to do what but typically there are techniques that people use is to decrease the learning rate as the number of iterations or epochs increase.

(Refer Slide Time: 2:55)



Learning rate decay

- Reduce the learning rate by a constant factor every epoch ^{or} every k epochs.
- Alternatively, you can check validation error and reduce learning rate by a factor k every time the validation error drops



So there are many ways to do that right, so one of them would be to reduce the learning rate by a constant factor every epoch or K every epochs, so there is spelling mistake here or every K epochs, so this is again decided by you, you have to again, this is, consider this as another hyper parameter that you have to optimise right, another way would be to check the valid, so you have validation data and training data, so you check the performance on the validation data and whenever the performance on the validation data the improves you decline, you decrease the learning rate by a certain fixed at factor okay, what that factor is? Of course has to be determined by try an error or some systematic search that we saw earlier.

(Refer Slide Time: 3:45)



Learning rate decay

- For a smooth decrease in learning rate over epochs you can use

$$\alpha = \frac{\alpha_0}{1 + \text{decay} * \text{epoch}}$$



Okay, one of the more automated ways also to ensure the very smooth decay, so for instances we in the previous techniques we saw you manually, not manually you set, you decrease the learning rate pie fixate amount, by a fixate factor every epoch, here this is more smooth way of decreasing the learning rate right, so this is your initial learning rate, which you set again, this is another hyper parameter that you have to figure out, divided by 1 plus there is some decay rate times the epoch number okay.

So as the number of epoch increase depending on the magnitude of the decay, your initial learning rate will also decrease okay, so this are very small way of decreasing your learning rate, there are also exponential schemes for taking your learning rates, so that is also possible, many of these software packages have these implemented as block boxes, you are welcome to use, of course you should know what you doing, this is the reason why we are explaining this here okay.

(Refer Slide Time: 4:52)



Weight Initialization

- There are a large number of weights in a neural network leading to a large search space for minima of the loss function
- Weights must be effectively randomly initialized to prevent convergence to false minima.



The next topic is weight initialization okay, so this is again a very important topic because if you do initialization the weights properly, then you will not get good convergence, you will see why that happens and then you have to keep, you know trying out different starting points to train a network again and again to before you finally hit on something, some initialization, that actually works. Okay.

So why it is important because in a typical neural network there are large number of its. Okay, hundreds of, thousands of weights for very small ANN going up to millions and millions of weights or large neural networks right, so which leads to a large search spaces, so that spaces is because we have to look, you to minimise your loss function and loss function is a function of your weights, so it is a multidimensional problem that you are solving and so if you want to optimize that loss function than its a very large search space of weights

So you are looking for a combination of one hundred million weights or more which will actually work and you want to affect, you want to weights must be initialized randomly, to effectively randomly to because to prevent convergence of false minima and also text your entire space, just as we saw as to how we have to pick your range of hyper parameters for, you know for hyper parameter optimization, we also have to pick up, we also have to figure out the range of weights that we, for initialization and so that the entire space of space function is covered okay, to some extent at least and that is required for the effective performance for neural network.

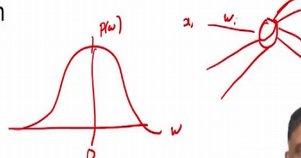
(Refer Slide Time: 6:40)

Weight Initialization



- Naïve initialization would involve sampling weights from a gaussian distribution with zero mean and unit variance
- Assume that input data x has also been appropriately normalized to zero mean and unit standard deviation

$w_i \rightarrow 0$ -mean 1-variance
 $x_i \rightarrow 0$ -mean 1-variance
 $x_i \rightarrow$ independent
 $w_i \rightarrow$ independent



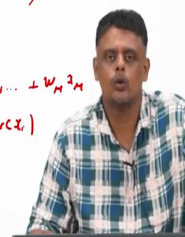
$$z = \frac{1}{M} \sum w_i x_i$$

$$z = w_1 x_1 + w_2 x_2 + \dots + w_M x_M$$

$$\text{Var}(z) = M \text{Var}(w) \text{Var}(x)$$

$$= M$$

N_{in}, N_{out}



So how is this typically done and what are the problems, your naive initialization would involve a sampling weights from a gaussian distribution or uniform random distribution with zero mean and unit variance right and so your distribution of weights for 1 weight let say will or bunch of weights will look like this, so this is the W and this is probability of W . Okay, typically this is a distribution from which you will pick right.

Now we can do this but there are some problems associated with this and will see what those problems are okay, assume that your input data X has also been normalized, so the number we normalized it that is called normalization to have it zero mean and unit standard deviation and we are also taking weights, so that they have zero mean and so your weights W_i have zero mean and unit variance, your input features X_i also have zero mean and unit variance right.

This, you know, we also assume that the individual X_i or independent W_i or independent, note that this might need not be, this did not be an individual X_i being independent need not be true and subsequently the W_i also need not be independent, especially in the case of images where the structure, but in most cases this is true. Okay, so then what happens, so let say we have 1 layer, first input layer as X_i and we have let say M features right.

So your linear combination would give raise to summation $W_i X_i$ right, summation 1 over M and the summation of this right, so we just write this out, so that will be this, let call this output Y right or actually I stick to the notation and right, so Z is, will ignore the bias okay, W not, Z is $W_1 X_1$ plus $W_2 X_2$ plus 1 so for plus $W_M X_M$, M is number of features, let us take

me about the first layer right, so the variance of Z it turns out under these assumptions that W_i are independent or X_i are independent, $W_i X_i$ are independent of each other, it turns out that the variance of Z would be M times the variance of W s and times the variance of X okay, that is what happens okay, which is nothing but approximately, so in this case M okay.

So what does it mean, so what is the implication? So the implication is that if you randomly sampled W s from a gaussian with zero mean and unit standard deviation and you do and of course you have normalized some features also to be that way, then the variance on your sum that goes into the Z number is the input to your sigmoid function, let say, let us consider sigmoid function just for sake of involvement M is input to your sigmoid function, then what happens we saw that for large values of Z which is possible right, because the variance of Z is M times something, so you can take a very large values, so in this case, the sigmoid would saturate.

So during back propagation the derivative would be zero and which means that the rates will not get updated, so you are stuck okay, so this is the problem with drawing from, so but of course people have been doing this and so you will have to do many trials in errors, so that some point will get one good combination which will give you do good back propagation okay, so this is just looking at it from one input to a neuron right, so what we have discussed is this is one neuron somewhere in the first layer all of these weights, these are the X_i and these are the W_i leading into it, also recall that there will be neuron is going from outside of them also. Okay.

So if you have a, say in this case I have used the M features because we are assuming that I am and the input layer and typically we would see instead of M I will say N in which is basically the number of neuron, number of ways that are feeding into, number of neurons that are feeding into a another neuron in the next layer okay, other terminology is N out which is the number of neurons or the waves emanating from a neuron in a layer okay, so that is you have to keep track on that.

So we have established that the variance of Z , which is, there is a linear combination of the inputs to a neuron time and tabulate the weights, so it is M times the M , it is pretty much M , so it makes sense, if we scale our choice of weights by M , so the variance should be, so we when try to sample, will sample from distribution is zero mean and variance 1 over M , that makes sense, so that is what we will do. Okay.

(Refer Slide Time: 12:16)

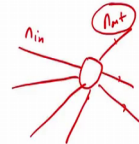


Xavier Initialization

- Here the initialization draws from a distribution with zero mean and a specified variation.

$$\text{var}(w) = \left[\frac{2}{n_{in} + n_{out}} \right]$$

$$\text{var}(w) = \left(\frac{1}{n_{in}} \right)$$



So the variance of W got from the previous layer argument would be got 1 over in this N in I call it. Okay, which is basically the number of neurons that are feeding into a particular neuron, so from a previous layer okay, so this is N in, of course you know that the number of neurons also, number of weights coming in from the neurons in the previous layer and of course from each neuron you have inputs going to multiple neurons is output right, so this number of output here we call it N out okay

Now we only looked at the forward pass, in the previous arguments we have only looked at for pass what happens? Now it turns out that if you consider the backward pass, that is basically when you are doing back propagation and you want to preserve gradients in, so then you have to keep track of when you have to consider this number also right, because the gradients feed in to each of this neurons from N out number of neurons right.

So then what this xavier initializations does based on the author of the paper who proposes initializations is to make sure the variance of W is the average of N in and N out, so it is scaled by an average of N in and N out, so which means that 2 over that. Okay, this is the solution, the often it turns out that it works very well for a wide variety of problems okay, so in both this arguments the idea is that when you have, it considered the weights and independent and the features are independent, turns out that the variants of the linear combination of them scales as the number of them, number of the weights or the number of neurons and to take that into account you scale the variance which you, scale the variance from the distribution from which you sample the weights by an appropriate factor okay.

So these are commonly used in practice, xavier initialisation is very commonly used in practice and so is this one, the $\frac{1}{N}$ in is also commonly used okay, so both of them give very good results for, you know convergence of a network very quickly, so all of these how they help is that, they help fast training okay, otherwise they will take for our to converge because you are the saturation than the gradients become zero, because of the saturation and finally back prop does not update the rates is effectively and so the learning it comes very low, by doing this investigation will have effective learning.

So this is our lecture on hyper parameter optimization, data normalization, data scaling as it call as well as, weight initialization okay, you have question, please post them on the form.