

Practical Machine Learning with TensorFlow
Dr. Ashish Tendulkar Google
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Lecture – 06
Gradient Descent Variations

[FL]. In the last class, we studied Gradient Descent. We understood the intuition behind gradient descent and looked at broad steps that are involved in gradient descent. In this class, we will continue to understand gradient descent. Some of the nuances of gradient descent and some of the variations of gradient descent that are being employed in the practical machine learning training. Let us begin.

(Refer Slide Time: 00:51)

Gradient Descent : b, w_1, w_2, \dots, w_m

$y = b + \sum_{i=1}^m w_i x_i$

(i) Randomly initialize b, w_1, w_2, \dots, w_m

(ii) Repeat until convergence:

(iii) Predict $\hat{y}^{(i)}$ for each data point in training

(iv) Calculate loss $J(b, w)$


(v) Calculate gradient of $J(b, w)$

(vi) Update b, w_1, w_2, \dots, w_m simultaneously
go to step (i)

(vii) Update b, w_1, w_2, \dots, w_m simultaneously
go to step (i)

$J(b, w) = \frac{1}{2} \sum_{i=1}^n (h_{w,b}(x^{(i)}) - y^{(i)})^2$
 \uparrow
 $\hat{y}^{(i)}$

$b^{(new)} := b^{(old)} - \alpha \cdot \text{gradient } b$
 $w_i^{(new)} := w_i^{(old)} - \alpha \cdot \text{gradient } w_i$
 $w_m^{(new)} := w_m^{(old)} - \alpha \cdot \text{gradient } w_m$



So, this is where we stopped in our previous class. So, let us quickly recap this particular scheme. So, we looked at gradient descent which has got 7 steps. So, we randomly initialize the parameter values. Then, we repeat until convergence, the following steps. The randomly initialized parameter values give us a point in the lost space, this point will correspond to a model in the model space. Since, we get a model with these parameter values, we can predict the output for each data point to the training using the particular model. Then, we calculate loss which is $J(b, w)$ you know using this particular equation.

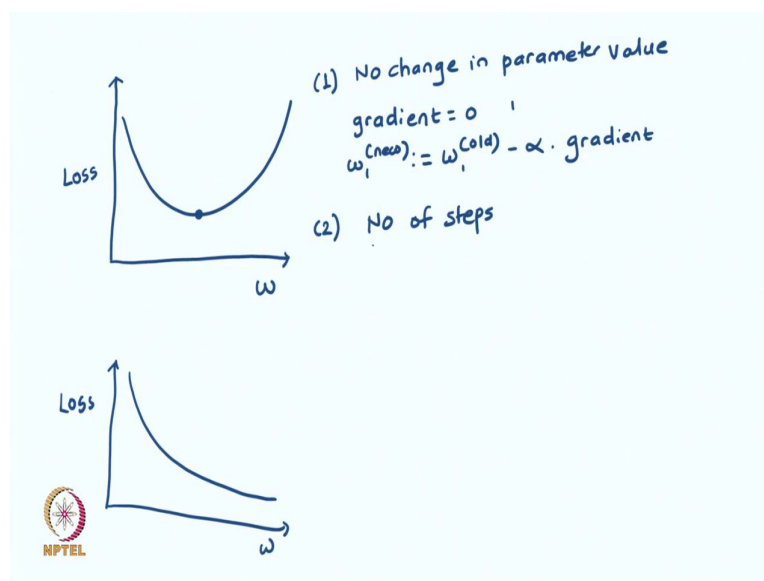
$$J(b, w) = \frac{1}{2} \sum_{i=1}^n (h_{w,b}(x^{(i)}) - y^{(i)})^2$$

b and w are biases and weights respectively, $h_{w,b}$ is the model parameterized by these weights and biases, and $x^{(i)}$ and $y^{(i)}$ are the i^{th} example and the i^{th} example's target respectively.

The loss is calculated as the squared difference between the predicted value and the actual value. We sum it across all the end points and use half as a factor for mathematical convenience and that is how we get our loss function. After calculating the loss function, we calculate gradient of the loss function at those parameter values and we use those gradients to update each of the parameter value to a new parameter value by subtracting the product of learning rate and a gradient with respect to that parameter.

And once we complete this process of calculating gradient with respect to each of the parameters, we update all the parameters to their new values simultaneously and then, we go back to step two, where we go to a new point and we repeat the process. We continue this iteration until the algorithm is converged. So, let us try to understand the notion of convergence and then, we will come back to understand how exactly we can calculate the gradient.

(Refer Slide Time: 03:15)



So, let us try to understand how the convergence works in gradient descent. So, we have loss on y axis and we have the value of parameter on the x axis and it is a loss function in the context of linear regression and this is the minima. So, what happens at minima? So, at minima, value of gradient is 0 and because value of gradient is 0, we do not see any change in the parameter value. Because this is how we calculate the change in the parameter value, we take the old value and we subtract the product of learning rate and gradient from that and if gradient is 0, we do not have anything to subtract, so, value do not change.

So, the first way in which we calculate the convergence is by observing that there is no change in parameter value, either in subsequent iterations or for past few iterations. What if so, we can also apply this gradient descent algorithm on some tricky functions which may not be convex. So, the another safeguard, so what happens is so let us say we have a function which is like a flat function after some time. So, what will happen is we will not see any change in the parameter value after some time or it will be very very slow.

So, we do not want to keep on making the changes and keep running this particular algorithm forever. So, we define a parameter called number of steps for which we want to run this. So, we either check for one of these two conditions and if one of these two conditions are being met, we stop the gradient descent updates. So, the two conditions are - no change in the parameter value in the subsequent iterations or in past few iterations or the number of steps that we decided to run gradient descent algorithm are done. So, we do not do any more updates to the parameter values.

So, this is how we handle convergence part of it. Let us try to understand how do we handle the gradient calculation part of it. As I earlier said gradient is nothing but the derivative of the loss function with respect to the parameter value.

(Refer Slide Time: 06:28)

$$\begin{aligned}
 J(\omega, b) &= \frac{1}{2} \sum_{i=1}^n (h_{\omega, b}(x^{(i)}) - y^{(i)})^2 \quad \omega \Rightarrow \omega_1, \omega_2, \dots, \omega_m, \\
 &\quad b \\
 \frac{\partial}{\partial \omega_1} J(\omega, b) &= \frac{\partial}{\partial \omega_1} \frac{1}{2} \sum_{i=1}^n (h_{\omega, b}(x^{(i)}) - y^{(i)})^2 \\
 &= \frac{1}{2} \cdot 2 \cdot \sum_{i=1}^n (h_{\omega, b}(x^{(i)}) - y^{(i)}) \cdot \frac{\partial}{\partial \omega_1} (h_{\omega, b}(x^{(i)}) - y^{(i)}) \\
 &= \sum_{i=1}^n (h_{\omega, b}(x^{(i)}) - y^{(i)}) \cdot \frac{\partial}{\partial \omega_1} (b + \sum_{j=1}^m \omega_j \cdot x_j^{(i)}) \\
 &= \sum_{i=1}^n (h_{\omega, b}(x^{(i)}) - y^{(i)}) \cdot x_1 \\
 \frac{\partial}{\partial \omega_j} J(\omega, b) &= \sum_{i=1}^n (h_{\omega, b}(x^{(i)}) - y^{(i)}) \cdot x_j
 \end{aligned}$$

So, loss is calculated as the squared difference between the predicted value and the actual value. This is across all the data points. So, there are weights w_1, w_2 all the way up to w_m and you have parameter b . So, what we do is, we calculate the partial derivative with respect to the parameter value. Let us say with respect to w_1 ; partial derivative of the loss with respect to w_1 is $\frac{\partial J(w, b)}{\partial w_1}$.

And so, let us apply derivative operator on both sides and try to calculate the whole thing. So, what we do is get half out and you apply this on this particular function. So, this two gets cancelled with this two. So, we have deliberately added this factor of half to get an advantage of mathematical convenience while calculating the partial derivative value.

So, what we get is the predicted value minus the actual value, the difference and we will do a small expansion here. So, this is going to be nothing but $b + \sum_{j=1}^m (w_j \cdot x_j^{(i)})$. Now, if we calculate the derivative of the whole thing with respect to w_1 , what happens is all other values are constant and derivative of constant is 0. So, only this particular term $w_1 \cdot x_1$ will survive, this term has parameter w_1 with respect to which you are calculating the partial derivative. So, only this particular term, we will survive; for all other terms the derivative will be 0.

So, what happens is we get and in the derivative of this particular term with respect to w_1 is x_1 . So, the derivative of the loss with respect to w_1 turns out to be the summation over difference into the value of the feature x_1 . So, this is the gradient with respect to w_1 .

$$\frac{\partial J(w, b)}{\partial w_1} = \sum_{i=1}^n (h_{w, b}(x^{(i)}) - y^{(i)}) \cdot x_1$$

(Refer Slide Time: 13:03)

The slide contains handwritten mathematical derivations and diagrams. At the top, a boxed equation shows the derivative of the loss with respect to the bias term b :

$$\frac{\partial J(w, b)}{\partial b} = \sum_{i=1}^n (h_{w, b}(x^{(i)}) - y^{(i)}) \cdot 1$$

Below this, the derivative with respect to a weight w_j is derived:

$$\frac{\partial J(w, b)}{\partial w_j} = \sum_{i=1}^n (h_{w, b}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

$$= \sum_{i=1}^k (h_{w, b}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Below the equations, there are two diagrams. The first diagram, labeled "k=1: Stochastic G.D", shows a horizontal line from 1 to n, with a small segment from 1 to k labeled "SGD". The second diagram, labeled "Batch g.d.", shows a horizontal line from 1 to n, divided into segments of size k, with the first segment labeled "Batch g.d." and the total length labeled "n data points". To the right of this diagram, it says "k: batch size", "Mini-batch g.d.", and "n/k batches iterations".

Now, let us also write the equation of gradient with respect to b , the bias term. In case of bias term, what happens is let us write it down. For $\frac{\partial J(w, b)}{\partial b}$, we take the difference between the predicted value minus actual value and we multiply it by 1 for the bias term. So, this is how we calculate it.

$$\frac{\partial J(w, b)}{\partial b} = \sum_{i=1}^n (h_{w, b}(x^{(i)}) - y^{(i)}) \cdot 1$$

Whereas, in case of other all other parameter values, it was something like this $\frac{\partial J(w, b)}{\partial w_j}$. So, you can see that the only difference is that this x_j a in case of each of the parameter. But in case of biased, we have x_j is equal to 1. So, this is how we calculate the gradient with respect to the particular parameter w_j .

$$\frac{\partial J(w, b)}{\partial w_j} = \sum_{i=1}^n (h_{w, b}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

So, you can see that this particular parameter calculation involves summing up over all the input data points, which is denoted by the above equation. So, what it is doing is, it calculates the prediction on each of the input data points and then, it takes the difference between prediction and the actual value on all the data points and sums it across them. So, this particular process, think of doing this on a very large training data set and it becomes a bottleneck.

So, this particular step takes quite a lot of time and hence, we are going to study a couple of techniques in a short while from now, which will help us to do this particular step more efficiently. So, the way we have seen is called batch gradient descent. So, here we take all “n” data points. We consider all n data points to calculate the gradient.

So, we have a couple of variations of this gradient descent idea. So, in one variation, we use a batch of “k” points instead of using all n points. Here, this number k is much smaller than n. What happens now is that instead of calculating the gradient on all input points, this sum gets reduced to k points.

So, we divide our training data into smaller chunks; each chunk has exactly k elements and this k is called as batch size. We normally use batch size which is a number which is power of 2; 32, 64, 512, 1024 etcetera.

So, this particular version of gradient descent is called as mini batch gradient descent. Where only difference is instead of taking all the data points, we select a chunk of k data points and calculate the gradient update. Now, there are a couple of new terms that come into play here. So, in each iteration, we are using exactly k points to calculate the gradient descent update. When we complete one gradient descent update for all the data, we call this as 1 epoch. So, 1 epoch is a full iteration of the complete training data of n points.

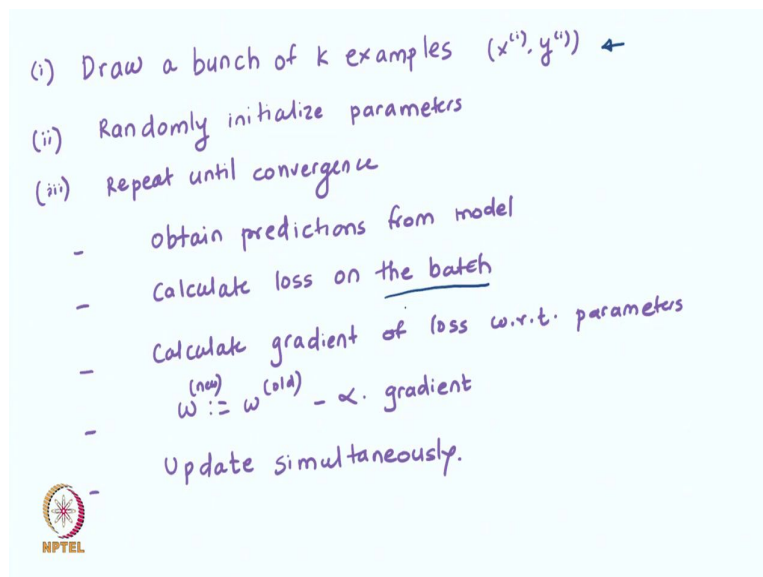
So, in an epoch, we will have n/k batches or n/k iterations. So, in an epoch we will have multiple iterations. So, n by k to be precise and as we will be doing multiple epochs to this, there will be more iterations. Normally we specify the number of iterations in terms of epochs. So, it is very important to understand what epoch is. We also specify the batch size

which is an important parameter in the mini batch gradient descent and you can see when we go into neural network and any other machine learning models also, mini batch gradient descent performs better or does more efficient learning than the batch gradient descent.

So, extending this idea further if you use the value of k . So, if you use k is equal to 1, you have a very special case; where we are making a gradient update just look by looking at a single example and this is called as stochastic gradient descent. So, in the case of stochastic gradient descent the value of k is equal to 1. So, overall, we have this scenario. At size 1, we have SGD and at size n , we have batch gradient descent. Mini batch is somewhere in between.

So, mini batch is preferred over S.G.D or batch gradient descent in general, but if you have extremely large amount of training data even some people use SGD and you will see later in the neural network that we use some variation of SGD to for learning neural networks. So, let us write down the gradient descent process again in the context of mini batch gradient descent and stochastic gradient descent.

(Refer Slide Time: 22:00)




(i) Draw a bunch of k examples $(x^{(i)}, y^{(i)})$ ←

(ii) Randomly initialize parameters

(iii) Repeat until convergence

- obtain predictions from model
- Calculate loss on the batch
- Calculate gradient of loss w.r.t. parameters
- $w^{(new)} := w^{(old)} - \alpha \cdot \text{gradient}$
- Update simultaneously.

 NPTEL

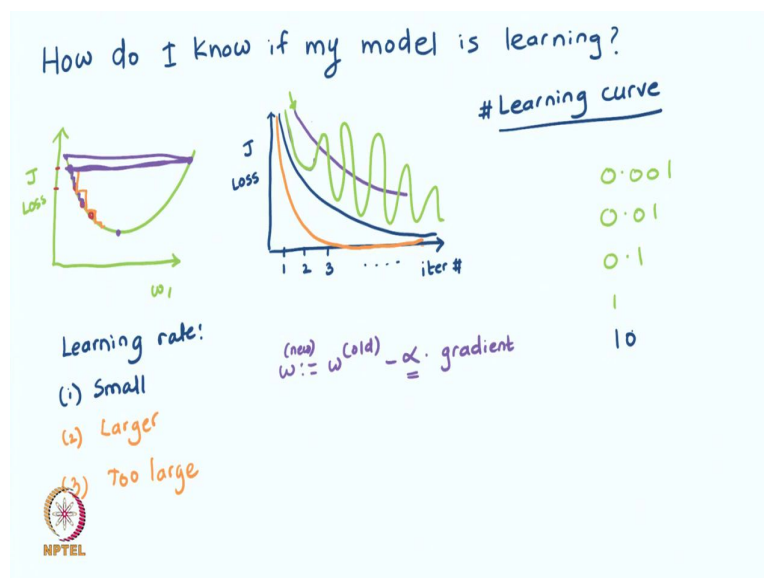
We draw a bunch of k examples; an example contains features and the labels. We first randomly initialize parameters; repeat until convergence. What do we repeat until convergence? We first obtain predictions from the model. Second, we calculate loss on the

batch. Then, we calculate we calculate gradient of loss with respect to each parameter. Next, we update these parameters simultaneously.

So, you can see that the only difference between the gradient descent and batch gradient descent is the first step, where we select a batch of example and we calculate it on the batch. So, the same process can be applied here, if we instead of k example if you select all n examples and if we calculate the loss on the batch of all the examples here, then you know it is a same process as the gradient descent.

So, we studied gradient descent and a couple of its variations. So, this makes us ready to train any machine learning model. So, you can use one of these techniques to train machine learning model. Now, the key question is how do we know whether the machine learning model is getting trained or is it facing some issues and if it is facing some issues, how do we really fix those issues?

(Refer Slide Time: 25:30)



So, remember that in gradient descent what is happening is this is our toy example, where we have a parameter value here and j or loss on y axis and this is our loss function. So, if your model is training correctly; what will happen is we start somewhere and we get updates and with each update, you can see that the loss is coming down. The loss was somewhere over here to begin with when we made this update, the loss became lesser and lesser and further

lesser. So, what happens is that if you plot a graph of the iteration number and the loss. So, ideally what should happen as we train the loss should come down and should come down to 0. Smoothly, it should come down to 0.

In real life unfortunately getting 0 loss is sometimes difficult because of noise in the training data and the noise in the training data occurs due to some issues in labeling or in data collection. So, what happens? So, if you have a learning curve, so these are the iteration numbers at iteration 1, we had very high loss iteration 2, iteration 3 and so on. So, you should see if your model is learning in subsequent iteration, you should see that your loss will gradually decrease.

Now, there are different characteristics of this learning curve. So, this particular curve is called as a learning curve. And this is one particular type of a learning curve, you can see that the learning curve strongly depends on the learning rate. So, learning rate what happens if you have very small learning rate? Then, you will possibly be taking baby steps to reach the minima which is over here. So, in that case your learning will be very very slow. So, your graph will look something like this. It is learning all right, but it is learning very very slowly ok.

What if you have slightly larger learning rate? Then, what might happen is that you will be making longer stride and you will reach obviously, faster. So, your you might come down faster larger in rate; you are you will reach maybe 0 error quite fast

Then, you might wonder if I make learning rate very high; I might be able to reach the minimum in no time, but unfortunately that is not the case. If you have too large; if you have too large learning rate; then, essentially your gradient updates.

$$w^{(new)} = w^{(old)} - \alpha \cdot gradient$$

Now, if the value $\alpha \cdot gradient$ is too large, I might end up crossing the minima to the other side.

And then, when I calculate the gradient here. So, I will probably oscillate again on the other side and I may not converge, I will keep on going from one side of the minima to the other

side. So, what you will observe is either you will not learn or sometimes your loss will come down and sometimes your loss will come down and then it will go up.

So, this kind of learning curve is a problematic learning curve and it tells you that there is something wrong with your learning rate and you should probably reduce your learning rate, if you see this kind of a pattern in your learning rate.

So, what you should ideally do is you should start with some learning rate like 0.001 and then increase it by order of magnitude something like this. So, this is how you can diagnose if your learning rate is too high or learning rate is too small and take corrective measures so that your machine learning algorithm trains efficiently and even before efficiency, you have to make sure that it is training and learning as per your expectations.

So, this brings us to the end of this particular module. In this module, we studied different optimization algorithms mainly of the flavor of gradient descent and its variations and we also understood how to monitor and diagnose problems in the learning or problems during the training and how to fix them. See you in the next session. [FL].