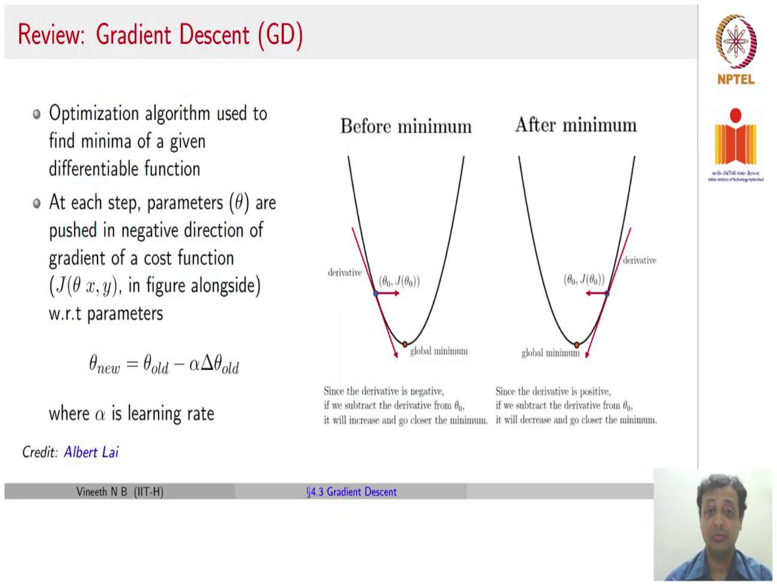
Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad
Gradient Descent and Variants – Part 01

We have so far seen an Introduction to Neural Networks and how one can train a feed-forward neural network or a multi-layer perceptron using back propagation and gradient descent. We will now move on to understanding the challenges of training such a neural network using gradient descent and propose variants and adaptations that could be useful in improving the effectiveness of training neural networks using gradient descent.

(Refer Slide Time: 0:55)



Review: Gradient Descent (GD)

- Optimization algorithm used to find minima of a given differentiable function
- At each step, parameters (θ) are pushed in negative direction of gradient of a cost function ($J(\theta\ x, y)$, in figure alongside) w.r.t parameters

$$\theta_{new} = \theta_{old} - \alpha \Delta \theta_{old}$$

where α is learning rate

Credit: Albert Lai

Before minimum
After minimum
derivative
$(\theta_0, J(\theta_0))$
global minimum
Since the derivative is negative, if we subtract the derivative from θ_0, it will increase and go closer the minimum.
Since the derivative is positive, if we subtract the derivative from θ_0, it will decrease and go closer the minimum.
Vineeth N B (IIT-H)
§4.3 Gradient Descent
NPTEL

We will start with a brief review of gradient descent, as we already said gradient descent is an optimization algorithm that is used to find the minima of any differentiable function. Remember again that the loss function that we use to train a neural network, means square error is what we use so far, but even if we use any other loss function that function has to be differentiable.

Please do not worry if you are not aware of what other loss functions to use, we will see plenty of them as we go through the rest of this course. And when we use gradient descent at each step parameters are pushed in the negative direction of the gradient of a cost function or error function or loss function whatever we choose to call it.

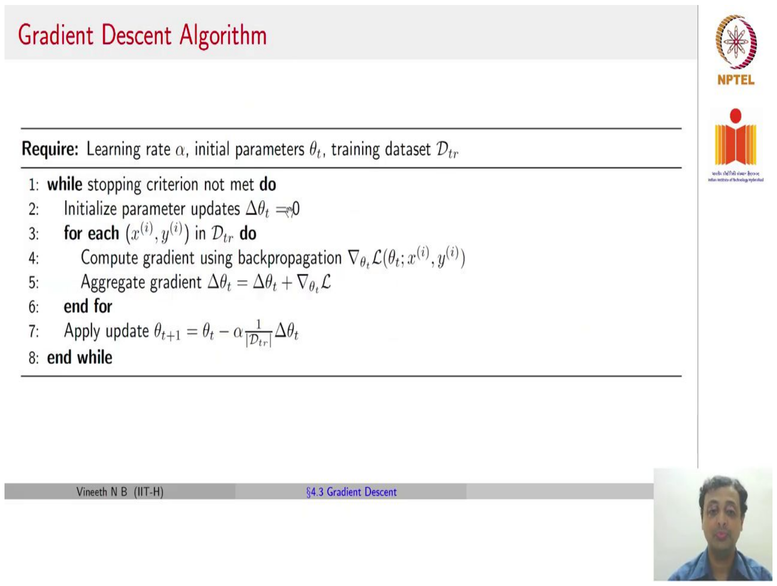
And the parameter update rule is given by

$\theta_{new} = \theta_{old} - \alpha\Delta\theta_{old}$. α is learning rate of step size times the change in the parameters. And we saw this visualization also in the last lecture where if you have a simple convex function, you will have, and if you consider a point to the left of the minimum, see this blue point here, the gradient there is negative so when you go in the negative direction of that you are going to be going towards the positive side or the right side which will take you towards the minimum.

If you have started at this point here on the right side the gradient there is positive so the negative gradient will go to the opposite direction which will once again be towards your minimum and this is an a visual illustration of how gradient descent can be used to minimize any objective function. Keep in mind here that this example of a function, this black curve here is a convex function.

So, you are going to have an exercise at the end of this lecture to know what convex functions are and we will discuss them a little bit later if you are not aware. But for the moment let us go ahead and try to understand how we can improve gradient descent.

(Refer Slide Time: 3:20)



Gradient Descent Algorithm

Require: Learning rate α, initial parameters θ_t, training dataset $\mathcal{D}_{tr}$

```
1: while stopping criterion not met do
2:     Initialize parameter updates $\Delta\theta_t = 0$
3:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
4:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
5:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
6:     end for
7:     Apply update $\theta_{t+1} = \theta_t - \alpha\frac{1}{|\mathcal{D}_{tr}|}\Delta\theta_t$
8: end while
```

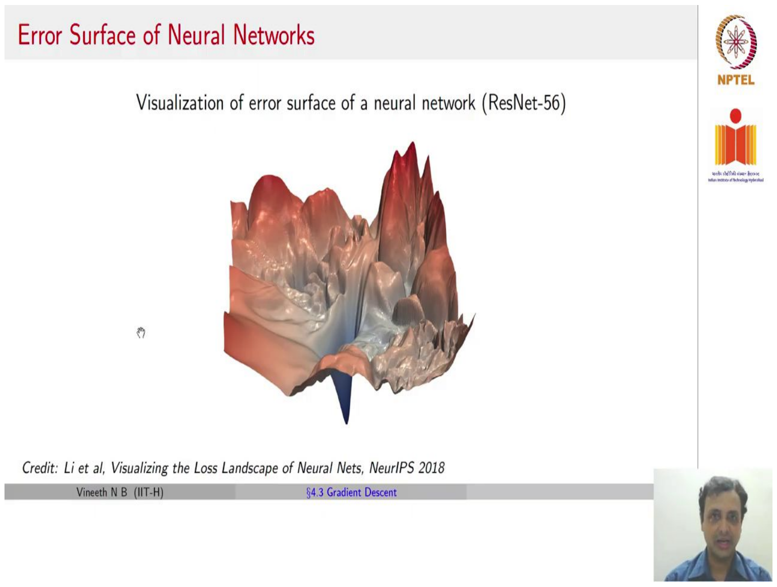
Vineeth N B (IIT-H) §4.3 Gradient Descent

Here is the gradient descent algorithm, you are given a learning rate, you have some initial parameters θ_t and a training data set D_{tr}. As long as the stopping criterion is not met, you initialize your parameter updates and for each point in your training dataset you compute your gradient using back propagation of the loss function with respect to each of your parameters. And you aggregate your gradients in your $\Delta\theta$ variable.

Once again just to remind you $\Delta\theta$ denotes the change in variables, ∇L denotes the gradient, please note the difference in notation since they can look similar at first group. Then finally you have to apply your update $\theta_{t+1} = \theta_t - \alpha\frac{1}{|D_{tr}|}\Delta\theta_t$

How long do you train this? Until your gradient becomes zero. Each of these for loops here is typically called one epoch, one epoch corresponds to one full iteration over your training dataset, so we compute the gradients across an epoch, average all your gradients and then update your parameters, that is the simple summary of gradient descent.

(Refer Slide Time: 5:01)



Error Surface of Neural Networks

Visualization of error surface of a neural network (ResNet-56)

Credit: *Li et al, Visualizing the Loss Landscape of Neural Nets, NeurIPS 2018*

Vineeth N B (IIT-H) §4.3 Gradient Descent

Now, let us move on to understanding what is known as the error surface of neural networks. The error surface of neural networks is simply a plot of all your parameters of the neural network versus the corresponding cost value or the loss function value that you would have. From now on when we say weights we are going to assume that it also encompasses biases in them, so just please assume it that way.

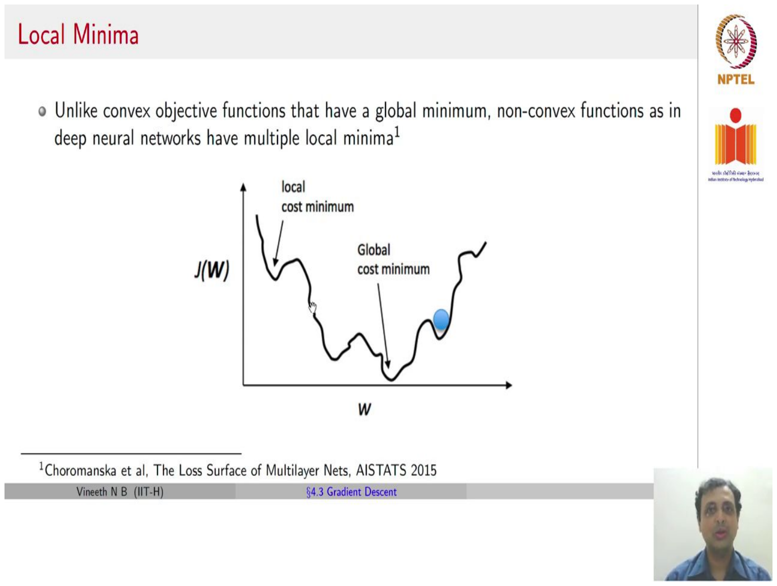
So, if you see this undulated surface in this slide, each point there is one particular weight configuration, which means one value assigned to each weight and the corresponding loss that was incurred when you propagate all your training data points through that weight configuration. Remember for every training data point you forward propagate it, you get an output, you would get a loss.

Next training data point, forward propagated, an output and a loss, now you average all of these losses that is going to be your overall loss for that weight configuration before you update further. This is an error surface which can be very complex, modern neural networks that are used in practice especially in computer vision have millions of parameters, a very popular network known as AlexNet which once again we will see later has close to 60 million parameters.

This one is ResNet which again has millions of parameters, which means although we see it in three dimensional space here, this kind of an error surface actually exists in a 60 million dimensional space. So, you can imagine the complexity that is going to be there on that particular surface.

And the goal when you train a neural network is to start somewhere on the surface which means you could be starting at any one particular point, remember that is the random initialization of the base and your gradient descent has to take you to the minimum. Clearly, you can see here that this is a very non-convex surface, it is not a single bowl like shape, it is a non-convex surface with lots of undulations, so it is not a trivial task to be able to traverse this surface.

(Refer Slide Time: 7:32)



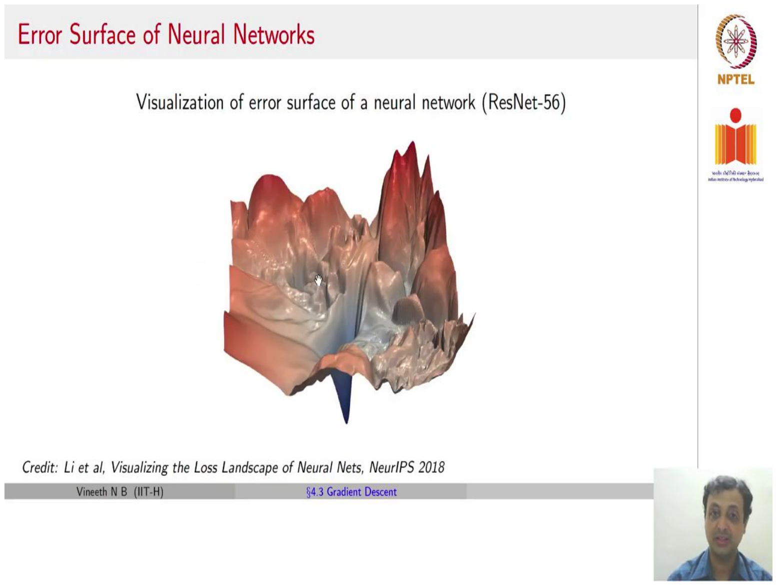
Local Minima

- Unlike convex objective functions that have a global minimum, non-convex functions as in deep neural networks have multiple local minima[1]

local
cost minimum
Global
cost minimum
J(W)
W

[1]Choromanska et al, The Loss Surface of Multilayer Nets, AISTATS 2015

Vineeth N B (IIT-H) §4.3 Gradient Descent



Error Surface of Neural Networks

Visualization of error surface of a neural network (ResNet-56)

Credit: Li et al, Visualizing the Loss Landscape of Neural Nets, NeurIPS 2018

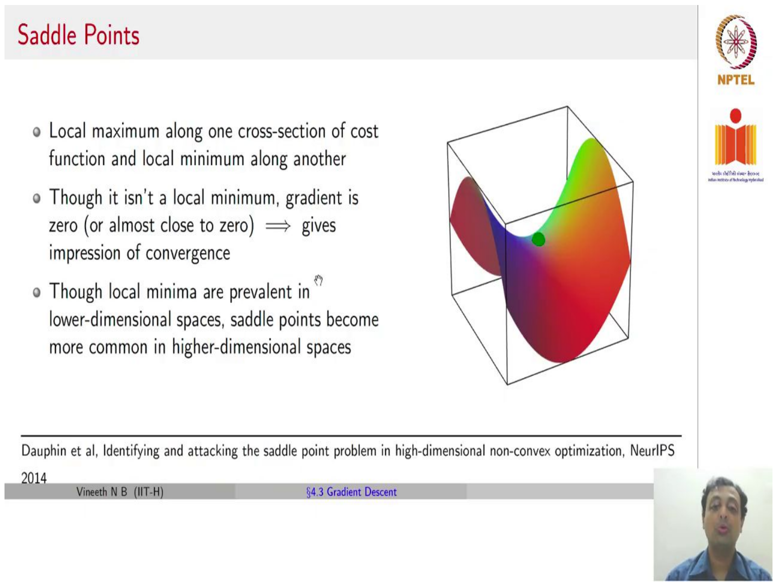
Vineeth N B (IIT-H) §4.3 Gradient Descent

One of the major problems when you have a non-convex objective function is that you are going to have many local minimum, which means the solution that you finally converge to after applying gradient descent is going to depend on where you start, depending on where you start that is going to decide which local minimum you are going to converge to. Why so?

Because when you hit one of this local minima, your gradient is going to be zero and your gradient descent algorithm will terminate, which means it is going to be extremely important to know where to start and unfortunately when we start training a neural network we do not understand the complexity of the error surface involved.

While we saw one such visualization in the previous slide, it is not possible to understand where your initialization could be or where the actual local minima could be and so on and so forth in a very high dimensional space. While there have been such efforts to visualize these inner surfaces, it is not good enough for us to use that for visualizing and training directly. Which means training to find a suitable local minima is still an important challenge here.

(Refer Slide Time: 8:55)



Saddle Points

- Local maximum along one cross-section of cost function and local minimum along another
- Though it isn't a local minimum, gradient is zero (or almost close to zero) $\Longrightarrow$ gives impression of convergence
- Though local minima are prevalent in lower-dimensional spaces, saddle points become more common in higher-dimensional spaces

Dauphin et al, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, NeurIPS 2014

Vineeth N B (IIT-H) §4.3 Gradient Descent

Similarly, you could also have what are known as saddle points, saddle points are as you all know from high school calculus are critical points again which means the gradient is 0, but it is minima along a set of dimensions and maxima along another set of dimensions. For us dimensions in this context are weights, we are talking about the dimension of that error surface and remember the error surface is a plot between weights and the cost function.

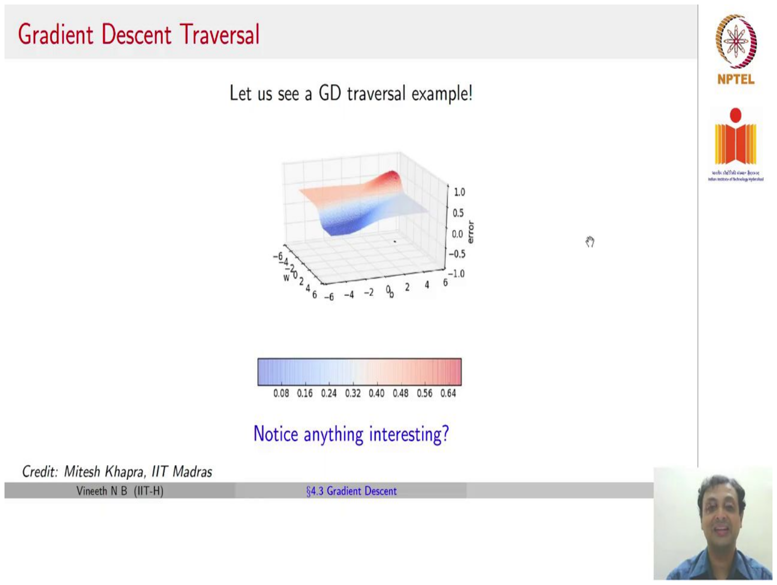
So, here is an example of a simple saddle function or a saddle point, so you have here. This is a point which is a minima along this dimension and a maxima along the other dimension. Why does this matter to us? Once again because the gradient is zero at a saddle point, if you, your gradient descent converges to saddle point you are going to end your training right there and a saddle point may not be the ideal solution because it is a maximum along certain dimensions and you may want to reach a better minima which can give you a better neural network solution that you can use in practice.

Another important observation which was made in this particular work called identifying and attacking the saddle point problem in high dimensional non-convex optimization was that as you go to higher and higher dimensional spaces which means as your network becomes more and more complex, more and more weights then your local minima which would be more prevalent in low dimensional spaces keep getting replaced by saddle points as the dimensionality increases.

Why so? A simple intuition for that is if your dimension of your error surface, so the dimension of your weights is going to be say 1 million, let us say you found a critical point maybe that critical point is a minimum for nine hundred and ninety-nine thousand nine hundred and ninety nine dimensions or weights, but there could be still that one dimension where it ends up becoming a maximum.

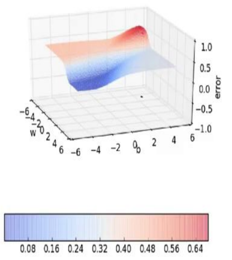
And when you have a very large dimensional space it is more probable that there are some points, there are some dimensions where that particular weight configuration could be a maximum, that is a simple intuition for why saddle points proliferate as you go to higher and higher dimensions.

(Refer Slide Time: 11:31)



Gradient Descent Traversal

Let us see a GD traversal example!



Notice anything interesting?

Credit: Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) §4.3 Gradient Descent

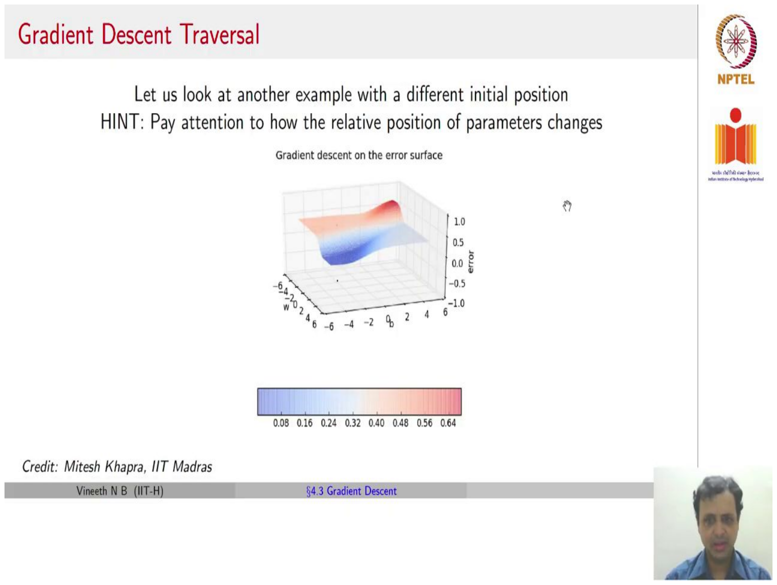
Let us see a simple gradient descent traversal example to understand an issue before we go there just to explain what is happening here, this is a plot of the error surface, as you can see, red is a high value, blue is a low value, so what this plot is showing you is it is taking one value of w, one value of b, it is a very simple neural network and this z-axis here plots the error, so you can see here that there are certain points where the error is high and certain points where the error is low and it is the surface looks like a carpet.

So, now let us, just before we, so we understand this which is the local minima here, where would you want your neural network to converge? At the deepest blue color because that is where your error is leased. So, let us now initialize from a random point on this error surface and see what gradient descent does, watch carefully.

You see now, on the top surface you see red plots, on the bottom surface you see black spots going, watch now that the updates are slow and then after some time the updates pick up speed and finally you converge to that blue minima that you were looking for. Did you notice something interesting?

Yes, there were some points in the traversal of gradient descent when the traversal became slow and there were certain points when traversal became fast. Let us keep this in mind when we go forward with the discussion.

(Refer Slide Time: 13:22)



Gradient Descent Traversal

Let us look at another example with a different initial position

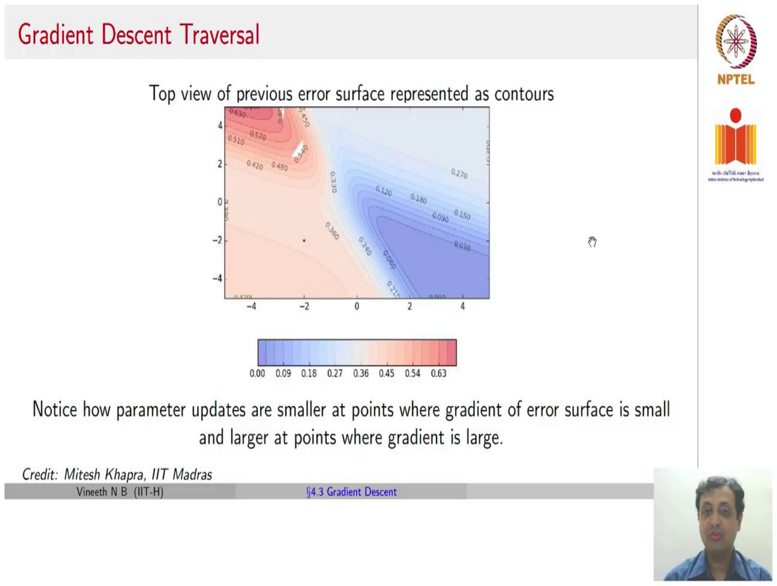
HINT: Pay attention to how the relative position of parameters changes

Credit: Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) §4.3 Gradient Descent

Let us look at the same error surface again, but let us initialize at a different point and now see what happens, we are going to initialize somewhere in the top left here, that is what you want to be looking for when you see this visualization. You see these red lines, the black lines, gradient descent is traversing initially a bit slow, it travels a bit fast, again a bit slow, again a bit slow and then finally converges. So, you can see that there are different points at which the speed of gradient descent can keep changing.

(Refer Slide Time: 14:07)


Gradient Descent Traversal

Top view of previous error surface represented as contours

Notice how parameter updates are smaller at points where gradient of error surface is small and larger at points where gradient is large.

Credit: Mitesh Khapra, IIT Madras

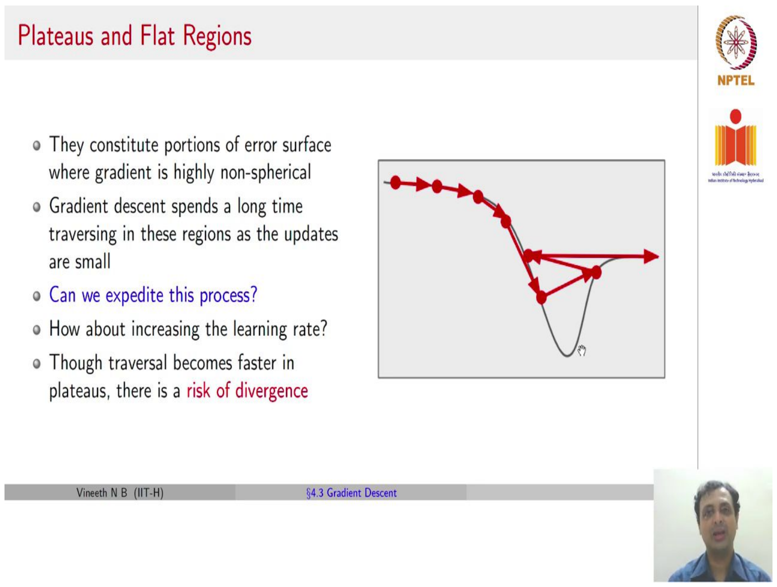
Vineeth N B (IIT-H) §4.3 Gradient Descent

Let us see this now from a slightly different perspective we are going to see this from the viewpoint of what is known as a contour plot. A contour plot is simply a two dimensional view of any surface, so if you take any surface let us imagine the Himalayas in front of your head, let us imagine a mountain range in front of your head, imagine that you are slicing the mountain range with a huge knife and now you are going to view from the top all of these slices laid down on a single table and that view is what you see as a contour plot.

What do you mean by a contour plot? So, if you take any of these contours what you see on the screen here, if you take one of these lines like let us follow this particular line here. All it means is that for all the points on that line the error is the same, these are also sometimes called iso contours, which means all the error values at those points are the same. Why? Because you took a cross section, a cross section simply means that the error value was the same at that particular point.

Now, let us see the same example that we saw on the earlier slide as a contour point. If you see the contour plot now you would notice that parameter updates which are given here, you can see the parameter update values are smaller at points where gradient of error surface is small. Remember we said that these are points which have a lesser gradient the blue areas, and the red areas have a higher error value. And whenever the gradient of the error surface is small the parameter updates are small and whenever the gradient is large these are points where the gradient is large the parameter updates are also large on those particular points.

(Refer Slide Time: 16:15)



Plateaus and Flat Regions

- They constitute portions of error surface where gradient is highly non-spherical
- Gradient descent spends a long time traversing in these regions as the updates are small
- Can we expedite this process?
- How about increasing the learning rate?
- Though traversal becomes faster in plateaus, there is a risk of divergence

Vineeth N B (IIT-H) §4.3 Gradient Descent

That brings us to this another facet of understanding how to train neural networks in terms of plateaus and flat regions. Plateaus and flat regions constitute portions of the error surface where the gradient is highly non-spherical. What does this mean? When we say non-spherical it means that each dimension of the gradient is of different, different values.

Remember that the entire gradient is a vector; it is the derivative of the loss function with respect to every weight in your neural network, you unroll all of your weights in the neural network across all your layers into a single vector. Now, you take the loss with respect to each of those weights and put them in the same vector, you would use chain rule and back propagation to compute those gradients but once you compute them you input all of them in a single vector.

Now, if all of those gradients have similar values at a particular weight configuration you would then say that that area of the error surface is spherical because it all has equivalent values in all dimensions. But when you go to a plateau or a flat region you are going to have very largely elliptical shapes where in one dimension things move very slowly, whereas in another dimension things could move rapidly, that is what we mean by non-spherical in this particular context. Non-spherical is elongated elliptical, spherical as a sphere and elongated elliptical.

So, gradient descent spends a long time traversing in these kinds of regions as the updates could be very, very small. Remember when you have plateaus and flat regions, it is like going through an area where the gradient very gradually tapers and when the gradient gradually tapers the gradient values are small, the parameter updates are going to be small as straightforward is that.

So, one question we could ask is when you are going through a plateau, cannot you just walk faster, cannot you just expedite this process? We can but there are some tradeoffs let us see what we can do there. One simple option is to take longer steps, you know your gradient just take longer steps. What does it mean to take longer steps in gradient descent?

Increase the learning rate or the step size alpha that we talked about. This is a good idea in general when you go through a plateau, but if you simply just increase the learning rate alpha to a very high value for your entire gradient descent traversable that may not be of use, let us see why.

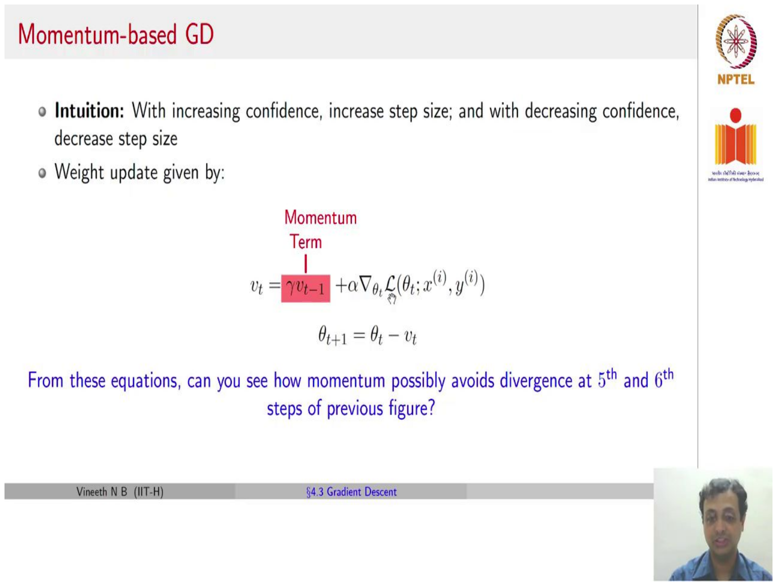
Here is an example that you see on the right. So, you see a certain error surface, this is a very simple error surface, where on the x-axis is a weight, on the y-axis is an error, a very simple one-dimensional, one parameter situation. So, you have these updates that go along the direction of the negative gradient that happens and at a particular point the gradient changes rapidly and you go to the next point.

At a certain point on your error surface you see here that the curve started jumping around. Why is that so? Because this happens when your learning rate is very, very high, let us try to understand this carefully. So, if you are at a particular point here where the mouse cursor is currently showing you.

Let us say the gradient is along a particular direction and if you choose to take a long step, your alpha was high on your x which is where you are making the parameter updates you may jump by a large amount and go somewhere here. And when you go there the corresponding error there is this particular value.

So, while you wanted to go towards this minimum, you took a large step and went to this value along the w axis and at that value along the w axis the error turns out to be high. So, the learning rate is too high you actually go to a point of higher error and once again from there if learning rate is high an even higher error and you diverge out of that local minimum rather than converge to that local minimum, not maybe this is the right local minimum you would have wanted to converge to, in that case keeping the learning rate very high is really not going to help you.

(Refer Slide Time: 21:02)



Momentum-based GD

- **Intuition:** With increasing confidence, increase step size; and with decreasing confidence, decrease step size
- Weight update given by:

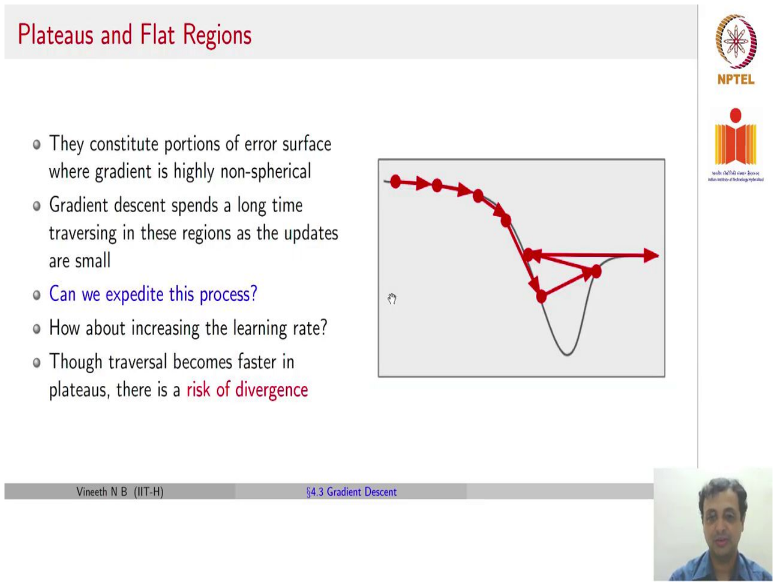
Momentum Term

$$v_t = \underbrace{\gamma v_{t-1}}_{\text{Momentum Term}} + \alpha \nabla_{\theta_t} \mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$$

$$\theta_{t+1} = \theta_t - v_t$$

From these equations, can you see how momentum possibly avoids divergence at 5th and 6th steps of previous figure?

Vineeth N B (IIT-H) §4.3 Gradient Descent



Plateaus and Flat Regions

- They constitute portions of error surface where gradient is highly non-spherical
- Gradient descent spends a long time traversing in these regions as the updates are small
- Can we expedite this process?
- How about increasing the learning rate?
- Though traversal becomes faster in plateaus, there is a risk of divergence

Vineeth N B (IIT-H) §4.3 Gradient Descent

That leads us to a method known as momentum based gradient descent. Momentum based gradient descent relies on the intuition that with increasing confidence increase the step size and with decreasing confidence decrease the step size. Simple intuition is, why not we make use of the direction that we have been traversing on so far to make our next update? Why should we rely only on the gradient at the current time step?

A good way to imagine all of this is as a blind person traversing Himalayas. So, Himalayas is a mountain range and you have to ideally, a helicopter takes you and drops you at a particular point on the Himalayas and your job is to navigate and reach the bottom of the Himalayas where the error value is the least.

Why did I say a blind person? Because if you had vision you could see far and directly go to where the minimum is, unfortunately gradient descent does not have that privilege. What does gradient descent do? Has a stick in its hand, keeps tapping everywhere around it and sees where the negative slope is the highest and goes in that direction at one step. This is the analogy that we are following now to traverse the error surface of the neural network.

Why is this important, why is this analogy important? Because now, if I were traversing the error surface of this neural network I would simply ask the question, let us say I was at a particular point and I have been coming down in a particular direction all the while and at one point the gradient at that point as I tuck my stick around tells me to go in a very different direction. The question I would ask is should I completely change direction now, I have been coming along one way so far, should I completely change direction now?

Maybe the answer is let us not do that, let us keep in mind that we were coming along a certain direction, use that to some extent and the new gradient direction to some extent and combine them in some way. This is exactly what momentum based gradient descent does. So, mathematically speaking, how do you implement it?

You say, you define something known as a velocity vector v_t, velocity is simply change of distance over time, so it is simply your update in parameters over one iteration, it is simply a name in this particular context. v_t which is the change of parameters that you are going to have in this time step is given by γ v_{t-1}. What was your change in weights in the previous time step into weighted by a particular coefficient γ plus the rest of it is standard gradient descent, alpha learning rate times the gradient of the loss with respect to the parameter θ.

What is this telling us? It is telling us that if you were coming along a particular direction and your gradient tells you to go to a different direction, combine them in some way and then get a velocity vector v_t which is what you use to update your parameters. Then you have $\theta_{t+1} = \theta_t - v_t$, which tells you the weight update.

So, keep in mind here that when v_{t-1} and the gradient are in the same direction, this will only give us more momentum in the same direction and that explains the reason for this name of the term if your v_t-1 and l are in the same direction you are going to use your past momentum to walk even faster in that direction. And this should give you an idea of why we are discussing this in the context of plateaus and flat surfaces. If we were going along the same direction for a while let that give us momentum to take longer steps in that direction and go faster to the minimum.

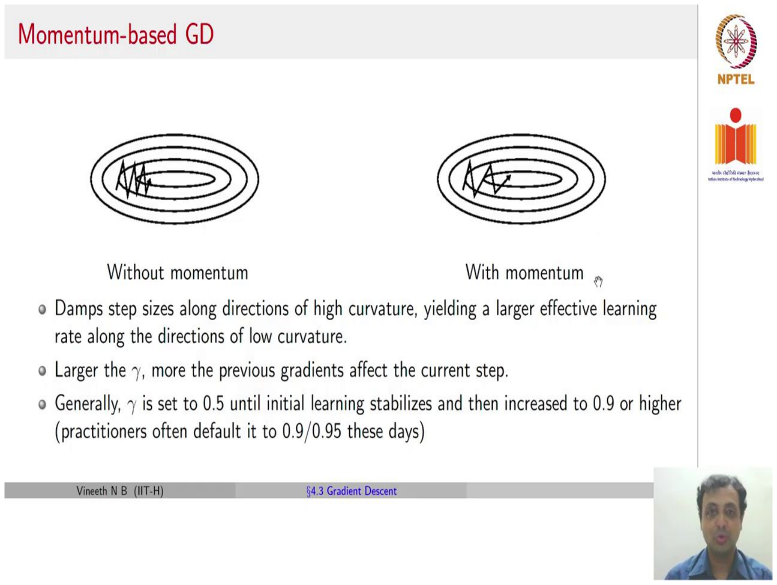
Now, you can probably see how momentum can avoid divergence in the previous figure, let us go back to the previous figure here, so can momentum avoid divergence here let us see. In this particular case you would have got the gradient to be something like this, the previous gradient was something like that and the combination of the gradients would have taken you in a direction which was directly close to the minimum.

And one important thing to keep in mind here is both of these while learning rate is typically chosen based on the designs what the user wants to give a value for, γ is typically a value between 0 and 1. Why is that so, why cannot γ be greater than 1? There is an important reason here, firstly you do not want that to out shadow the gradient that is a simple reason, but the more important reason is remember v_{t-1} has a component of v_{t-2} in it, because momentum was used in computing v_{t-1} in the previous time step.

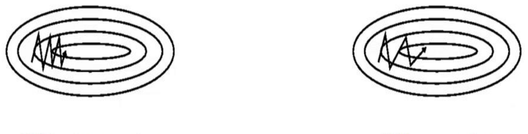
So, now this would become γ so the component of v_{t-2} that influences v_t will be γ^2 into v_{t-2} and by ensuring that γ lies between 0 and 1, γ^2 will be smaller than γ which means how much v_{t-2} contributes to v_t will be lesser than how much v_{t-1} contributes to v_t, that is one reason to choose γ between 0 and 1.

And that should again tell you how combining these should now give you a smaller time step, you are not giving, your alpha is not a very high learning rate but it now keeps you within check. So, now you can also see that you are not going to randomly diverge. So, if you are at this point and the gradient told you to go in this direction but your previous gradient had told you to go to a particular direction, you now combine these two and you may actually come back to the minima from the next data point, from the next weight configuration, this could help you avoid divergence even in these scenarios.

(Refer Slide Time: 27:41)



Momentum-based GD



- Damps step sizes along directions of high curvature, yielding a larger effective learning rate along the directions of low curvature.
- Larger the γ, more the previous gradients affect the current step.
- Generally, γ is set to 0.5 until initial learning stabilizes and then increased to 0.9 or higher (practitioners often default it to 0.9/0.95 these days)

Vineeth N B (IIT-H) §4.3 Gradient Descent

So, here is a contour plot visualization, a very simple contour plot visualization for momentum. So, once again these are contour plots, So you could imagine this to be an elliptical mountain cross section and you are seeing it from the top. And you can see here that when you do not use momentum, you start from a particular point on this error surface, this is the point on the other surface that you start.

When you see things as contour plots the gradient would always be normal to the contour plots. So, when you see it as the entire surface you would be going up but when you see it as a contour plot the gradient is going to be normal to the contour plot. So, at a particular point your gradient will take you this way and then you go there and the gradient takes you the other way, you go this then the gradient will take you again the other way and so on and so forth.

You still are going towards your minimum which is in the center of this contour plot, but you are going to be oscillating to reach that minimum that you are going for. Keep in mind here that if this error surface or this contour plot was spherical, currently we are looking at it as elliptical which means certain dimensions gradient is higher, certain dimensions gradient is lower.

But if this error surface was spherical then your gradient would be normal and it would straight point to the center or the minimum and you would probably reach in one step. So, imagine an spherical error surface you can try to draw it and see that the normal at any contour point will point straight to the center and that will take you provided you choose suitable step size you will a learning rate you would straight go to the center.

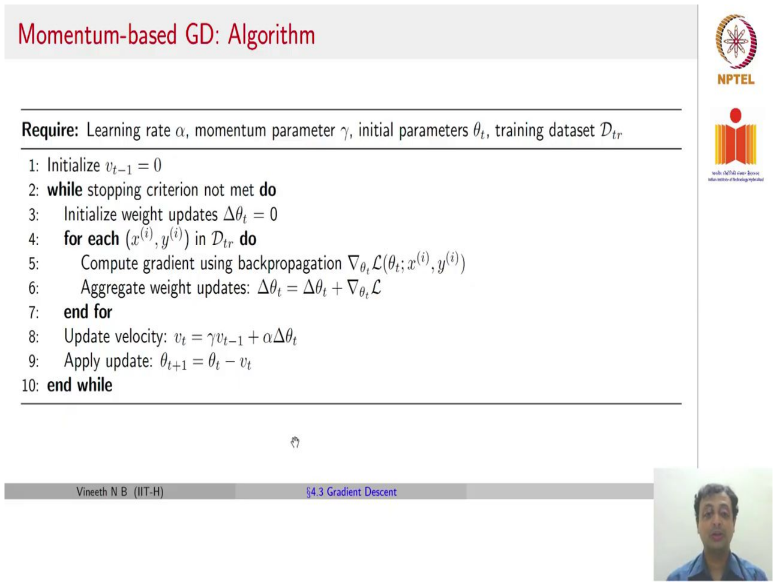
But you will see here that you probably may not find the need for momentum when your error surface is spherical, momentum is more useful when your error surface is non-spherical which is once again more likely to happen in a complex error surface. When you have momentum you see that the oscillation seems to be reducing, let us try to understand why.

You first go for one step to the next step, the first gradient. When the next gradient comes you will combine it with some portion of the previous gradient so that takes you here, now let us see what happens at this particular point your gradient is going to point you in a particular direction but your previous gradient was pointing you in this direction, you take a sum of these two gradients and that gradient will take you in this direction.

So, the sum of those two gradients will take you in this direction and this way your oscillations reduce, you still oscillate but your number of oscillations reduce and you can quickly get to your minimum. More simply speaking momentum dams the step sizes along directions of high curvature where you get an effective larger learning rate on the directions of low curvature, rather when your error surface is very steep go slow, when your error surface is flat go fast.

Larger the γ more the previous gradients affect the current step. A general practice is to start γ with 0.5 until your initial learning stabilizes. Why? Because you may not be able to trust your gradients, initially your gradients could be going in different directions, it is remember again the blind man on Himalayas, if you keep traversing that for some time then your gradient probably, you understand what direction is the right direction to go, it stabilizes after some time then you can increase your momentum parameter γ to 0.9 or 0.95, wait for your train to stabilize a bit and then increase momentum because you can then trust the previous directions you are walking on or traversing by. But generally in practice people often just set it to 0.9 or 0.95 from the very beginning itself.

(Refer Slide Time: 31:49)



Momentum-based GD: Algorithm

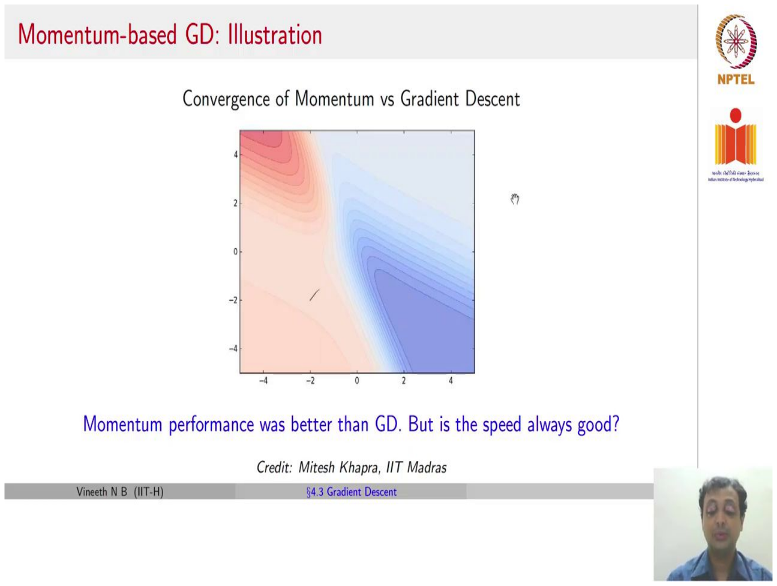
```
Require: Learning rate $\alpha$, momentum parameter $\gamma$, initial parameters $\theta_t$, training dataset $\mathcal{D}_{tr}$
1: Initialize $v_{t-1} = 0$
2: while stopping criterion not met do
3:     Initialize weight updates $\Delta\theta_t = 0$
4:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
5:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
6:         Aggregate weight updates: $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
7:     end for
8:     Update velocity: $v_t = \gamma v_{t-1} + \alpha\Delta\theta_t$
9:     Apply update: $\theta_{t+1} = \theta_t - v_t$
10: end while
```

Vineeth N B (IIT-H) §4.3 Gradient Descent

Here is the algorithmic version of a momentum based gradient descent, you have a learning rate alpha, a momentum parameter γ, initial parameters, training data set, pretty much the same algorithm as gradient descent, the only change now is lines 8 and 9 where you compute your velocity which was γ times v_{t-1} plus alpha times $\Delta\theta_t$, which is your update based on your gradient and then you complete your parameters.

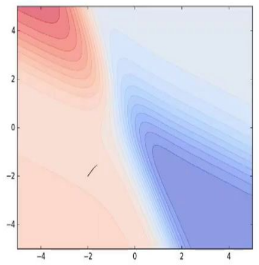
One small change here is we remove the 1 by t training cardinality of the retraining set here just for convenience, you can assume that when we aggregate the weight updates you are subsuming that there and taking the average across the training data points. So, for convenience you are going to avoid that.

(Refer Slide Time: 32:42)



Momentum-based GD: Illustration

Convergence of Momentum vs Gradient Descent



Momentum performance was better than GD. But is the speed always good?

Credit: Mitesh Khapra, IIT Madras

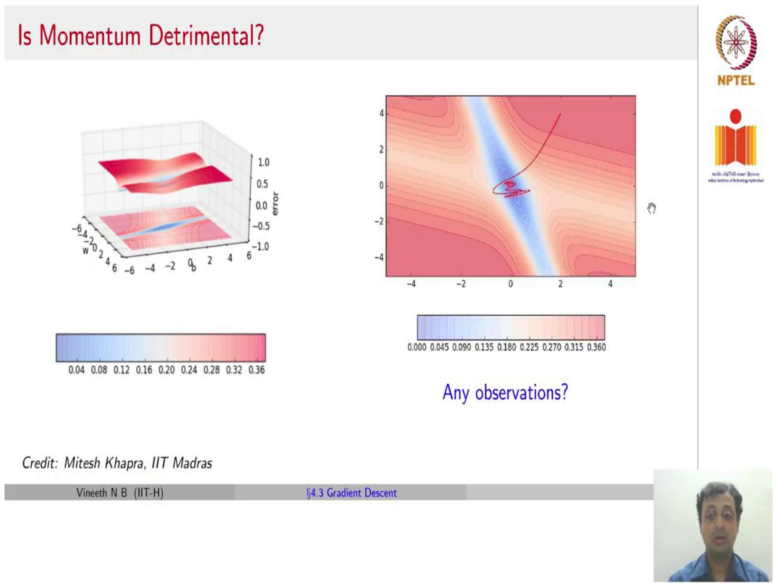
Vineeth N B (IIT-H) §4.3 Gradient Descent

Here is a visualization of convergence of momentum, this is gradient descent once again this is a contour plot you can see here that red values are high values, blue values are low values, so you know that you want to converge to somewhere in this bottom right region where error values are low.

So, this is your gradient descent curve until a certain point, a certain number of iterations. Now, let us try to see how momentum does on the same surface. You can see the red curve now, you can see the steps that it is taking are long steps, it seems to overshoot the minima, take a u-turn, come back and seems to get close to the minimum.

So, what is happening this is what we saw on the previous slide that when the error surface is elliptical the gradient makes you oscillate a little bit around the minimum before converging to the minimum itself. So, clearly the momentum performance was better than GD, in fact we will mention later in terms of what step and what error was there in this particular example. But it seems to make a sense that momentum this seems to go faster than GD in this scenario, but is it always good is a question that we would like to ask.

(Refer Slide Time: 34:05)


Is Momentum Detrimental?

Any observations?

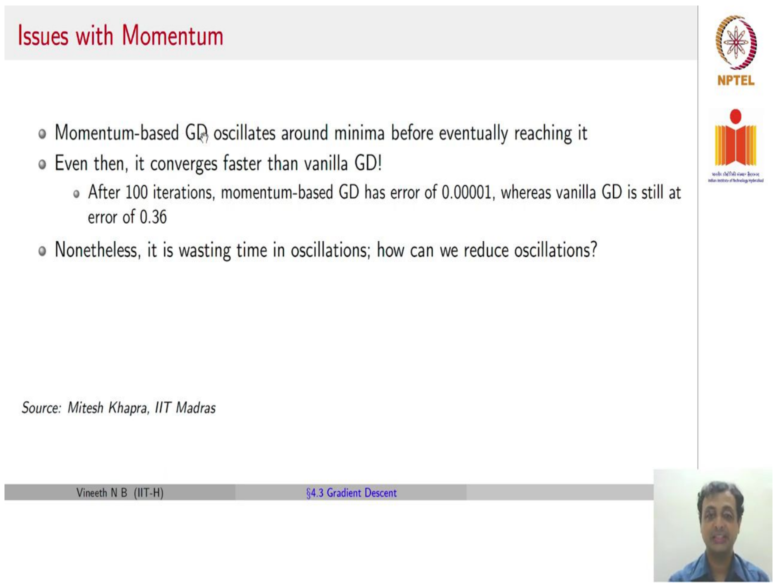
Credit: Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) §4.3 Gradient Descent

To see that let us take this particular example, here is the error surface, you can see here that this is something like a carpet with a hollow somewhere in between. So, you see here that blue corresponds to the minimum, red is again a high value, this is your error surface and the corresponding contour plot is shown on the right, so this is the contour plot you are seeing this from top, where you want to get to this center point here which is the minimum of that error surface.

Now, you can see gradient descent traversals for a few iterations here, in the same number of iterations the gradient descent traveled this distance you can see the mouse cursor, let us try to see what momentum does. You can see the red line again, it seems to be shooting forward and it reaches the region of the minima, then keeps oscillating and finally gets to the minimum. Let us try to summarize that clearly.

(Refer Slide Time: 35:07)



Issues with Momentum

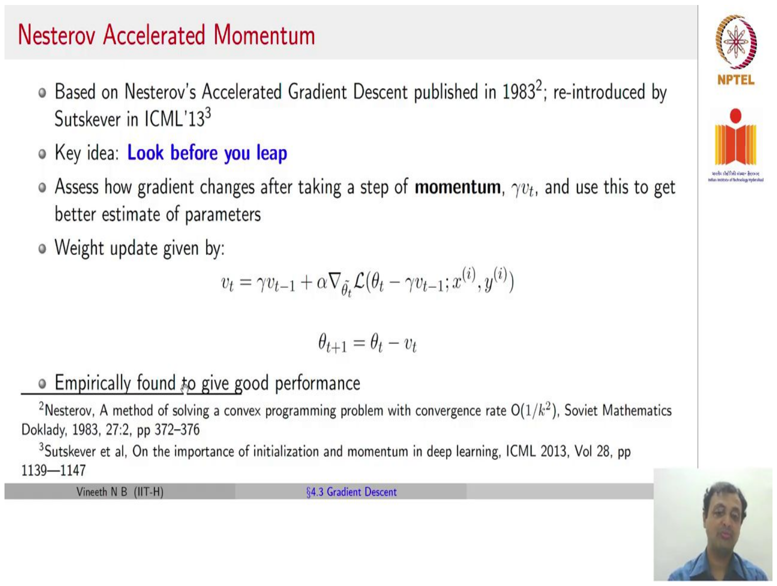
- Momentum-based GD oscillates around minima before eventually reaching it
- Even then, it converges faster than vanilla GD!
  - After 100 iterations, momentum-based GD has error of 0.00001, whereas vanilla GD is still at error of 0.36
- Nonetheless, it is wasting time in oscillations; how can we reduce oscillations?

Source: Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) §4.3 Gradient Descent

Momentum based gradient descent oscillates around your minimum before eventually reaching it, even then it converges much faster than vanilla GD. So, just as I said after 100 iterations which was the number of iterations that was shown for gradient descent of the same figure, momentum based gradient descent has an error of 10 power minus 5, whereas vanilla gradient descent is still at an error of 0.36. Nonetheless, we see that momentum seems to be wasting some time in oscillating around that area, let us ask ourselves if we can reduce that oscillation time in some way.

(Refer Slide Time: 35:50)


Nesterov Accelerated Momentum

- Based on Nesterov's Accelerated Gradient Descent published in 1983[2]; re-introduced by Sutskever in ICML'13[3]
- Key idea: **Look before you leap**
- Assess how gradient changes after taking a step of **momentum**, γv_t, and use this to get better estimate of parameters
- Weight update given by:

$$v_t = \gamma v_{t-1} + \alpha \nabla_{\tilde{\theta}_t} \mathcal{L}(\theta_t - \gamma v_{t-1}; x^{(i)}, y^{(i)})$$

$$\theta_{t+1} = \theta_t - v_t$$

- Empirically found to give good performance

[2]Nesterov, A method of solving a convex programming problem with convergence rate $O(1/k^2)$, Soviet Mathematics Doklady, 1983, 27:2, pp 372–376

[3]Sutskever et al, On the importance of initialization and momentum in deep learning, ICML 2013, Vol 28, pp 1139—1147

Vineeth N B (IIT-H) §4.3 Gradient Descent

That leads us to another method known as Nesterov Accelerated momentum. This is based on Yurii Nesterov's work, Yurii Nesterov is a huge researcher who has created a significant impact in the field of optimization. This work called accelerated gradient descent of his was published in 1983 and was recently reintroduced in the Machine Learning context in 2013.

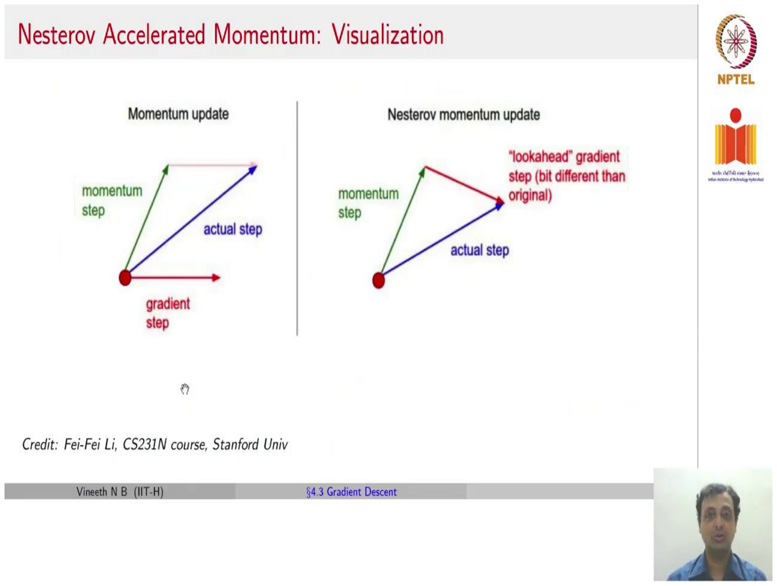
And the key idea of Nesterov accelerated momentum is to develop the idea of momentum to include one key thought which is look before Yurii rather I am going in a particular direction on the Himalayas again, traversing the Himalayas trying to find out that minimum point of the valley.

And I do get some sense that my current gradient is telling me to go in a different direction. Can I go one step further along the way I was coming, see the gradient there and then decide rather than use the gradient at the current time step? So, you want to compute a look ahead gradient or and then use that knowledge in taking your current step. Let us see what that means when we write out the equations.

So, we are saying now that this equation is very similar to what you saw for the momentum equation $v_t = \gamma v_{t-1} + \alpha \nabla_{\tilde{\theta}_t} L(\theta_t - \gamma v_{t-1}; x^{(i)}, y^{(i)})$, what is the difference here? If you observe carefully the main difference between this equation and the equation that we had a couple of slides back for momentum is this term here, this term for momentum was simply θ_t but this term for us now is $\theta_t - \gamma v_{t-1}$, what does this mean?

This means that we take θ_t which is the current parameters, v_{t-1} was my previous parameter update, I will have one more update of that with γ with my momentum parameter, go to that step, compute my loss on that weight configuration and then use that gradient to make my move in this step. This idea is empirically found to give good performance.

(Refer Slide Time: 38:23)



Nesterov Accelerated Momentum: Visualization

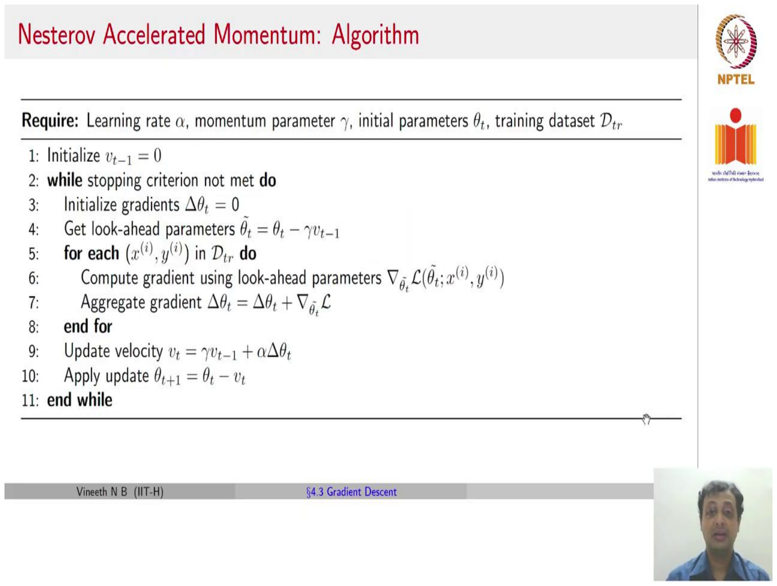
Credit: Fei-Fei Li, CS231N course, Stanford Univ

Vineeth N B (IIT-H) §4.3 Gradient Descent

Here is a visualization of how momentum, Nesterov momentum differ. So, you can see here in the momentum update that momentum based gradient descent is a combination of two vectors, the momentum step which is the direction in which you have been traversing and the gradient step which is what the current gradient is telling you to do, and your actual step is an weighted addition of these two vectors which is perhaps going to tell you the step that you have to take now.

In Nesterov momentum update you first follow your momentum step and then at that step you find what is the gradient and use that gradient to be able to take your actual step, so you do a momentum plus the look ahead gradient which will probably give you this vector. So, there is a slight difference in where you would reach after one step using momentum and Nesterov accelerated momentum.

(Refer Slide Time: 39:20)



Nesterov Accelerated Momentum: Algorithm

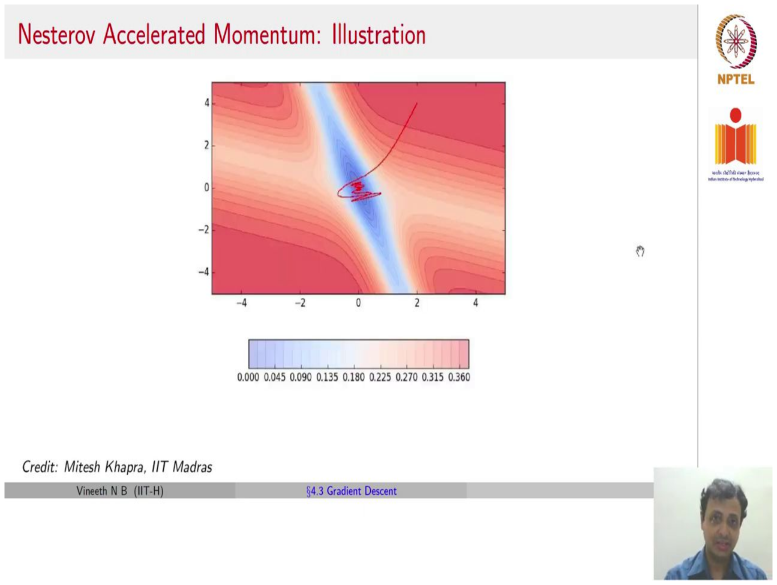
```
Require: Learning rate $\alpha$, momentum parameter $\gamma$, initial parameters $\theta_t$, training dataset $\mathcal{D}_{tr}$
1: Initialize $v_{t-1} = 0$
2: while stopping criterion not met do
3:     Initialize gradients $\Delta\theta_t = 0$
4:     Get look-ahead parameters $\tilde{\theta}_t = \theta_t - \gamma v_{t-1}$
5:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
6:         Compute gradient using look-ahead parameters $\nabla_{\tilde{\theta}_t}\mathcal{L}(\tilde{\theta}_t; x^{(i)}, y^{(i)})$
7:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\tilde{\theta}_t}\mathcal{L}$
8:     end for
9:     Update velocity $v_t = \gamma v_{t-1} + \alpha\Delta\theta_t$
10:    Apply update $\theta_{t+1} = \theta_t - v_t$
11: end while
```

Vineeth N B (IIT-H) §4.3 Gradient Descent

Here is the summarized algorithm for Nesterov accelerated momentum, it is once again the same as the momentum based gradient descent, but for step 4 here, step 4 gets your look ahead parameters θ t minus γv_{t-1}, you are going to denote that as tilde θ t and your loss is computed with respect to tilde θ t and that is the gradient that you keep accumulating in your parameter updates and the rest of it follows your momentum based gradient descent algorithm.

(Refer Slide Time: 40:02)

Nesterov Accelerated Momentum: Illustration



Credit: Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) §4.3 Gradient Descent

Here is the illustration of the same example that we saw a few slides ago. This was the momentum traversal that we saw a few slides ago. Let us see how Nesterov momentum performs in the same setting. You see a blue curve starting out there at the same initialization, it seems to form a momentum and here is where it differs, it does not oscillate as much as what momentum does.

The reason being, when you do momentum remember you are oscillating so you have your gradient which takes you to the other bank of that minimum, that gradient tells you to go back to the other bank of that minimum. So, now because your Nesterov momentum is looking ahead it is going to also see where you would go in the next step and use that right away this allows you to minimize these jumping around the minima before you actually converge.