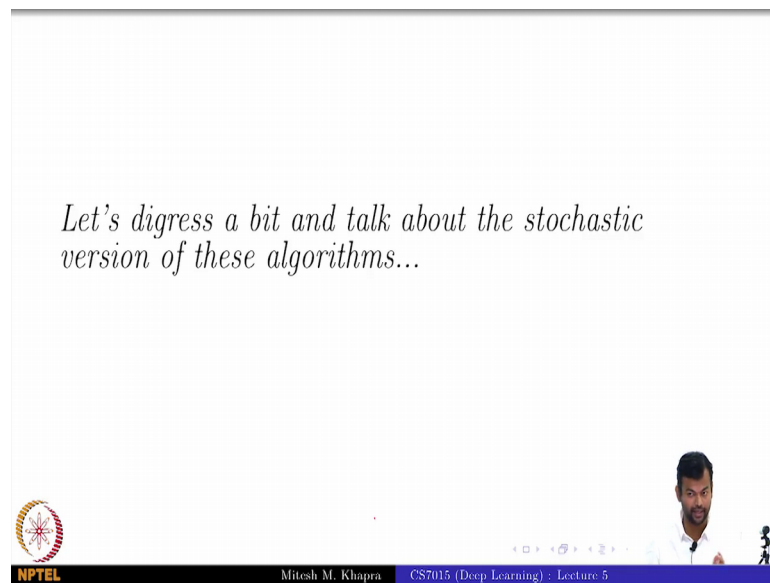


Deep Learning
Prof. Mitesh M. Khapra
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 5.6
Lecture – 05
Stochastic and Mini-Batch Gradient Descent

And now we look at Stochastic and Mini Batch versions of these algorithms ok.

(Refer Slide Time: 00:20)



So, we will digress a bit, actually we should have ended up somewhere else, but I was just going to digress a bit ok.

(Refer Slide Time: 00:25)

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x - b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = 2, 2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why?

NMF

NPTEL Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

So, this is the original gradient descent code that we had, and I have highlighted something in this red box ok.

So, notice that the algorithm actually goes over the entire data once, before making an update. It has going over this entire for loop, which is over all the data points. Of course, in this toy example I had only two data points, but in I practice I will have many many data points. I go over all the data points compute the derivatives and then make this one update why?

Student: (Refer Time: 00:57).

Because that is the right thing to do ok. This was the exact formula that we painfully derived right that the gradient with respect to the loss function right, which we had the summation i equal to 1 to n remember, and the true derivative was a sum of the derivatives with respect to all the data points; that is what we analytically derived and hence we are doing that, it was that is the right thing to do, not for any other purpose that is what it should always be right. So, that is the right thing to do, because this is a true gradient and we actually derived it ok.

(Refer Slide Time: 01:28)

```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5*(fx - y)**2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = 2, 2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

- Notice that the algorithm goes over the entire data once before updating the parameters
- Why? Because this is the true gradient of the loss as derived earlier (sum of the gradients of the losses corresponding to each data point)
- No approximation. Hence, theoretical guarantees hold (in other words each step guarantees that the loss will decrease)
- What's the flipside? Imagine we have a million points in the training data. To make 1 update to w, b the algorithm makes a million calculations. Obviously very slow!!
- Can we do something better? at stochastic gradient descent

NPTEL
Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

And hence this was not an approximation, so all the theoretical guarantees hold. If I do this I know that now this is the true gradient or the true derivative and if I move in the direction opposite to the gradient everything falls in place, because I proved it using Taylor series

But what is the flip side of this. This is the right thing to do, but what is the flip side. If you have millions of point, we will go over all these million points and make this one update. Now imagine the consequence, when you are in a plateau region right, even that momentum or whatever your movement in the plateau is going to be relatively smaller right. You are going over these million points and making that tiny delta update right. So, imagine how much time it will take your algorithm to converge, you get the problem ok


So, the algorithm will take a million calculations and then make one tiny update to your w ok, this is going to be very slow. Can we do something better always right. So, let us take a look at stochastic gradient descent fine.

(Refer Slide Time: 02:36)

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = 2, 2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w + eta * dw
            b = b - eta * db
```

```
def do_gradient_descent():
    w, b, eta, max_epochs = 2, 2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w + eta * dw
            b = b - eta * db
```

- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)




So, I have done a very subtle change to the code what is it, do not tell me indentation, but that is what I have done, so you can tell me that. So, what is happening now? For every data point I am making an update to my w values. This is perfectly fine, yes ok

Now, the algorithm updates the parameters for every single data point. If you have a million data points, how many updates will be make in one pass over the data? A million for every data point will make an update right. So, that slowness factor in what is known as batch gradient descent right, batch gradient descent is when you look at the entire data and then make one update ok.

(Refer Slide Time: 03:19)

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = 2, 2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
        w = w + eta * dw
        b = b - eta * db
```

- Stochastic because we are estimating the total gradient based on a single data point. Almost like tossing a coin only once and estimating $P(\text{heads})$.
- Notice that the algorithm updates the parameters for every single data point
- Now if we have a million data points we will make a million updates in each epoch (1 epoch = 1 pass over the data; 1 step = 1 update)
- What is the flipside? It is an approximate (rather stochastic) gradient
- No guarantee that each step will decrease the loss
- Let's see this algorithm in action when we have a few data points



Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

What is the flip side, what does this module titled stochastic gradient descent? So, what is the flip side? These are not the true gradients. The true gradient is summation over all the points. Now this is no longer the true grading, this is just a point estimator, this is just a approximation of the gradient right. And stochastic, because we are calculating the gradient based on a single data point right, it is a sampling one data point and computing the gradient that this is what the entire population looks like right.

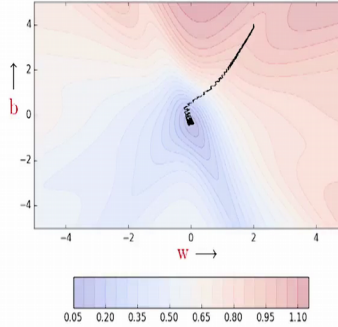
This is almost similar to tossing the coin once and saying that this is what the probability of heads is. If it lands at heads then the probability is one; otherwise its 0 right, you see the error you see the problem with that right, as opposed to tossing the coin a thousand times and then deciding the probability is just tossing it once. So, this is always going to be a denies right this, this is going to be bad

So, now there is no guarantee that each step will decrease the loss why, because the guarantees were only when you are doing the right thing which was to compute the gradients over all the data points. Now there is no theoretical guarantees right, because it is all stochastic now. So, it is possible that in a particular data point your loss might increase also, the overall loss on the data. With respect to that point it might decrease, but the overall loss right

So, now let us see this algorithm in action and I want you to make certain observations about this. So, this is the code that I am going to run now ok. So, let us see ok.

(Refer Slide Time: 04:44)

- We see many oscillations. Why? Because we are making greedy decisions.
- Each point is trying to push the parameters in a direction most favorable to it (without being aware of how this affects other points)
- A parameter update which is locally favorable to one point may harm other points (its almost as if the data points are competing with each other)
- Can we reduce the oscillations by improving our stochastic estimates of the gradient (currently estimated from just 1 data point at a time)
- Yes, let's look at mini-batch gradient descent



Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

So, I will start and you have to observe and let me know and this is really becoming an eye test for all of you, but that is good ok. So, for nothing interesting to observe or already maybe ok

Remember I am running gradient descent, this is not momentum, not Nesterov this is gradient descent ok, I have already given you the answers. What do you observe?

Student:

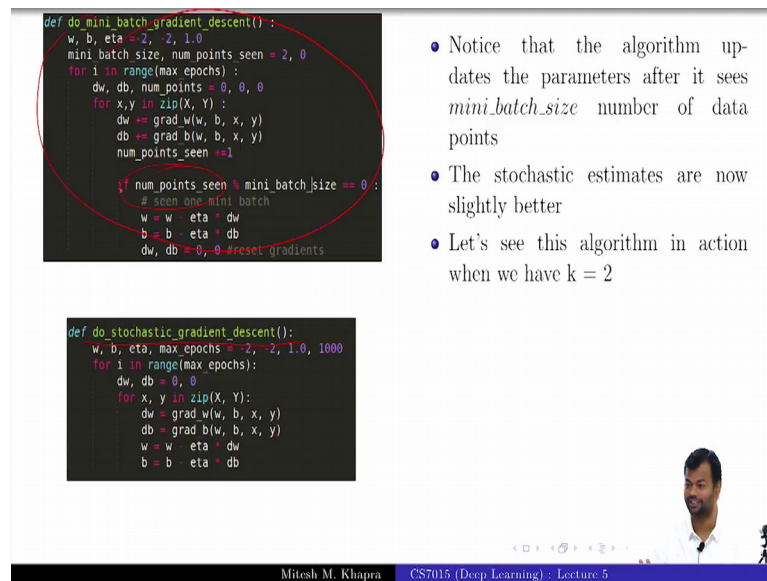
I can still pretend an answer a let us do that. We see many oscillations, why. Why do we see the oscillations? Are these oscillations the same as the oscillations that we see in momentum? No these are different, everyone gets that right, why are there oscillations. What is each click here correspond to one data point right. So, what is happening here? Because we are making greedy decisions right we are looking at one point, this point says to decrease the loss with respect to me move in this direction and we blindly move in that direction right

Now, we look at the next point, it says oh no no wait you need to move in this direction. So, we again move in that direction. So, all these points are actually trying to just make things better for themselves. They are not thinking about what is happening to all the other points in my data right. So, all these points are actually competing with each other.

So, some decision which I took with respect to where to move, which was locally favourable for one of these points may not be good for the other point right. Hence I keep these tiny oscillations which I make, these are the stochastic noise that you are seeing now right, everyone gets this ok. This is fine ok.

Now, can we reduce the oscillations by improving the stochastic estimates? Always yes fine. So, let us see what do I mean by that.

(Refer Slide Time: 06:57)



```
def do_mini_batch_gradient_descent():
    w, b, eta = 2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs):
        dw, db, num_points = 0, 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points_seen += 1
        if num_points_seen % mini_batch_size == 0:
            # seen one mini batch
            w = w - eta * dw
            b = b - eta * db
            dw, db = 0, 0 # reset gradients
```

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = 2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

- Notice that the algorithm updates the parameters after it sees *mini_batch_size* number of data points
- The stochastic estimates are now slightly better
- Let's see this algorithm in action when we have $k = 2$

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

So, we look at a mini batch version of this. So, what I am going to do is, instead of. So, this code is actually for mini batch stochastic gradient descent, it is a very minor alteration on the stochastic gradient descent, I will just let you stare at it for a minute or. So, what I am doing here is, I am, instead of doing it for every point, I am waiting for a certain number of points and then making the update right, that is what I am doing here ok

Now, for this I have kept k equal to 2 what does that mean? I look at two points compute the derivatives with respect to them and then make an update for two points at a time ok, what do you expect? No what do you expect with respect to this code ok.

(Refer Slide Time: 07:47)

- Even with a batch size of $k=2$ the oscillations have reduced slightly. Why ?
- Because we now have slightly better estimates of the gradient [analogy: we are now tossing the coin $k=2$ times to estimate $P(\text{heads})$]
- The higher the value of k the more accurate are the estimates
- In practice, typical values of k are 16, 32, 64
- Of course, there are still oscillations and they will always be there as long as we are using an approximate gradient as opposed to the true gradient

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

So, let us see we will try to run this now and you will start seeing a red curve here and make some observations about this ok. So, this is the red curve, yeah its visible. Oops I do not read any of those.

Student: (Refer Time: 08:07).

If you need to fix this right ah, these bullet us should come only after the curve has finished this journey ok. Do not read any of that ah. So, what do you see about the red curve? It is completely contained inside the black curve; that means, its oscillations are smaller than the black curve right. Does that make sense why this is happening, because now you are not listening to just one point, you are listening to two points and then at least you are doing something better right instead of just taking one.

So, what is the analogy with respect to our coin toss experiment? You are tossing the coin twice and then deciding what is the probability are heads or tails right. So, it is always going to be slightly better than tossing it only once right. And now what would happen in the limit if I keep increasing this, you will end up with a batch gradient descent where you look at the entire data

So, looking at only one data point is bad, because it is very noisy, looking at the entire data is bad, because it is very time consuming. So, you need to do something in between which is mini batch gradient descent ok, and typically you look at values of 16 32 64, but

it also depends on the amount of data you have and if you have a billion points you might actually want to look. Because if you have a billion points and you have a bad size of 64 you will take 1 billion by 64 times to finish the data once right.

So, you might want to keep a larger batch size at that point right, but just ignore that, but you will try different batch sizes and see which one works better. So, in the assignment I will be asking you in to experiment with bad sizes, yes ok. No sorry wrong question, I will be asking them to implement stochastic and mini batch also or only vanilla.

Student: Mini batch.

Mini batch fine that is fine ok. So, you will see this in your assignment ok. So, everyone sees what was the difference between stochastic and mini batch right, you have better estimates now and therefore, this red curve is contained inside the black curve fine ok.

(Refer Slide Time: 10:08)

Some things to remember

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	$\frac{N}{B}$

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

So, you have some things to remember; one epoch get used to this terminology, 1 epoch is 1 pass over the entire data. One step is one update to the parameters, n is equal to the number of data points and b is equal to the mini batch size. Now you have to fill in the second column. In vanilla or the batch gradient descent what is the number of steps that you take in 1 epoch.

Student: 1.

1 in stochastic gradient descent

Student: N.

N n.

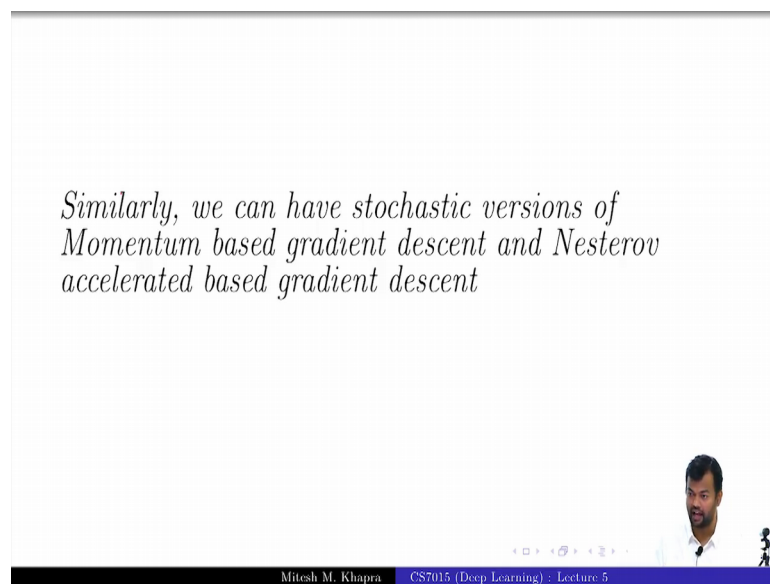
Student:N.

In mini batch gradient descent.

Student: N by b.

N by b, everyone gets that. So, get used to this ok. So, this epoch step batch size all this is something that you will see regularly when you are reading papers on deep learning fine.

(Refer Slide Time: 10:53)



So, similarly we can have the stochastic versions of momentum based gradient descent and Nesterov accelerated gradient descent ok.

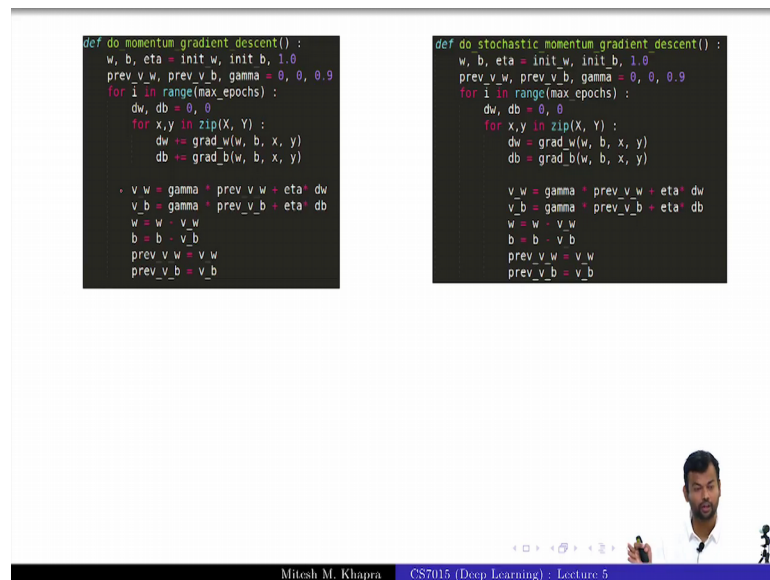
(Refer Slide Time: 11:02)

```
def do_momentum_gradient_descent():
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b

def do_stochastic_momentum_gradient_descent():
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)

            v_w = gamma * prev_v_w + eta * dw
            v_b = gamma * prev_v_b + eta * db
            w = w - v_w
            b = b - v_b
            prev_v_w = v_w
            prev_v_b = v_b
```

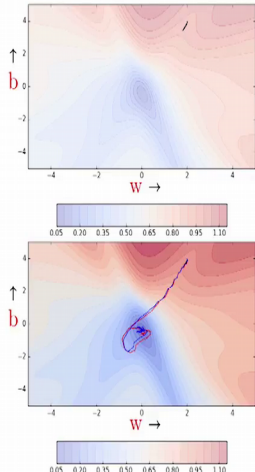


Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5

So, these are just the codes, it is very easy to see what is happening here. Again basically this is just an indentation right. So, if you look at the difference between the two codes, I have just indented it inside; that means, I am making these updates for every data point right and same thing you could do for Nesterov also ok.

(Refer Slide Time: 11:21)

- While the stochastic versions of both Momentum [blue] and NAG [red] exhibit oscillations the relative advantage of NAG over Momentum still holds (i.e., NAG takes relatively shorter n-turns)
- Further both of them are faster than stochastic gradient descent (after 60 steps, stochastic gradient descent [black - top figure] still exhibits a very high error whereas NAG and Momentum are close to convergence)



NPTEL Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 5 63/87

Now, let us see ah. This guess what is it? This is the gradient descent, stochastic gradient descent ok. Now let us see if you have really understood nag and momentum based

gradient descent. One of these curves here corresponds to stochastic nag, the other one corresponds to stochastic momentum. Tell me which one is which

Student: Blue pill.

Blue pill red pill. Blue is.

Student: (Refer Time: 12:01).

How many of you say that. Ok I am confused ok. How many of you say that blue is momentum, how many if you say that red is momentum? Oh there is so many, you do not have an opinion

Student: Sir not clear.

Not clear, I will buy that. So, ok. So, look at this, who is taking longer u turns; momentum or nag? Momentum roughly which guy is taking the larger u turns?

Student: Red guy.

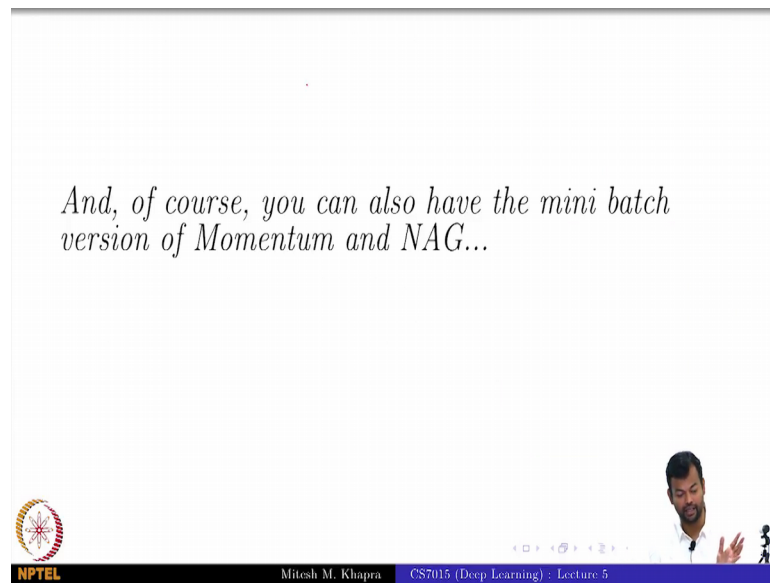
Red guy right, I mean roughly speaking. There is only one point to judge by this here, because here they are almost same and that could happen in practice right, because this is now noisy ok. So, the red curve corresponds to.

Student: Momentum.

Momentum, because it is taking a larger u turn, we saw that momentum takes larger u turns and the blue curve is corresponding to nag ok. So, no I remember this was an error on the slide yeah. So, this has to be red and this has to be blue, so ok. So, the momentum is actually red and the nag is blue because it is taking a shorter u turn and the reason you do not see it very clearly is because both of these are running in this stochastic mode right ok.

But you still see the relative advantage of them that nag still takes shorter u turns, both of them are faster, still faster than vanilla gradient descent.

(Refer Slide Time: 13:35)



You see that black curve at the top and both of these are faster than them, both of them, all three have run for the same number of iterations right. After 60 steps you see what happens to stochastic gradient descent and what happens to nag and momentum basically understand right fine. And of course, you can have the mini batch versions of momentum and nag also.