

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

Module - 5.9
Lecture – 05
Gradient Descent with Adaptive Learning Rate

So, finally we come to the more interesting part. So, in this module we look at Gradient Descent with Adaptive Learning Rate. So, first we will see motivation or intuition for why we need this and once you get the motivation, I believe the rest should be straightforward, ok.

(Refer Slide Time: 00:31)

• Given this network, it should be easy to see that given a single point (x, y) ...

$$y = f(x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$
$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$
$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b)$

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

So far what we have been doing is, please pay attention on this slide. I need to define some notations and you should not get confused with that, ok. So far we have been dealing with the situation, where we had just one feature which was x and one weight corresponding to it which was w and one bias which corresponded always on input, right. Now, we are going to look at the situation where we have more than one inputs. That means, earlier we were basing our predictions only based on the director and now, we are the director, actor, genre, IMDb ratings and so on, right

So, here $x^1 x^2 x^3 x^4$, these are four different features or four different inputs that I have, and this is not x square. Just I know it is obvious, but I am just making it clear,

right. So, this is $x^1 \times x^2 \times x^3 \times x^4$, ok. It is not probably the best choice of notation, but I will just stick to that. So, now each of these has a corresponding $w^1 \ w^2 \ w^3 \ w^4$, and this is how your decision looks like. It is the dot product between the weight vector and the input vector, ok. This is how I am going to decide and that is a single sigmoid neuron again.

So, given a single point x, y , do I need to again go through this computation? Sorry w, p oh sorry, ok. I will just erase this. So, this w is actually the vector w . So, it includes $w^1 \ w^2 \ w^3 \ w^4$ and I am trying to take the derivative with one element of that vector. Do I need to show you how to compute this? Have you seen this before? Can you tell me I will show you the derivative with respect to w^1 , can you tell me it will be a product of some terms? Can you tell me what is the last term going to be?

(Refer Slide Time: 02:41)

x^1
 x^2
 x^3
 x^4
 1

$y = f(x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$

$x = \{x^1, x^2, x^3, x^4\}$

$w = \{w^1, w^2, w^3, w^4\}$

- Given this network, it should be easy to see that given a single point (x, y) ...
- $\nabla w^1 = (f(x) - y) * f(x) * (1 - f(x)) * x^1$
- $\nabla w^2 = (f(x) - y) * f(x) * (1 - f(x)) * x^2$... so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient
- What happens if the feature x^2 is very sparse? (*i.e.*, if its value is 0 for most inputs)
- ∇w^2 will be 0 for most inputs (see formula) and hence w^2 will not get enough updates
- If x^2 happens to be sparse as well as important we would want to take the updates to w^2 more seriously
- Can we have a different learning rate for each parameter which takes care of the frequency of features ?

76/87
 MITESH M. KHAPRA CS7015 (Deep Learning) : Lecture 5

Everyone gets this? You remember this form. So, only thing which is changing is this guy, right. So, this part is exactly what we have derived and when we had one input, we just call it x and now, we have multiple inputs. So, it will depend on that particular input, right whichever w^1 corresponds to, ok.

Now, make an interesting observation there, ok. So, sorry before that yeah this is obvious if there are n points, we will just take the sum of the gradients with respect to the n points, ok. Now, what happens if the feature x^2 is sparse? What do I mean by that? It is mostly 0, ok. What does that mean? So, I am looking at lot of movie data, ok. Amir Khan

acts in a very few movies. So, if I have a feature which says actor Amir Khan, then that is going to be 0 for most of the movies in my data scene, right. That is what I mean by sparse, ok.

So, if I have 10,000 movies, then probably only 50 of them would have this feature as one, ok. Does that make sense? So, it is going to be very sparse. Now, if the feature is sparse, why do we care about it? What will happen? What do we really care about when we are talking about optimization in this course, the gradients, right. That decides how well we move in the plane that we are considering w b plane or the other in the end dimensional region that we care about, ok.

So, now if x_2 is sparse, what would happen to this? It will be 0 lot of times because x_2 is 0 lot of times, right. So, now just take a minute to understand this, right. So, now remember let us talk about stochastic gradient descent or mini batch gradient descent or even batch descent. You are going over all the 10,000 points that you have, you are computing the gradient with respect to all the parameters.

One of those parameters happens to be w_2 , right. You have gone over 10,000 points, but in how many of those you will actually get the gradient for this. Only in the 15 which x_2 was present, right. Everywhere else the gradient would be 0. So, that means your sum of the gradients, overall the endpoints is going to be small or big.

Student: Small.

Small for this particular feature or for this particular weight; it is going to be small, right because you do not have enough samples where you are seeing this. So, now what would happen to the update? You started with a random value for w_2 , ok. After one epoch or making one entire pass of the data, what would happen to the updates for w_2 ? Very small, very few updates compare this to a feature which is dense. Do you get a lot of updates? So, you see there is something unfair happening here. If a feature is sparse, it is not getting updated enough, right.

Now, that was ok. In one situation if this feature was not really important, but now consider the exact example which I gave you which is this, an Amir Khan movie or not, but suppose I am doing a classification whether this movie is going to be hit or not, I would believe this feature is very important because almost always when he is the actor,

the movie is a hit, right. So, you really cannot ignore this feature. You want to learn the parameters correctly for this feature. Do you get the setup right? There could be cases where your feature is very sparse, but at the day at the same time very predictive of the output that you are trying to learn, right and in this case, the output is whether the movie would be a hit or not, ok.

The other example could be is Christopher Donald, the director. So, yes probably directed less than 10 movies, but all of them have been at some point in the IMDb top 250 or something, right. So, that is a very important feature, but you will not get it very frequently in your data, right. So, you cannot really ignore these features. That means, you still want to learn these features properly. So, you have sparse features; you have dense features. We understand that for the sparse features, the updates would be slower and for the dense features, the update would be faster, ok. All of us get this, fine. Oh no, the sparse would be 0 in most cases. No no. So, you will do this 0 mean thing, right.

No, but if it is a same value and you are going to 0 mean the data, right. So, the value even if it is 1, it is going to be very close to 0, right. So, you always assume 0 means otherwise all this does not make sense, right because if your features are not in the same range, then anyways you are in trouble, right fine. So, this is what I was trying to say that the gradient with respect to w_t is going to be 0 for most inputs and hence, w_t will not get enough updates and as I said if this is an important feature, we cannot really ignore it. We have to make sure that it learns better.

So, what is the case that I am making for? What do we actually need? Can you relate it to the discussion on learning rate that we have been having? So, if the feature is sparse, you know it is going to get very fewer updates. So, can we change its learning rate, so that feature gets updates a bit faster as compared to the other features. So, you get the motivation, right? How to do this is a separate story, but at least we need to do this. Is that clear? How many of you get this? Ok fine.

(Refer Slide Time: 07:49)

Intuition

- Decay the learning rate for parameters in proportion to their update history (more updates means more decay)

Update rule for Adagrad

$$v_t = v_{t-1} + (\nabla w_t)^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

... and a similar set of equations for b_t

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

So, the intuition is decay the learning rate for parameters in proportion to their update history. So, you have been recording the update history, you have been looking at the parameter, you know all the gradient w^2 that you had calculated so far, right. How many times you had computed the gradients and what those values were actually now for these sparse features? Those are going to be 0.

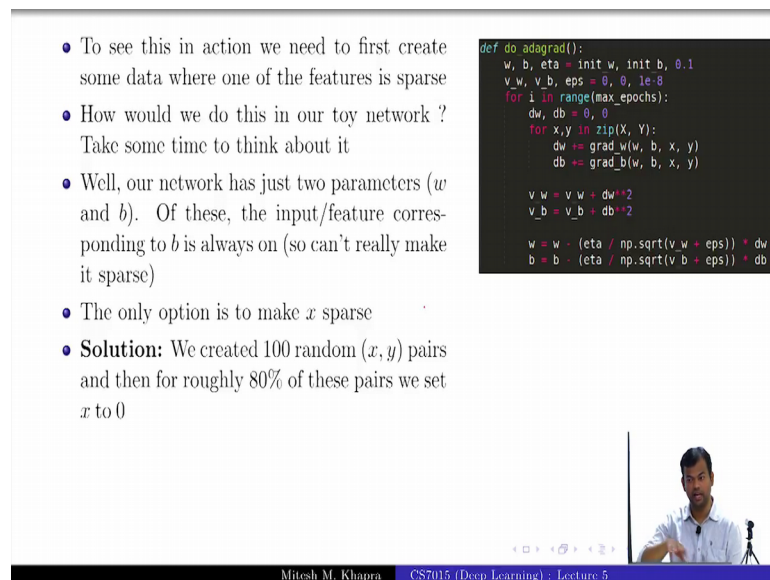
So, your cumulated history is going to be small, right. For a dense feature, it is going to be high. So, why not make the learning rate inversely proportional to this history. That means, if the feature has been updated fewer times, give it a larger learning rate. If it is not updated, if it is updated many times, give it to a smaller learning rate. Can you give me a mathematical formula for doing this? This is the intuition. Just think about it for a minute learning rate inversely proportional to update history, ok good. How many of you get that, but most of you will get it once I show you the answer.

This is my gradient which I had computed so far. I mean at this time, step I will keep accumulating it in a history vector, ok. So, at time step 0, I will take the magnitude of this again. I am taking the magnitude right because it does not matter whether you made an update in the positive direction or the negative direction, you just matters that whether how much, by how much it move. So, I will just square this quantity, so that I can get rid of the sign, ok. So, I am taking the magnitudes and I am storing all that. So, at time step t what would v_t contain? It is $\text{grad } w_0^2$ plus w_1^2 $\text{grad } w_1^2$ and so on

up till time step t , ok.

Now, this was my if I ignore this quantity, this was my normal gradient descent update rule, ok. Now, do you see what I have done? I have divided the learning rate by whatever history I had accumulated. So, for the dense features what would happen is, the learning rate will increase or decrease with time, the learning rate will decrease, right and for the sparse features, relatively less right; In fact, if you have written gotten 0 updates so far. So, when you have to update the first few times, you will have a very high learning rate. Does that make sense, right because this quantity would be 0. So, our eta would actually be very large, ok. So, you see how that intuition got converted into some reasonable formula, fine.

(Refer Slide Time: 10:13)



- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network? Take some time to think about it
- Well, our network has just two parameters (w and b). Of these, the input/feature corresponding to b is always on (so can't really make it sparse)
- The only option is to make x sparse
- **Solution:** We created 100 random (x, y) pairs and then for roughly 80% of these pairs we set x to 0

```
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    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 = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

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

Now, can you tell me a way of actually realising this? I want to show you that what happens when you have sparse data and I want to do this with the example that we had where we had only one feature and other feature was always on, right. So, how do I create this sparse data? So, you should think about these because these are things you will have to do when you are practising machine learning and if you are working with the problem and you want to create some simulated data, so that you can verify some hypothesis that you have. So, how would you do this?

See I am going to create thousand data points, right which is x, y points and of course, I have this $x = 0$ which is always on, right. So, $x = 0$ is always on. I cannot make that sparse.

What about the other feature? If I am creating thousand data points, what should I ensure is that 80 percent of them or some 90 percent of them is always 0, right. Just as the Amir Khan case and most of the data it is going to be 0. So, what we will do is as I said we just have two parameters w and b . We cannot make sparse is always going to be on. So, what we will do is, we will make x sparse, we just create random x , y sparse and then, for 80 percent of those we will set x to 0, right. So, now this x feature is going to be very sparse. Is that fine?

So, now I have created some data which is sparse. One of the features is sparse and now, I want to see what happens when I run gradient descent momentum and Nesterov accelerated gradient descent and how does the algorithm behave and now, if I apply this algorithm which I did not name, it is called Adagrad, ok. This algorithm is called Adagrad if I apply this algorithm, then what how does the situation change, fine.

(Refer Slide Time: 11:55)

- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical (b) axis and there is very little movement along the horizontal (w) axis
- Why? Because in our data, the feature corresponding to w is sparse and hence w undergoes very few updates ...on the other hand b is very dense and undergoes many updates
- Such sparsity is very common in large neural networks containing 1000s of input features and hence we need to address it

Let's see what Adagrad does....

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

So, this is what gradient descent momentum and nag do. Now, at least the difference between momentum and nag should be clear. Nag blue curve is inside the red curve right. So, oscillations are slightly smaller, ok. This is how they behave, ok.

Now, there is something very interesting that these algorithms are doing for this particular data set that I have created. Can you spot it? What is the interesting thing happening here I want you to take some time and think about and relate it to the discussion that we just had how many of you see what is happening here, ok. Very few I

will give a hint, ok. It is almost as if these algorithms went to a school where they did not teach Pythagoras theorem. Now, related to the discussion that we just had what is happening initially. So, initially what is happening is you started from here, and this is the w, b planes. So, you have w on the horizontal axis and b on the vertical axis.

What is happening to all your updates initially where are you moving? You are moving along the b direction. Are you making any movements along the w direction? No. Why was w sparse? Its gradients are mostly 0. It was not being able to make any updates in the w direction or it was able to do make updates in the b direction. It did as much as it could do after reaching here it realizes that there is no point in going to be further, right. It actually took u-turn because it realise that there is nothing I cannot really go ahead. I have to now start working in a direction of w .

So, now in practice although in this toy example, it does not, it still converges fast, but in practice what will happen is you have just moved in one direction, reached a point and now from there again you are going to take right turn and reach to your destination, right. So, you are taking, you are doing something which is not fast. This is not how you would go from this point to this point. There has to be a better way, right and this is happening because w is not getting updated frequently. All the updates are initially done for b .

Now, when it is no longer possible to change d because your d is the optimum value for b , then only you start changing w and that to very slowly because it will have to wait for many updates. For that to happen how many of you get this? So, this is exactly what is written on the slides because in our data the feature corresponding to w is sparse and hence, w undergoes very few updates and these very dense and it undergoes a lot of a updates, ok.

Now, such sparsity is very common in large neural networks which have thousands of features, right. So, you can imagine this. Now, if I have thousands of features, now suppose I am doing credit card fraud detection, now say one of my features is corresponding to some education that the person had and suppose he has done some very less sort after degree or less sort after curriculum.

So, that feature is going to be sparse where most of the cases, but I cannot ignore it. May be this is the most predictive feature that I might have, right. So, you could think of various cases where you have thousands of features out of which many are going to be

off for a given example, right. Everyone sees that this is the real world scenario where lot of your features are going to be sparse and in many cases, you cannot ignore the sparse features, fine. Now, let see what Adagrad does. Any guesses?

(Refer Slide Time: 15:18)

- By using a parameter specific learning rate it ensures that despite sparsity w gets a higher learning rate and hence larger updates
- Further, it also ensures that if b undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)
- What's the flipside? over time the effective learning rate for b will decay to an extent that there will be no further updates to b
- Can we avoid this?

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

So, I am running this, we should start seeing something a green curve starting from here, ok. Do you see what is happening expected? Now, try to guess if you are going to run into a problem. I have deliberately halted the algorithm. I just want you to think if you are going to run into a problem, ok. All of you think you have something which makes sense. So, now I have run it for in this case again this is the toy example. Hence, you do not see a lot of difference between these algorithms in terms of number of steps taken to converge, but in real world application, it would be very different, but now what has happened is I have run the algorithm for as much I can and I am then stuck here. I am not being able to move forward. Why is this happening?

Well, I am the histories accumulating it is growing. Now, what am I doing to the learning rate? I am just killing it, right. It is eta by a very large constant. Now, that is going to be very small. So, no matter how big my gradient is, it is going to get multiplied by a very small learning rate and I cannot just move any forward anymore, right. So, see that will happen. That is why in this case this is some point here which I do not want to go over now and it is this.

In fact, I do not have an explanation for that, but this one observation which people have

made that remember we have the square root in the denominator. If you remove the square root in principle, you are still doing the same thing, right. You are still making it inversely proportional to a cumulated history, but it does not work well when you do that. That I do not know why it happens and I just read these comments at several places that it does not work when you remove the square root from the denominator, but that is not important for this discussion. That is just point for reference later on, ok.

So, right now what I am trying to say is that it did the right thing. It started making updates for w also and started making larger updates, hence we see this simultaneous moment and o w and b direction. But the flip side is over a period of time, the effective learning rate for b will decrease so much that we no longer be able to move in the vertical direction, right and if I am not being able to move in the vertical direction, we will not reach the minima. In this particular example not always, but in this particular example you need to move further in the direction of b , but a learning rate is not allowing you to do that, right. So, that is what is happening.

So, now can you avoid this? Yes. How? Multiply by, so first divide it, so that the decreases, then multiply it. So, that does not decrease. All of these are interesting ideas. I am not I mean it is very, we say upfront whether this is wrong or right, but yeah these are you get the idea basically something is happening which is you are aggressively killing the learning rate.

(Refer Slide Time: 17:59)

Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)
- As a result after a while the frequent parameters will start receiving very small updates because of the decayed learning rate
- To avoid this why not decay the denominator and prevent its rapid growth

Update rule for RMSProp

$$v_t = \beta * v_{t-1} + (1 - \beta)(\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

... and a similar set of equations for b_t

$\beta \geq 0.95$
 $v_1 = 0.05 \nabla \omega_1^2$
 $v_2 = 0.95 * 0.05 \nabla \omega_2^2 + 0.05 \nabla \omega_2^2$
 $v_t = \nabla \omega_1, \nabla \omega_1$

81/87

Now, I just want to make sure that you are not so aggressive, ok. So, what happens because of the aggressive killing, is the frequent parameters, they start receiving fewer updates. Now, this is what RMSProp does, ok. I want you to stare at this for a minute, ok. Assume that beta is going to be something which is greater than 0.90 or 0.95 or something and try to make sense of what is happening, try to imagine what v_t is going to look like, in terms of g_0 , g_1 and so on. To start from v_1 and see what happens what was v_1 earlier and what it is going to be now ok, but it still grows my magnitude when I am still adding stuff. So, how does it help me in not blowing of the denominators?

So, yeah I think you most of you get. So, again this is the trick is basically you are using this exponentially, exponential moving average, right. So, even at the first step earlier I was doing g_t^2 . Now, actually doing 0.05 into g_t^2 , oh sorry g_1^2 , right. So, that is what my v_1 is going to be. Now, what is my v_2 going to be? It is going to be 0.95 into 0.05 g_1^2 plus g_2^2 , right. So, this quantity is even shrinking further and at each step this is going to keep a 0.05, and you see now at each step this is going to get multiplied by this quantity and shrink further, ok.

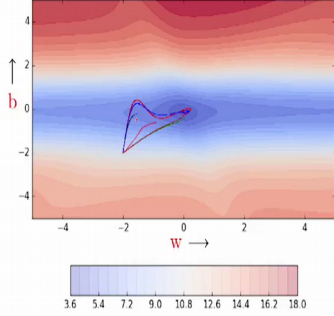
So, now I am not aggressively growing the denominator. I am not considering the full gradient, but only a fraction of it and in fact, a very small multiple of it. So, I am still accumulating the history, but I am not being very aggressive while doing that, right. So, you understand this. You did not get this, ok.

(Refer Slide Time: 19:54)

```
def do_rmsprop():
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 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 = betal * v_w + (1 - betal) * dw**2
        v_b = betal * v_b + (1 - betal) * db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



- Adagrad got stuck when it was close to convergence (it was no longer able to move in the vertical (b) direction because of the decayed learning rate)
- RMSProp overcomes this by being less aggressive

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

So, now let us see if we run what would happen. Any guesses, ok? So, initially now this is I think a brown curve. It is already there, but you can see it. So, I will keep running it and at some point, it will diverge from the green curve. Yeah do you see that? Now, I have reached its destination, right; so, at the point where the b learning rate, the learning rate for b was getting killed. In this case that does not happen because you have prevented the denominator from growing very large actually multiplied by its small values, so that it does not grow very fast. How many of you get this? Ok. Not as many as I would like. Can you please raise your hands, ok? Yeah some people as just lazy fine.

So, Adagrad got stuck when it was close to convergence because the learning rate was killed and it was no longer able to move in a direction of b , but for RMSProp, it overcomes this problem by not growing the denominator very aggressively, ok. Now, can you think of any further modifications? There is everything that you learned so far and my everything yeah.

Yeah I am not very sure why that I agree that I am also bit surprised that it completely overlaps with it. I checked it and that is how it turns out to be and guessing it is an artifact of the artificial data that I have created. So, it is trying to say is actually making sense that it should not overlap so much, right. Initially it should slightly be biased towards b and then, probably that is what you are trying to say right, but I told it just an artifact of this data that I have, but what matters is from as going to say illusion, but from

the illustration is that it actually does not kill the learning rate, right.

(Refer Slide Time: 21:48)

Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t \nabla w_t$$

$E[m_t] = E[g_t]$

$m_t =$

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

So, what is the one idea that now think of everything that you learned in starting from gradient descent, then you tried to improve it using something, then you tried to further improve it and so on and now, we have taken a slide d 2. From there you are now focusing on the learning rates, but there were other things which you are doing earlier. Can you bring those back, add momentum. How many of you say add momentum as if I can just added. You are right actually, ok.

So, let us see what we can do. So, it does everything that RMSProp does. That means, it tries to make the learning rate inversely proportional by sane to cumulated history. By sane mean it does not allow the x 2 blow up and it also will use the cumulative history of the gradients. So, let us see the update tool for Adam. So, what is this term doing? Actually it is taking a moving average of; there is the same as the momentum base role, right. Just taking a moving average of your gradients, ok. The same analogy that I am going to phoenix market city, I am just taking all my history into account, and v_t is again a cumulative history. This is the same as what was happening in RMSProp, right where you get lost. Is this fine? Everyone is with this, ok.

Now, what would be the next step? Can you give me the final update rule? At least think about it. m_t into v_t , no. Ok just try to think about it and it is very hard to say it out there are too many grads and suffixes and so on. So, just think about what you did in the

momentum case, ok. Now, there is one more step which I am going to ignore. I will just say what that step is and then, I will come back to that later on.

So, this is something known as bias correction, ok. Just ignore it for the time being. I will come back to this discussion. Just for the time being just assume that I am taking m_t and dividing it by some quantity, right. So, for all practical purposes I am just using m_t , just dividing it by a quantity, just for now. That should suffice and then, my final update rule is going to be this. How many of you expected to do with this or at least understand that this is fine or not many, ok.

So, let me go over this. What did you expect here in a normal gradient descent? They should have been $\text{grad } w_t$. That means the derivative with respect to current w , ok. Instead of that I am using a cumulated history. Instead of using just this quantity, I am using a cumulated history. Does it make sense? This is same as momentum based gradient descent. How many of you get that? Ok and now, this quantity there is nothing new. This is the same as what RMSProp suggested that you divide the learning rate by a cumulated history of gradients, right. So, just a combination of these two, one is take care of the learning rate and the other is use a cumulative history. Does it make sense now? Ok, fine.

Now, this part is something that I need to tell you about. So, I will tell it to you after I run the algorithm and then, I will come back to that, but is the update rule clear that it is a combination of momentum plus killing the learning rate, fine.

(Refer Slide Time: 25:19)


Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t \leftarrow$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2 \leftarrow$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

... and a similar set of equations for b_t



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

It is a similar set of equations for bt.

(Refer Slide Time: 25:21)

```
def do_adam():
    w, b, dw, db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], []

    w, b, eta, mini_batch_size, num_points_seen =
    init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0, 0, 0
    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)

        m_w = beta1 * m_w + (1 - beta1) * dw
        m_b = beta1 * m_b + (1 - beta1) * db

        v_w = beta2 * v_w + (1 - beta2) * dw**2
        v_b = beta2 * v_b + (1 - beta2) * db**2

        m_w = m_w / (1 - math.pow(beta1, i))
        m_b = m_b / (1 - math.pow(beta1, i))

        v_w = v_w / (1 - math.pow(beta2, i))
        v_b = v_b / (1 - math.pow(beta2, i))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b
```

- As expected, taking a cumulative history gives a speed up ...

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

Now, let us see what happens to this algorithm is actually call at Adam. It stands for Adaptive Moments, right. Yeah what is can you tell me why that name?

Why moments?

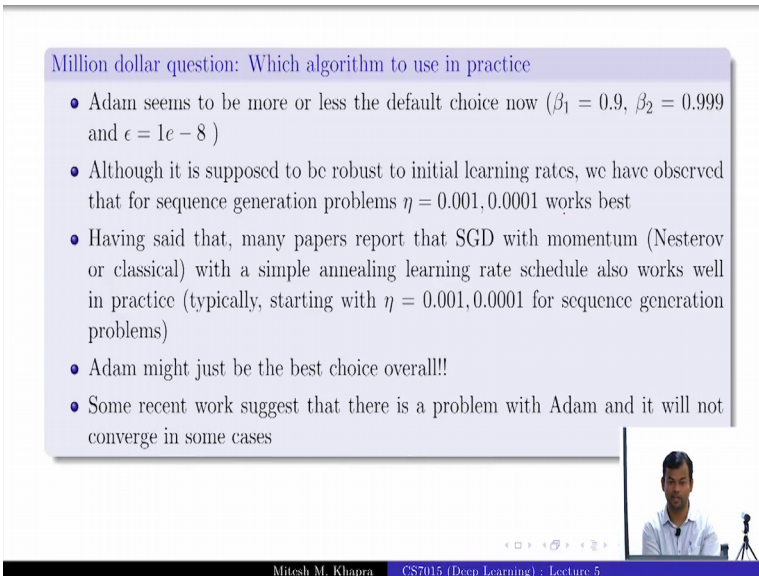
Student: Sir, mean is.

Good, where is the mean? Here this is a mean. This is a moving exponentially weighted average, right. This is an exponentially weighted mean. What about this? What is this quantity? If you take the average of this is the second moment, right exponentially weighted second moment, right. So, using the first moment and the second moment we come up with an adaptive learning rate, ok. Is that fine?

So, now I will run this algorithm. Are you able to see this? See a coloured curve k. So, it is here you see that now do you see what happen? Do you see this curve? Everyone sees that, ok. So, what is happening, it is taking u-turns, right. So, again whatever happens because of momentum, it is happening in this case and then, finally it will converge again. Let me be clear that in this case now it should be very clear. We need to change, who is TA for the slide. So, this colour needs to be changed or it should be bright right from the first. So, what is happening is it is getting overlaid and then, it becomes bright when we need to have a brighter colour right from the beginning, ok.

So, this again in this toy example, right; You do not really see the speed as such because all of them are converging you know almost the same number of steps, but this again I repeat for the toy example, but at least you see that the behaviour is very different and behaviour is consistent with whatever you have put into the update rule, right. In one case the learning rate gets killed, in the second case it does not decay and in third case when you using this moments, sorry this momentum term you again have this behaviour similar to the momentum gradient descent, where you actually overshoot and then, you come back, ok. So, is that clear all these algorithms? Now, here is the million dollar question.

(Refer Slide Time: 27:53)



Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e-8$)
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems $\eta = 0.001, 0.0001$ works best
- Having said that, many papers report that SGD with momentum (Nesterov or classical) with a simple annealing learning rate schedule also works well in practice (typically, starting with $\eta = 0.001, 0.0001$ for sequence generation problems)
- Adam might just be the best choice overall!!
- Some recent work suggest that there is a problem with Adam and it will not converge in some cases

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

Which of these do you use in practice? So, what are the options that you have for your back propagation assignment? Even if you have not read the assignment, you should just tell me based on whatever you have learned you have gradient descent.

Student: Momentum.

Momentum.

Nag RMSProp.

Student: Adagrad.

Adagrad Adam, ok. So, which of these would you choose and if there is one or which is

called eve, but it did not really gain much momentum, but Adam. So, in practice Adam seems to be more or less the default choice. I should tell you that recently there was a paper or called couple of papers which actually show that there is a slight error I mean there is you could showcase where Adam will not actually converge as expected with, but still then after that as is the case in whole of deep learning resources that one person sees this work and immediately the next is someone else. This does not work or vice versa, right.

So, someone show that this does not work. Adam does not work in some cases, but then someone else did detailed study showing that in most practical applications, you have taken a toy data set where you can show something under some conditions. Adam will not converge, but if I look at real world data sets like m or image data or something, those conditions do not hold there. So, Adam really works well. So, in practice Adam is more or less the standard choice. Nowadays at least all the image classification work which deals with convolutional neural networks and convolutional neural networks and so on, that uses Adam as the optimization algorithm.

We have used it largely for a lot of sequence to sequenced learning problems and it works well. Although it is supposed to be robust to the initial learning rate, right because you are tampering with the learning rate as you go along, right. You are not sticking to eta, but you are conveniently blowing it up or shrinking it based on your requirement. So, it should not be sensitive to the initial learning rate, but we have observed that at least for the sequence generation problems if you use one of these learning rates as a starting point, they work best of course. Of course these are heuristic, right. We also depends on how much data you have and so on.

If you are going to train, but only thousand samples and first of all of course you should question why are you using deep learning, but you have gone pass that question already. Has everyone else has, then you are still be using a deep neural network and in that case may be these learning rates are going to be very small, but in general for a large number of data sets out there which lot of academic research happens which are of reasonable size, these learning rates happen to be well in tractors, ok.

Now, having said that many papers report that sgd with momentum either the Nesterov momentum or the Vanilla momentum with a simple annealing learning rate; We

remember we did this learning rate decay either a constant decay or that heuristic decay that after you look at the validation loss and then, decide whether to decay or not. That also seems to work at part of the Adam, right. So, my advice be that if you really know what you are doing with sgd and momentum right, that means if you really know how to look at the laws, how to track it, how to adjust the learning rates and so on.

With a little bit of manual tampering, it should work as well as Adam, there are people which show that it works well as Adam, but if you are just a practice. There who does not really want to bother too much about setting the learning rate, setting the momentum, setting the schedules on both of them, remember for momentum also we had a schedule and was just given by one of these papers and it might differ for your application. You might want to treat that a bit. So, if you are not really bothered about doing all these things, then Adam would just be over all the best choice, right with very minimum tempering of the initial learning rate, right.

As I said some recent work suggested there is a problem with Adam and we will not converge in some cases, but then it still I mean I would say that juries not out on that yet because there is of course theoretical angle to it and also, the practical angle. Again practice has been used widely for the last 3 to 4 years. At least and it works well in a large number of applications, right. So, that is why Adam would typically be the overall best choice, ok.

Now, there is this one thing which I need to do which is I need to tell you why do we use this bias correction? So, now what do you actually want to you are taking a mean, ok. You do not want to rely on the current estimate of the gradient, but you want to take an exponentially moving average of the gradients, right

Now, what would you actually would be doing all this, what is the inclusion behind this since you are talking about moments and so on? Can you think in terms of probability distributions? So, let me just try to say this we write that your gradients, your values of $grad_{wt}$, right and I will just I think alternately use gt instead of $grad_{wt}$, just needs to gradient in that form. It actually comes from some distribution depending on the point at which you are, right. The gradient would change, but it comes from a certain distribution and now, what you actually want at any time step when you are making this update, this particular update ok. Is it clear? Yeah when you are making this update, what would you

actually want? It should not move too much away from sorry.

So, now your gradients how you are computing say if you are doing the stochastic version, you are computing it for every point that you have, right. With respect to that point you would have some loss function and some derivative with respect to your parameters. If you move on to different point, you will have some different parameters. So, there is some randomness in this, ok. So, I am saying that these gradients would be treated as random variables which can take on values according to a certain distribution, ok

Now, what do I mean? So, what would I actually want when I am making an update. So, I have to make the basic choices. I could have just use $\text{grad } w_t$ which is the derivative with respect to the current time step. Add the current time step, ok. Instead of that I know why I am not happy with that because it has this problem that it could pull me to the extreme. So, at this point is actually saying change it, change your w value in a particular way which is more suited to me some other point would say something else. So, what we want is that whatever update we make should be very close to the dash of the distribution mean of the distribution, right and instead of computing the mean, we are computing a moving average and exponentially moving average, ok.

So, now what do we actually want to say I said that g_t is the random variable for denoting the gradient, ok. What do I actually want? I want the expected value of m_t should be equal to what the true expected value of g_t . This is what I want because I want to I do not want my updates to move in the extreme. It should be closer to the average to the mean of the distribution. Do you agree that this is my wish list? This is what something that I should desire for, ok. Now, let see what is m_t actually if I want to write it as a formula.

(Refer Slide Time: 35:28)

$$m_t = \beta m_{t-1} + (1-\beta) g_t \Rightarrow \nabla w_t$$

$$m_0 = 0$$

$$m_1 = (1-\beta) g_1$$

$$m_2 = \beta(1-\beta) g_1 + (1-\beta) g_2$$

$$m_3 = \beta^2(1-\beta) g_1 + \beta(1-\beta) g_2 + (1-\beta) g_3$$

$$E[m_t] = E\left[(1-\beta) \sum_{i=1}^t \beta^{t-i} g_i\right]$$

$$= E[g_t] \cdot (1-\beta) \sum_{i=1}^t \beta^{t-i}$$

$$E[m_t] = (1-\beta) E[g_t]$$

$$E[m_t] = E[g_t]$$

So, I have m_t is equal to $1 - \beta$. I will call this as g_t , right. So, remember the g_t is grad of w_t , ok. So, now let us try to write formula for this. So, m_0 , I will set it to 0. So, m_1 is going to be $1 - \beta$ into g_1 , ok. m_2 is going to be β into $1 - \beta$ into g_1 plus $1 - \beta$ into g_2 and m_3 is going to be β into $1 - \beta$ square g_1 plus β into $1 - \beta$.

Student: Beta square.

Sorry beta square.

Student: Minus beta.

Minus beta, wait is the first term correct.

Student: Yes.

No beta, ok. So, wait what am I? Oh beta is getting multiplied, to g_2 plus $1 - \beta$ into g_3 . So, what is the general formula going to be? It is m_t is equal to, so $1 - \beta$ can come out, ok. Summation i is equal to 1 to t $1 - \beta$.

Student: Beta square.

Oh god sorry, beta raise to $3 - i$ and g_i , right ok. So, this is what my m_t is, ok. Now, let me take the expectation of this, ok. Now, this is β $1 - \beta$. Now, this is going to

be is that fine. So, what is this? This is an ap gp. What is the sum going to be?

So, it is going to be 1 over 1 minus. Oh it is actually sorry 1 minus β raise to t over 1 minus β . Is that fine? So, what will happen is, this will get cancelled and what you are left with this 1 minus β raise to t into e of gt , ok. So, what is the relation that you have E of mt , ok. E of mt is equal to 1 minus β raise to t into E of gt . What did you actually want?

Student: E of gt .

Right so, now how will you ensure that divide by divide mt by 1 minus β raise to t and that exactly the bias correction that we have done, ok. Sorry about this messy derivation, but I guess most of you get it. If not, we will just type it properly and upload it in the slides. How many of you got this? Most of you got, fine; so, that is the similar derivation for vt also, fine. So, that is why we need the bias correction.