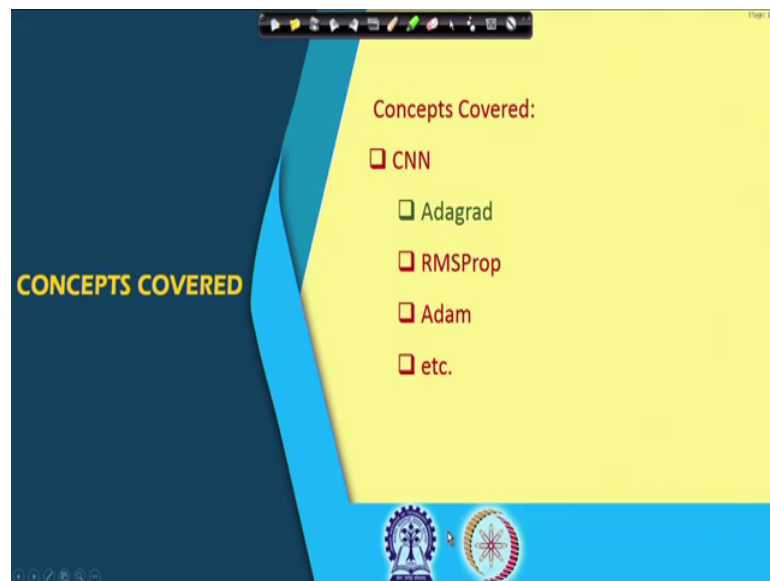


**Deep Learning**  
**Prof. Prabir Kumar Biswas**  
**Department of Electronics and Electrical Communication Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 45**  
**Optimisers: RMSProp, AdaDelta and Adam Optimiser**

Hello welcome back to the NPTEL online certification course on Deep Learning. For last few lectures we are discussing about various techniques for improvement of gradient descent algorithm or gradient descent algorithm is nothing, but the algorithm of training of the neural network or deep neural network that we are talking about. So, so far we have discussed about the momentum optimization technique which optimizes the gradient descent algorithm, we have also talked about the Nesterov accelerated gradient based optimization technique and we have also talked about the Adagrad. So, in our previous class we have talked about the Adagrad algorithm in particular.

(Refer Slide Time: 01:13)



So, in today's class we will talk about two more algorithms, one of them is RMSProp and the other one is Adam and we will also see a very closely related algorithm which is very closely related to RMSProp.

(Refer Slide Time: 01:33)

Adagrad

$$g_t = \frac{1}{n} \sum_{X \in \text{Minibatch}} \nabla_W L(W_t, X) \quad r_t = \sum_{\tau=1}^t g_\tau \circ g_\tau$$
$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\epsilon + r_t}} \circ g_t \leftarrow \frac{1}{\sqrt{\epsilon + r_t^{(i)}}} \cdot g_t^{(i)}$$

$\circ \rightarrow$  element-wise product

So, as we have seen in the previous class the Adagrad algorithm is given something like this, that at time  $t$  you compute the batch gradient; gradient of the loss function with respect to the weight vector or with respect to the parameter vector. And, then what you do is, you go on accumulating the squared gradient or you take the sum of squared gradient of all the historical gradient values and this sum of squared gradient is used to scale the gradient of all individual locations.

So, as a result our upgradation or weight upgradation algorithm that we have seen earlier is given by this expression, where if  $W_t$  is the parameter vector or weight vector at time  $t$ . Then at time  $t$  plus 1 we get our updated weight vector as  $W_t$  minus eta upon square root of epsilon I plus  $r_t$  times  $g_t$  where, these operations are actually done element wise; that means, whenever I rewrite 1 over squared root of epsilon I plus  $r_t$ . So, for individual components this will actually be 1 over square root of epsilon plus  $r_t^{(i)}$ . So, this is the sum of the squared gradient; sum of the squares of the  $i$ th component of the gradient vector and you take the square root of this and times  $g_t$ ; that means, this will be multiplied by corresponding  $i$ th component of the gradient  $g_t$  and that will be added with the  $i$ th component of  $W_t$ .

(Refer Slide Time: 03:40)

Adagrad

$$g_t = \frac{1}{n} \sum_{X \in \text{Minibatch}} \nabla_W L(W_t, X) \quad r_t = \sum_{\tau=1}^t g_\tau \circ g_\tau$$
$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\epsilon + r_t}} \circ g_t$$

$\circ \rightarrow$  element-wise product

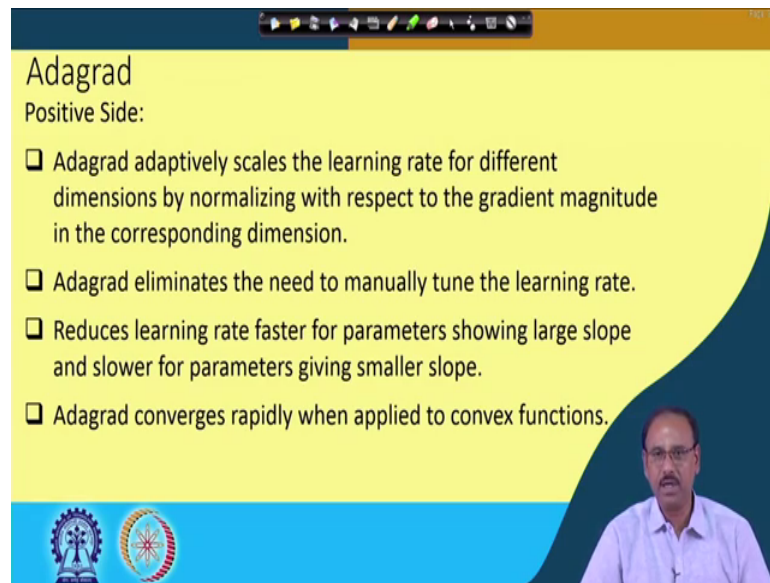
Handwritten notes:  $W_{t+1}^{(i)} = W_t^{(i)} - \frac{\eta}{\sqrt{\epsilon + r_t^{(i)}}} g_t^{(i)}$

So in fact, your expression will be that if I go for component wise  $W_{t+1}$  ith component will get  $W_t$  ith component of this minus eta by square root of epsilon plus  $r_t$  times  $g_t$ . So, this is the component wise operation that you perform, and here you find that you are basically scaling the ith component of the gradient vector by square root of sum of squares of the corresponding component of the gradient vector. And, what you are taking the summation over all the previous the gradient values or the partial derivative values that has been computed. So, this is sum of all the historical values.

And, this is what is actually scaling the ith component of the gradient vector which effectively is updating the ith component of the weight vector or the parameter vector. So, what is the effect of the scaling is that if for certain component say for  $W_i$  you find that  $\frac{\partial L}{\partial W_i}$  which is nothing, but our  $g_t$ . So, if this  $\frac{\partial L}{\partial W_i}$  is very large then this corresponding  $r_t$  will also be very large. So, as a result  $1/\sqrt{r_t}$  which will be small and that is making your learning rate of the corresponding ith component of the parameter to be small. And, if this is small;  $r_t$  is small and then the corresponding learning rate of the ith component of the parameter vector will also be large.

So, this is how Adagrad algorithm is adaptively tuning the learning rate of individual parameters or individual weight components of the weight vector.

(Refer Slide Time: 05:56)



The slide is titled "Adagrad" and lists the following positive aspects:

- ❑ Adagrad adaptively scales the learning rate for different dimensions by normalizing with respect to the gradient magnitude in the corresponding dimension.
- ❑ Adagrad eliminates the need to manually tune the learning rate.
- ❑ Reduces learning rate faster for parameters showing large slope and slower for parameters giving smaller slope.
- ❑ Adagrad converges rapidly when applied to convex functions.

The slide also features a video inset of a man speaking in the bottom right corner and two logos in the bottom left corner.

So, we have also seen that what are the positive points of this Adagrad algorithm. So, the first positive point is it adaptively scales the learning rate for different dimensions by normalizing with respect to the gradient magnitude in the corresponding direction. So, if the gradient magnitude in a particular direction is more the learning rate in that direction will reduce or as if the gradient component in a particular direction is small, the learning rate in that particular direction will not reduce that much. So, that is how you adaptively tune the learning rate in different directions.

And you do not have to manually set any of the hyper parameters. So, initially we have the initial learning rate given by  $\eta$ , but once we fix that is final; we need not update it anymore. And, as a result the learning it becomes faster and you quickly converts to the minima of the error function. So, these are the positive points of the Adagrad algorithm.

(Refer Slide Time: 05:56)

Adagrad

Negative side:

- ❑ If the function is non-convex:- trajectory may pass through many complex terrains eventually arriving at a locally region.
- ❑ By then learning rate may become too small due to the accumulation of gradients from the beginning of training.
- ❑ So at some point the model may stop learning.

And, the negative point is if the function is non-convex which is quite possible given your high dimensional space in which the error function is defined, then while the algorithm proceeds the trajectory of the weight vector or the parameter vector may pass through many complex terrains before coming to a locally convex region. So, the moment it comes to locally convex region, we want that the algorithm will quickly converge to the minima of this locally convex region.

But, because the learning rate is being gradually reduced it is being scaled by square root of the sum of the squares of the corresponding partial derivative. And, this summation is taken from  $t$  equal to 0, it is actually accumulative and this being sum of the squares it always goes on increasing, it does not reduce at all. So, as number of iterations proceeds your scaling factor that is  $1$  upon square root of  $\epsilon + r$  that will go on increasing. And, as a result your learning rate may become very very small with time and in some cases the machine may stop learning at all. So, that is the problem of Adagrad algorithm.

So, let us see that how the other suggested algorithms take care of or try to solve this problem which is given by Adagrad. So, the algorithm that we will talk about is what is RMSProp which tries to address this problem of Adagrad algorithm that is vanishing learning rate as the time increases as the number of iteration proceeds.

(Refer Slide Time: 08:44)

**RMSProp**

- ❑ RMSProp uses exponentially decaying average of squared gradient and discards history from the extreme past.
- ❑ Converges rapidly once it finds a locally convex bowl.
- ❑ Treats this as an instance of Adagrad algorithm initialized within that bowl.

$$\frac{1}{\sqrt{\epsilon + r_t^{(i)}}}$$
$$r_t^{(i)} = \sum_{\tau=1}^t [g_{\tau}^{(i)}]^2$$

So, what is this RMSProp algorithm does is instead of taking the accumulative sum of squares of the gradients of the sum of the squares of the past and gradients or this past gradient starts from time  $t$  equal to 0. So, your basically the operation that was done in Adagrad algorithm is  $r_t$ , the scaling factor which is  $1$  upon square root of epsilon plus  $r_t$ . So, if you go for component wise this  $r_t$  is nothing, but sum of  $g_{\tau}^{(i)}$  square of this or let me put it as  $g_{\tau}^2$  instead of  $g_{\tau} g_{\tau}$  square and you take the summation of  $\tau$  is equal to say  $1$  to  $t$ . So, you find that this being a square term and which you are going on adding. So,  $r_t$  goes on increasing, it monotonically increases (Refer Time: 09:57), it does not reduce.

So, that is the problem with the Adagrad algorithm and in case of RMSProp instead of using this cumulative or acumulative sum of squared gradients the RMSProp takes exponentially decaying average of the squared gradient. And, it does not consider the extreme past histories while accumulating the sum of square gradient. And, as a result of this the algorithm converges rapidly, once it reaches once your vector reaches locally convex ball short of surface that (Refer Time: 10:41). And, what you can do is we can assume this point once it reaches that locally convex error surface, that as if your Adagrad algorithm is initialized at that point within that locally convex ball.

So, that is what the RMSProp does, RMSProp does not take the accumulative sum of squared gradients from the very beginning, but it takes an exponentially decaying

average of squared gradient. Now, let us see that what is this exponentially decaying squared gradient.

(Refer Slide Time: 11:34)

$$s_1 \ s_2 \ s_3 \ s_4 \ \dots \ s_i \ s_{i+1} \ \dots$$

$$v_t = \beta v_{t-1} + (1 - \beta) s_t$$

$$v_0 = 0$$

$$v_1 = \beta v_0 + (1 - \beta) s_1 = (1 - \beta) s_1$$

$$v_2 = \beta v_1 + (1 - \beta) s_2 = \beta (1 - \beta) s_1 + (1 - \beta) s_2$$

$$v_3 = \beta v_2 + (1 - \beta) s_3 = \beta^2 (1 - \beta) s_1 + \beta (1 - \beta) s_2 + (1 - \beta) s_3$$

$$\beta = 0.9$$

$$1 - \beta = 0.1$$

It is something like this, the suppose we have a sequence of numbers or sequence of samples given by say  $s_1 \ s_2 \ s_3 \ s_4$ , it goes on  $s_i \ s_{i+1}$  and it goes on like this. So, it goes on with time  $t$ . So, the exponentially decaying average, exponentially decaying average is defined like this; I want to define say  $v_t$  at time instant  $t$  which is the exponential decaying evidence. This will be same as  $\beta v_{t-1}$ , where  $v_{t-1}$  was the exponentially decaying average at time  $t-1$  plus  $(1 - \beta) s_t$  and this is initialized to say  $v_0$  is equal to 0. So, you start with this initialization.

So, once I initially  $v_0$  is equal to 0; that means,  $v_1$  will be  $\beta v_0 + (1 - \beta) s_1$  and because  $v_0$  is equal to 0; so, effectively you get  $v_1$  is equal to  $(1 - \beta) s_1$ ; at time  $t$  equal to 2 I get  $v_2$  which is nothing, but  $\beta (1 - \beta) s_1 + (1 - \beta) s_2$  and this  $v_1$  is nothing, but  $(1 - \beta) s_1$ . So, this will simply be  $\beta (1 - \beta) s_1 + (1 - \beta) s_2$ , similarly  $v_3$  will be  $\beta^2 (1 - \beta) s_1 + \beta (1 - \beta) s_2 + (1 - \beta) s_3$ . So, you find that in this term  $\beta v_{t-1}$  this term is weighted, I mean I have an weighting component which is  $\beta$  into  $\beta^2$  into  $\beta^3$  and so on.

And, then beta is actually less than one a typical value of beta is taken to be 0.9 so, 1 minus beta will be 0.1. So, all the previous sample values as I go as I compute say s 1 s 2 s 3 and so on, that will go on reducing exponentially. And, that is the advantage of taking the exponentially decaying average of the square gradients and this is what is used in case of RMSProp problem. So in case of RMSProp, the scaling factor is not the cumulative sum of gradient histories, but it is the exponentially decaying average of the squared gradients.

(Refer Slide Time: 15:01)

The slide titled "RMSProp" contains the following mathematical expressions:

$$g_t = \frac{1}{n} \sum_{\forall X \in \text{Minibatch}} \nabla_w L(W_t, X)$$

An arrow points from the above equation to the next one:

$$r_t = \beta r_{t-1} + (1 - \beta) g_t \circ g_t \rightarrow \text{Exponentially decaying average}$$

Below this, the weight update rule is shown:

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\epsilon + r_t}} \circ g_t$$

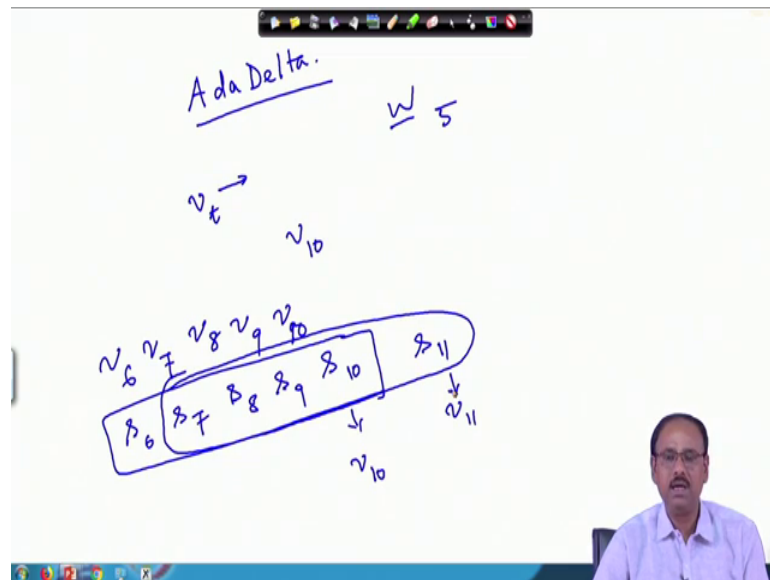
So, if I go to the updation algorithm is in RMSProp, the updation algorithm will be like this; you will find that you will compute the gradient in the same way as we have done in case of Adagrad, right. So, this is the gradient computed over a batch and then in order to compute  $r_t$  this is not the sum of the previous squared gradients, but it is an exponentially decaying average of the squared gradients ok. So, this  $g_t$  into  $g_t$  this gives you the squared gradient which is initialized to 0. So, this expression that  $r_t$  is equal to beta times  $r_{t-1}$  plus  $(1 - \beta)$  times  $g_t$  into  $g_t$ , this gives you the exponentially decaying average of  $g_t$  square.

And, then you are update algorithm that is  $W_{t+1}$  is equal to or  $W_t$  plus 1 is equal to  $W_t$  minus eta by square root of epsilon plus  $r_t$  plus  $g_t$  this algorithm remains the same as in case of Adagrad. So, only difference is this  $r_t$  is not the cumulative sum of squared gradients, but this  $r_t$  is the exponentially decaying average of



the squared gradients. And we said that there is a very closely related algorithm very closely related to this RMSProp; so, that closely related algorithm is what is known as AdaDelta.

(Refer Slide Time: 16:44)



So, we have a closely related algorithm known as AdaDelta. So, what AdaDelta does is in case of RMSProp you are taking the exponentially decaying average of the squared gradient; AdaDelta instead of taking the exponentially decaying average of squared gradient it computes the moving window average. So, you can take a more window size of say  $W$ . So, when you compute  $v_t$ ,  $v_t$  is computed over a past window size of  $W$ . So, if I take the window size  $W$  is equal to say 5, in that case in order to compute say  $v_{10}$  it will take the first 5; that means, it will take  $v_{10}$   $v_9$   $v_8$   $v_7$  and  $v_6$  or the say  $s_6$   $s_7$   $s_8$   $s_9$  and  $s_{10}$ . So, the average will be computed over this window of 5.

Similarly, when you compute  $s_{11}$ , this is what is your  $v_{10}$ , when you compute  $v_{11}$ , I have  $s_{11}$  which is the next sample, the average will be taken over  $s_7$   $s_8$   $s_9$   $s_{10}$  and  $s_{11}$  that will give you the average of 11. So, you are computing average over past samples which are within this window size of  $W$ . So, this is what is moving window average. So, you find that this RMSProp which takes exponentially decaying average, the AdaDelta takes a moving window average of the squared gradients. So, that is the only difference between RMSProp and AdaDelta. And in fact, both these algorithms were

proposed almost simultaneously, but independently, and both of them gives almost similar performance.

(Refer Slide Time: 18:59)

RMSProp with Nesterov Momentum

$$\tilde{W}_t = W_t + \alpha v$$

$$g_t = \frac{1}{n} \sum_{X \in \text{Minibatch}} \nabla_W L(\tilde{W}, X)$$

$$r_t = \beta r_{t-1} + (1 - \beta) g_t \circ g_t$$

$$v_{t+1} = \alpha v_t - \frac{\eta}{\sqrt{\epsilon + r_t}} \circ g_t$$

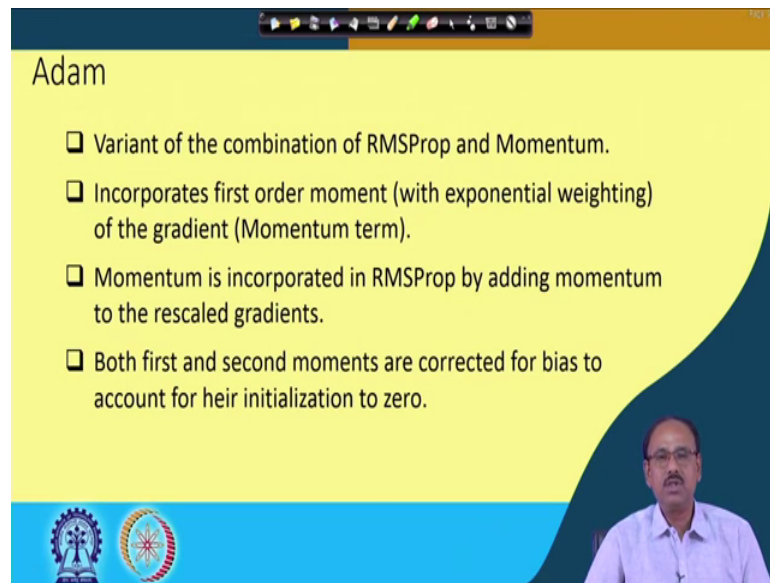
$$W_{t+1} = W_t + v_t$$

So, this is what is you are RMSProp algorithm, you can also improvise upon RMSProp algorithm with an Nesterov of momentum term. So, as we have seen that in case of Nesterov of accelerated gradient technique you take the gradient not at location  $W_t$ , but you take the gradient at a loop ahead position. So, assuming that your previous momentum term is  $v$ , you are looking ahead at location  $W_t$  plus alpha times  $v$  and you are taking the gradient at that location  $W_t$  plus alpha times  $v$  instead of computing the gradient at location  $W_t$ .

So, that is what your Nesterov accelerated gradient is. So, you are instead of computing the gradient at location  $W_t$ , if I take the gradient at location  $W_{\Delta}$  where,  $W_{\Delta}$  is the position or to loop ahead position. And, rest of the algorithms remains the same; so, it is as before you are taking the exponentially decaying average of the sum of squared gradients, exponentially decaying average of the squared gradients and then your update rule remains as before ok. So, this is what is RMSProp, which is part that improvised with Nesterov of momentum.

So, given this algorithm now the other algorithm that we said that we will be talking about is what is known as Adam or adaptive moments.

(Refer Slide Time: 20:45)



Adam

- ❑ Variant of the combination of RMSProp and Momentum.
- ❑ Incorporates first order moment (with exponential weighting) of the gradient (Momentum term).
- ❑ Momentum is incorporated in RMSProp by adding momentum to the rescaled gradients.
- ❑ Both first and second moments are corrected for bias to account for their initialization to zero.

So, what is this Adam algorithm? Adam algorithm you can consider this to be variant of the combination of RMSProp and momentum. So, in case of RMSProp we did not have any concept of momentum. So, here we can incorporate both the first order momentum and the second order momentum. Second order momentum is nothing, but the sum of squared gradients or exponentially decaying average of the squared gradients as in case of RMSProp which is used for scaling the learning rate in individual directions. And, along with that if we add the momentum term, where the momentum will be scaled according to the square root of the exponentially decaying average.

So, if I use both this first order momentum and second order momentum because sum of squared gradients is nothing, but average of the squared gradients is nothing, but equivalent to your second order momentum. So, you use both this first order and second order moment that becomes a variant of RMSProp and this is what is known as Adam.

So, in case of Adam you are including both first and second moments for weight updation or parameter updation and in addition you Adam incorporates one more term, that it tries to correct the bias to account for initial to zero. So, what he said is that when you are taking exponentially decaying average, you are initializing the average value at zero at  $t$  equal to zero, right.

(Refer Slide Time: 22:41)

The whiteboard contains the following handwritten text:

$$v_t = \beta v_{t-1} + (1-\beta) s_t$$
$$v_0 = 0.$$
$$\left(\frac{\partial L}{\partial W}\right)^2$$

A presenter is visible in the bottom right corner of the whiteboard frame.

So, for this exponentially weighting average what we did is we have computed  $v_t$  is equal to some beta times  $v_{t-1}$  plus  $(1-\beta)$  times  $s_t$ . So, when you go for exponentially decaying average of the squared gradients this  $s_t$  is nothing, but your gradient  $\frac{\partial L}{\partial W}$  square of this. So, here what we are doing is you initialize  $v_0$  to 0; so, as a result all the computations that you are performing; particularly the computations at the initial sample levels that equal to  $t=1$  and  $t=2$  and  $t=3$  and so on; all of them are actually biased toward 0. So, in order to avoid this bias what this adam algorithm does is, it goes for correction of the bias term.

(Refer Slide Time: 23:36)

The slide is titled "Adam" and contains the following content:

$$g_t = \frac{1}{n} \sum_{X \in \text{MiniBatch}} \nabla_w L(W_t, X)$$

Biased first and second moments

$$s_t = \beta_1 s_{t-1} + (1 - \beta_1) g_t$$
$$r_t = \beta_2 r_{t-1} + (1 - \beta_2) g_t \circ g_t$$

At the bottom left, there are two logos: one of a gear and a person, and another of a gear and a sun. A presenter is visible in the bottom right corner of the slide frame.

And, the correction is done by dividing the computed values or the exponentially decaying average values by 1 minus beta. So, the operation is something like this. So, as before you compute the gradient over a batch at time t which is  $g_t$  then you compute the first and or exponentially decaying average of the first and second moment of  $g_t$ . So, the exponentially decaying average of the first moment of  $g_t$  is given by  $s_t$  is equal to some  $\beta_1$  times  $s_{t-1}$  plus  $(1 - \beta_1)$  times  $g_t$ .

And, the second moment exponentially decaying average of that is given by  $r_t$  is equal to some  $\beta_2$  times  $r_{t-1}$  plus  $(1 - \beta_2)$  times  $g_t^2$  which in this case is written as  $g_t$  into  $g_t$ . And, we have said that this particular symbol small square this is used to represent element wise multiplication. And, as we said before that this exponentially decaying average that you have computed, it is biased towards 0 because at  $t$  equal to 0 both  $s_t$  and  $r_t$  that is  $s_0$  and  $r_0$  were initialize to 0.

(Refer Slide Time: 25:07)

Adam

Bias corrected first and second moments

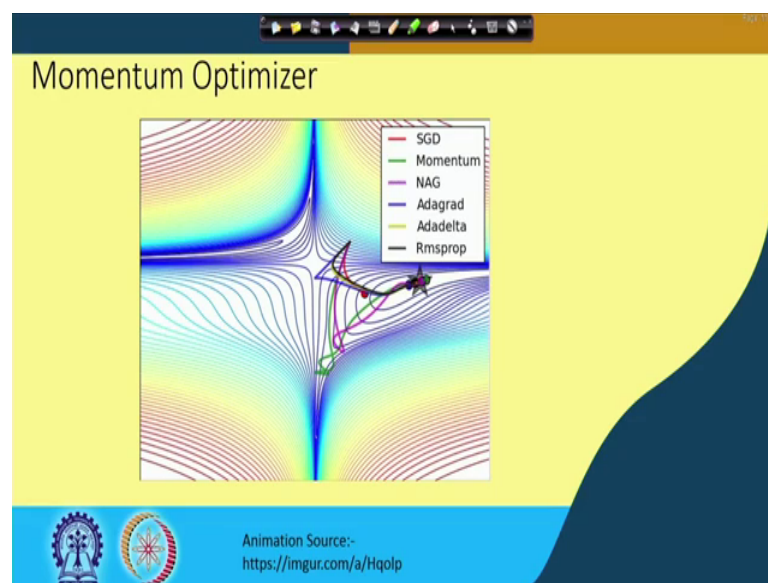
$$\hat{s}_t = \frac{s_t}{1 - \beta_1} \quad \hat{r}_t = \frac{r_t}{1 - \beta_2}$$

$$W_{t+1} = W_t - \eta \frac{\hat{s}_t}{\sqrt{\epsilon + \hat{r}_t}}$$

So, in order to take care of this bias the Adam algorithm goes for bias correction and for bias correction it takes  $\hat{s}_t$  which is the bias corrected first moment which is  $s_t$  upon  $(1 - \beta_1)$  ok, but the computation of was  $s_t$  like this,  $s_t$  was  $\beta_1$  times  $s_{t-1}$  plus  $(1 - \beta_1)$  times  $s_t$ . So, after correction it becomes  $s_t$  upon  $(1 - \beta_1)$ . So, this corrected first moment is represented as  $\hat{s}_t$ , similarly the corrected second moment which is  $\hat{r}_t$  is nothing, but  $r_t$  by  $(1 - \beta_2)$ .

So, once given this your weight updation of the parameter updation rule simply becomes  $W_{t+1}$  is equal to  $W_t$  minus  $\eta$  times  $\hat{s}_t$ , where  $\hat{s}_t$  is the bias corrected first moment upon square root of  $\epsilon$  I plus  $\hat{r}_t$ , where  $\hat{r}_t$  is the bias corrected second moment. So, you find that this is nothing, but similar to your RMSProp algorithm where you are incorporating where, this Adam algorithm incorporates bias correction operation and it also incorporates the first moment in the update step. So, this is  $\eta$  times  $\hat{s}_t$  where,  $\hat{s}_t$  is the first moment of the gradients. So, this is your Adam optimizer which optimizes the gradient descent operation.

(Refer Slide Time: 26:59)



So, now we can compare with this animation, you can see that I can compare the relative performance of different optimization algorithms. So, as you see over here this red curve, the red curve is actually the pure SGD algorithm or Stochastic Gradient Algorithm, the blue one gives you the momentum. Then you have NAG Nesterov accelerated gradient operation, then you have Adagrad, then you have a Adadelta, then you have RMSProp. So, you find over here that the SGD which you find that it is still diver converging whereas, the other algorithms have already come first.

And as we have seen before that your momentum and the NAG as we said that, NAG that is Nesterov of accelerated gradient gives slight improvement over momentum, right. So, you find that these two green and the pink one they represent a momentum and NAG optimization and they are very close.

So, we have discussed about the various optimization techniques, where this optimization techniques tries to improve the learning rate given by gradient descent algorithm. So, we will stop here today and will talk about how the other challenges that you get in gradient descent or the learning algorithms of deep neural networks are addressed by different approaches in our future lectures.

Thank you.