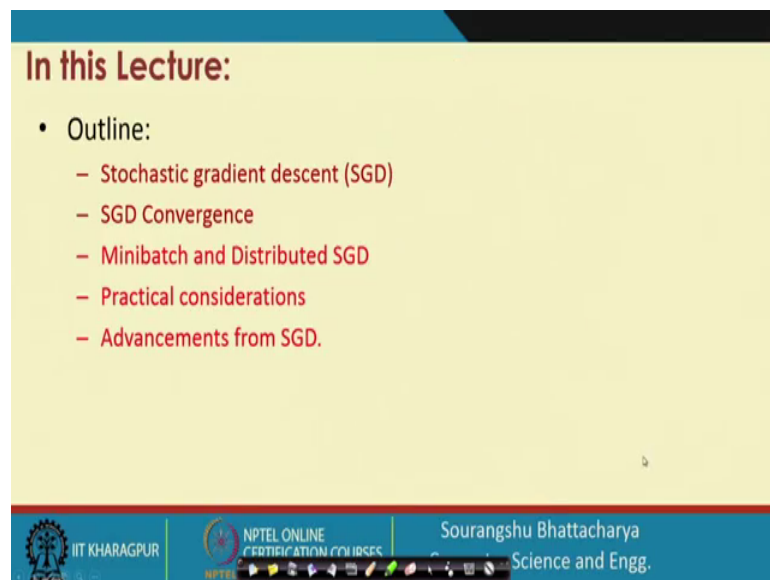


Scalable Data Science
Prof. Sourangshu Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 21b
Stochastic Optimization (Contd.)

Hello everyone. Welcome to the 21st lecture of NPTEL course on Scalable Data Science. I am Professor Sourangshu Bhattacharya from Computer Science and Engineering Department at IIT, Kharagpur. And today, we are going to discuss about Stochastic Optimization.

(Refer Slide Time: 00:37)



In this Lecture:

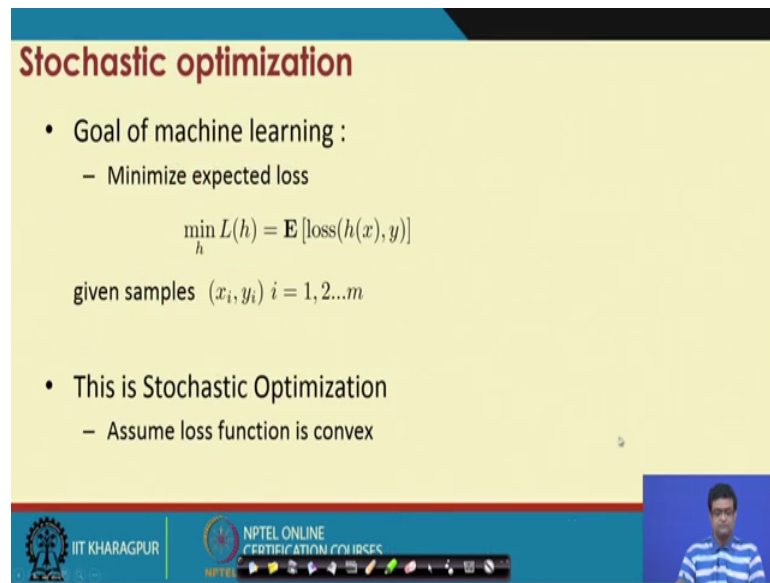
- Outline:
 - Stochastic gradient descent (SGD)
 - SGD Convergence
 - Minibatch and Distributed SGD
 - Practical considerations
 - Advancements from SGD.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Sourangshu Bhattacharya
Science and Engg.

So, in the first part of this lecture, we have discussed the stochastic optimization problem. And we have discussed the stochastic gradient descent algorithm. And we have described proof for the stochastic gradient descent algorithm. As to why and in what sense does the stochastic gradient descent algorithm converge.

So, in this class, we are going to discuss about the practical aspects; like we are going to discuss the mini-batch SGD, and the distributed SGD. And we are going to discuss some other practical considerations while implementing SGD. And then very importantly, we are going to discuss some very recent advancements on stochastic gradient descent. Many of which are have actually led to a significant improvement in running time, and they have also arguably led to the revolution in you know deep learning community.

(Refer Slide Time: 01:57)



Stochastic optimization

- Goal of machine learning :
 - Minimize expected loss

$$\min_h L(h) = \mathbb{E} [\text{loss}(h(x), y)]$$

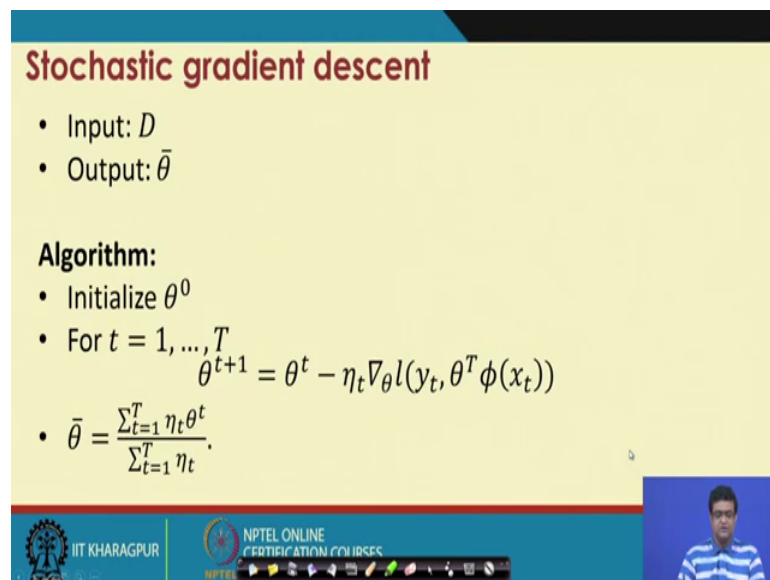
given samples $(x_i, y_i) \quad i = 1, 2, \dots, m$

- This is Stochastic Optimization
 - Assume loss function is convex

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this we have already discussed. So, machine learning is a stochastic optimization problem.

(Refer Slide Time: 02:06)



Stochastic gradient descent

- Input: D
- Output: $\bar{\theta}$

Algorithm:

- Initialize θ^0
- For $t = 1, \dots, T$
$$\theta^{t+1} = \theta^t - \eta_t \nabla_{\theta} l(y_t, \theta^T \phi(x_t))$$
- $$\bar{\theta} = \frac{\sum_{t=1}^T \eta_t \theta^t}{\sum_{t=1}^T \eta_t}$$


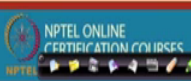

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And you can solve it with stochastic gradient descent algorithm, which is this algorithm.

(Refer Slide Time: 02:15)

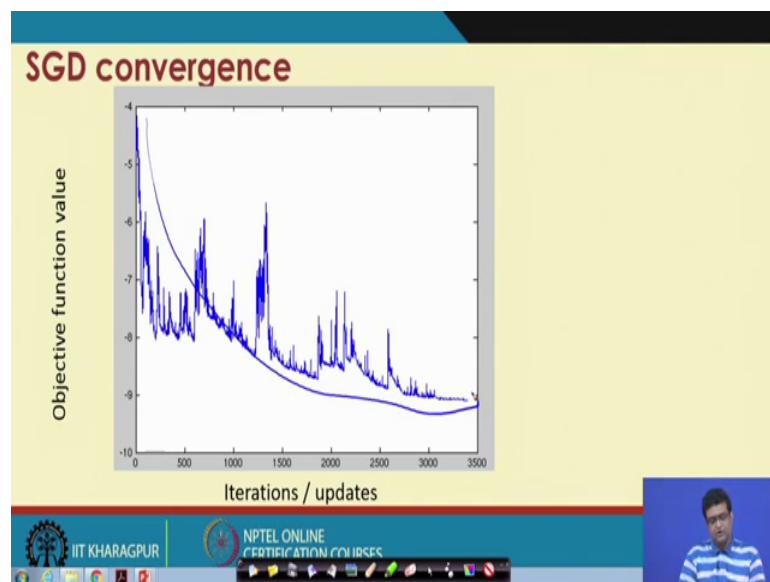
SGD convergence

- Expected loss: $s(\theta) = E_{\mathcal{P}}[l(y, \theta^T \phi(x))]$
- Optimal Expected loss: $s^* = s(\theta^*) = \min_{\theta} s(\theta)$
- Convergence:
$$E_{\bar{\theta}}[s(\bar{\theta})] - s^* \leq \frac{R^2 + L^2 \sum_{t=1}^T \eta_t^2}{2 \sum_{t=1}^T \eta_t}$$
- Where: $R = \|\theta^0 - \theta^*\|$
- $L = \max \nabla l(y, \theta^T \phi(x))$



And we know that the algorithm converges or the rather the loss function of the parameter or the loss of the parameter converges in the expected sense as you increase the number of updates or epochs.

(Refer Slide Time: 02:38)



And this is a typical picture of convergence. So, even though normally you would look at it and say that this is really a very oscillating sequence, but actually a in expectation. It is converging something like this, it is going something like this. So, this is indeed how stochastic gradient descent works ok.

(Refer Slide Time: 03:10)

SGD - Issues

- Convergence very sensitive to learning rate (η) (oscillations near solution due to probabilistic nature of sampling)
 - Might need to decrease with time to ensure the algorithm converges eventually
- Basically – SGD good for machine learning with large data sets!

Handwritten notes on the right side of the slide:

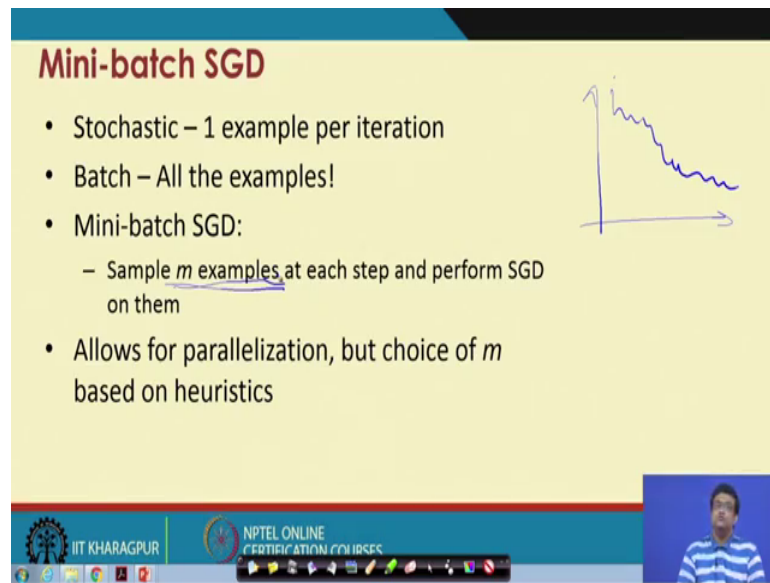
- $\sum_{t=1}^{\infty} \eta_t^2 < \infty$
- $\sum_{t=1}^{\infty} \eta_t \rightarrow \infty$
- $\eta_t = \frac{1}{\sqrt{t}}$

The slide footer includes the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

So, now what are the issues with stochastic gradient descent? So, the first issue is that the convergence of stochastic gradient descent is very sensitive to the learning rate or this η that you use ok. So, you remember that in the proof, we have given two conditions for η . So, the first condition is that summation over so summation over t is equal to 1 to infinity ηt^2 should be finite. So, should be less than infinity ok.

And summation over t is equal to 1 to infinity ηt should be tending to infinity ok. So, this is the condition that we have put. So, there may be, so there may be many types of sequences, which satisfies these conditions. So, for example so $1/t$ to the power let us say so $1/\sqrt{t}$ is one such sequence, which is that if you take η , so your sequence of ηt is something like this ok. So, this sequence satisfies these two conditions, you can check that this sequence satisfies these two conditions ok. So, the question is what type of convergence rate, do you choose.

(Refer Slide Time: 05:27)



Mini-batch SGD

- Stochastic – 1 example per iteration
- Batch – All the examples!
- Mini-batch SGD:
 - Sample m examples at each step and perform SGD on them
- Allows for parallelization, but choice of m based on heuristics

The slide includes a hand-drawn graph on the right side showing a noisy, downward-sloping curve, representing the loss function over time for a stochastic optimization process. The graph has a vertical y-axis and a horizontal x-axis. The curve starts at a high point and generally trends downwards with significant oscillations, eventually leveling off at a lower point.

The slide footer contains the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. A small video inset in the bottom right corner shows a man in a blue and white striped shirt.

So, we will come to that ok. Before we come to this so we would like to so describe what is called the mini-batch stochastic gradient descent. So, we have seen the batch gradient descent, where if you have let us say in examples, you choose all the examples in computing the gradients ok.

If you use the stochastic gradient descent algorithm that I have just described, you use one example per iteration or one example per update ok. So, you need not so these are two extremes, instead you can select maybe a certain constant number. So, you can select m examples m randomly selected examples to compute the gradient at each step or at each update. And then perform SGD ok.

So, this version of SGD is called the mini-batch SGD. So, this is the first and you can show that by similar arguments that this version of the SGD also converges ok. So, this is the first improvement and it is seen that the oscillations the so if you look at the curve of SGD, you see oscillations of this type. So, this oscillations become much less. So, with as the number of examples that you use in each mini-batch increases the amount of oscillation reduces a lot ok.

So, this is one simple, but very useful trick that people used to make SGD converge better ok. Another important outcome of this heuristic of using m examples for each update is that it allows for parallelization. So, you may choose these m examples from m

different servers. And then do an update or you may do update based on m different examples from a given server ok.

So, all of these things are possible. So, there are various ways like all reduce SGD, then there is the parameter server, which is like the centralized update, which we normally discuss. And then there is also some kind of round robin SGD where in a round robin fashion the parameters get updated in each of the servers ok. So, all these types of parallelization or distributed processing is possible ok.

(Refer Slide Time: 08:54)

Example: Text categorization

- **Example by Leon Bottou:** *2008*
 - Reuters RCV1 document corpus
 - Predict a category of a document
 - One vs. the rest classification
 - $n = 781,000$ training examples (documents)
 - 23,000 test examples
 - $d = 50,000$ features
 - One feature per word
 - Remove stop-words
 - Remove low frequency words

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, here is one concrete example, this is also historically one of the very important step. So, this is by Leon Bottou, it is a due to a paper in 2008 by Leon Bottou. And what he did was, so he was so this sort of led to a resurgence or renewed interest in this algorithm of stochastic gradient descent ok.

So, what he did is he took a large text classification corpus. So, you have 781,000 training examples, and 23,000 test examples. And you have about 50,000 features which is the number of words in the vocabulary that we are considered ok. And, then he is trying to so is the problem that he is trying to solve is to predict the category of a documents. So, there are a certain number of categories, and he is trying to predict the category of a document ok.

(Refer Slide Time: 10:07)

Example: Text categorization

- **Questions:**
 - (1) Is SGD successful at minimizing $f(w,b)$?
 - (2) How quickly does SGD find the min of $f(w,b)$?
 - (3) What is the error on a test set?

Handwritten notes: "loss minimization" with an arrow pointing to question (1). A bracket underlines questions (2) and (3).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And so, these are the questions basically that you want to ask, when you have implemented this SGD algorithm. So, is it successful at minimizing the objective function ok. How quickly does SGD find the minimum objective function? And then what is the error on a test set.

So, note that even though generally it is true that so the objective function in case of this kind of training is typically the loss on training set. So, loss on training set ok. This may or may not reduce the loss on test set ok, so that is another thing that depending on how well the model generalizes.




(Refer Slide Time: 11:09)

Example: Text categorization

- **Questions:**
 - (1) Is SGD successful at minimizing $f(w,b)$?
 - (2) How quickly does SGD find the min of $f(w,b)$?
 - (3) What is the error on a test set?

	Training time	Value of $f(w,b)$	Test error
Standard SVM	23,642 secs	0.2275	6.02%
Fast SVM	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	0.2275	6.02%

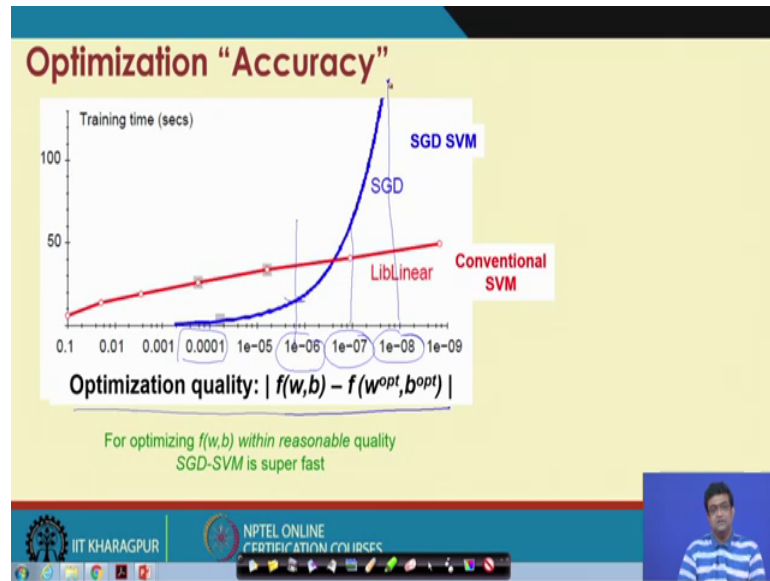
(1) SGD-SVM is successful at minimizing the value of $f(w,b)$
(2) SGD-SVM is super fast
(3) SGD-SVM test set error is comparable



So, these are the running times ok. So, you see that so at that time whatever was the standard SVM solver, that took 23,000 seconds. And there was a fast SVM solver, which took 66 seconds to solve. And all of them have and the SGD SVM that Leon Bottou implemented took only 1.4 seconds, this was a huge speed up ok.

And this speed up, so this speed up comes so, this is the value of the objective function and this is the test error. So, see that it comes at almost no compromise with either the value of the objective function or the test error ok. So, the answer was that all three metrics are better for SGD in solving SVM than the existing algorithm.

(Refer Slide Time: 12:29)

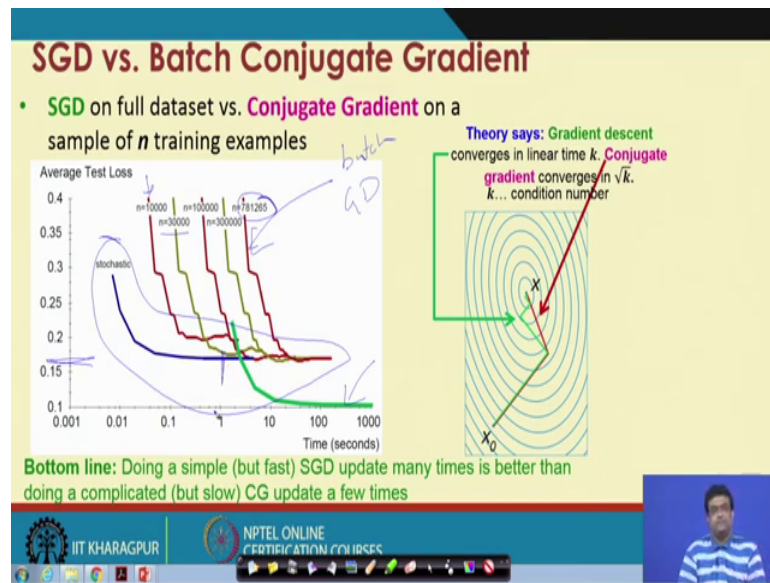


Now, this graph is on the x axis, it is showing how well the different algorithms either SGD or a conventional SVM algorithm is able to solve the optimization problem. So, x axis is plotting the optimization quality ok. So, this is the absolute difference between the objective function value that you achieve and the actual objective function value ok.

And on y axis is the amount of time taken to achieve this ok. So, what you see here is that if you want to get within let us say 10^{-4} of the actual objective function value, then it takes almost no time ok. If you want to get within 10^{-6} , maybe it takes around let us say 10 seconds, whereas the existing algorithm takes 30 seconds.

But, if you want very high precision, so if you want to go to 10^{-7} or 10^{-8} , then the time taken increases kind of exponentially ok. So, this increases exponentially, so it never almost, never reaches 10^{-8} . So, the point here is that if you want a reasonable quality solution, then SGD is good. Otherwise, you may want to go for other higher order method, which we will come to ok.

(Refer Slide Time: 14:36)



And this is a comparison between the SGD and batch conjugate gradient descent. So, basically the blue line here is full SGD ok, so the blue line here is full SGD. And these are the SGD for different mini-batch sizes. So, this is for mini-batch size of 10,000, this is mini-batch size of 30,000 and so on and so forth.

So, this is the mini-batch size of 781,000 which is something like gradient descent. So, this curve is for a batch gradient descent batch gradient descent ok. And this curve is for conjugate gradient descent, which is the second order method. So, the conjugate gradient descent is known to take very few steps ok. And now the x axis here is the time in second, and y axis is the loss on the test set ok.

So, as you can see that they achieve a reasonable loss, not a very low loss. So, this conjugate gradient descent achieves a very low loss, but stochastic gradient descent and different mini-batch versions of stochastic gradient descent achieve a reasonable loss within a small amount of time ok.

(Refer Slide Time: 16:20)

Practical Considerations

- Need to choose learning rate η and t_0

$$w_{t+1} \leftarrow w_t - \frac{\eta}{t + t_0} \left(w_t + C \frac{\partial L(x_t, y_t)}{\partial w} \right)$$

- Leon suggests:
 - Choose t_0 , so that the expected initial updates are comparable with the expected size of the weights
 - Choose η :
 - Select a **small subsample**
 - Try various rates η (e.g., 10, 1, 0.1, 0.01, ...)
 - Pick the one that most reduces the cost
 - Use η for next 100k iterations on the full dataset

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, like we were discussing, so we need to choose the learning rate η . So, there are roughly so there are two types of choices. So, one is so the learning rate is of this type ok. So, one option is the learning rate of this type ok, where the learning rate is of this form $\eta / (t + t_0)$. So, it is essentially of the form $1 / t$ ok.

So, and then the question remains that how you how to choose $\eta / (t + t_0)$ and t_0 ok. So, the suggestion is that you choose t_0 such that the expected initial updates are comparable with expected size of weights ok. So, basically the magnitude of this number and the magnitude of this update are roughly same ok.

And you choose η actually this should just be η , this should not be η / t . And you choose η such that the oscillations do not diverge or you take a small sub sample. And you try out various learning rates logarithmically something like 10, 1, 0.1, 0.01 and so on and so forth. And for whichever it does not diverge, and it reduces the cost you use that and then run on the full data set.

(Refer Slide Time: 18:27)

Practical Considerations

- **Sparse Linear SVM:**
 - **Feature vector x_i is sparse (contains many zeros)**
 - Do not do: $x_i = [0, 0, 0, 1, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, \dots]$
 - But represent x_i as a sparse vector $x_i = [(4, 1), (9, 5), \dots]$
 - **Can we do the SGD update more efficiently?**
$$w \leftarrow w - \eta \left(w + C \frac{\partial L(x_i, y_i)}{\partial w} \right)$$
 - **Approximated in 2 steps:**
$$w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$$
 cheap: x_i is sparse and so few coordinates j of w will be updated
$$w \leftarrow w(1 - \eta)$$
 expensive: w is not sparse, all coordinates need to be updated

Regulariser

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, the question comes that so this is the update that one has to do. Now, the update has two components. One is the existing parameter w , and other is the gradient. Now, for a large number of examples, what you can see is that this gradient is a sparse vector something like this. So, the vector has almost all the increases 0, and only a few entries which are non-zero.

So, you can store the vector something like this that the 4th entry is 1, and the 9th entry is 5 ok. And you can also transmit the vector like this. If this is the case, you can break the update into these two steps. So, the first step is that you update using only the loss part not the regulariser. So, this term comes because of the regulariser ok. So, you update it only with the loss part. And then you do further update of w as w times 1 minus eta. So, this will basically do this kind of an update. So, this is one practical way of speeding up the procedure.

(Refer Slide Time: 20:09)

Practical Considerations

- **Solution 1:** $w = s \cdot v$
 - Represent vector w as the product of scalar s and vector v
 - Then the update procedure is:
 - (1) $v = v - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
 - (2) $s = s(1 - \eta)$
- **Solution 2:**
 - Perform only step (1) for each training example
 - Perform step (2) with lower frequency and higher η

Two step update procedure:

- (1) $w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
- (2) $w \leftarrow w(1 - \eta)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the basically the update. So, you store v , and you store s . And then you perform you perform the step one, which is the first update every time, because this one depends on the examples and this changes. But, the second update is just an updation of the scalar. So, you perform this may be with much lower frequency.

(Refer Slide Time: 20:51)

Practical Considerations

- **Stopping criteria:**
 - How many iterations of SGD?**
 - **Early stopping with cross validation**
 - Create a validation set
 - Monitor cost function on the validation set
 - Stop when loss stops decreasing
 - **Early stopping**
 - Extract two disjoint subsamples **A** and **B** of training data
 - Train on **A**, stop by validating on **B**
 - Number of epochs is an estimate of k
 - Train for k epochs on the full dataset

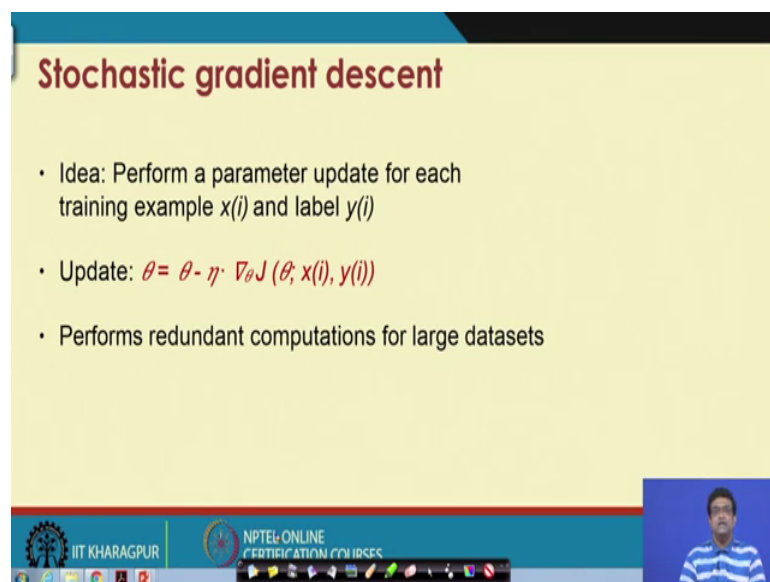
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next question comes that when you are training machine learning models with stochastic gradient descent, how do you stop. So, one way of stopping is that use you can do early stopping with a cross validation. So, you create a small validations set. And

instead of computing the loss on the entire training set, which is a very large set. Assuming that that is why you are using SGD in the first place.

So, you create a small validation set. You compute the cost function on the validation set and stop, when the loss stop decreasing ok. And then another way of doing the same thing is that you extract two disjoint subsamples A and B of training data. You train on A, and stop by validating on B. So, the number of epochs you need to you know complete the training on A, gives you an estimate of number of epochs. Then you train on the full data set using that many epochs, so let that be k. So, you train on the full data set using that many epochs.

(Refer Slide Time: 22:43)



The slide is titled "Stochastic gradient descent" in a bold, dark red font. It contains three bullet points: "Idea: Perform a parameter update for each training example $x(i)$ and label $y(i)$ ", "Update: $\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x(i), y(i))$ ", and "Performs redundant computations for large datasets". The slide is part of an NPTEL online course from IIT Kharagpur, as indicated by the logos at the bottom. A small video inset in the bottom right corner shows a man in a blue and white striped shirt.

- Idea: Perform a parameter update for each training example $x(i)$ and label $y(i)$
- Update: $\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x(i), y(i))$
- Performs redundant computations for large datasets

So, these are the practical considerations for implementing SGD in an efficient manner ok. Now, this is just this is the same SGD algorithm that I have described.

(Refer Slide Time: 23:03)

Momentum gradient descent

- Idea: Overcome ravine oscillations by momentum
- Update:

- $v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J(\theta)$
- $\theta = \theta - v_t$

SGD

SGD with momentum

Equipotential contours

The slide features two diagrams of concentric elliptical contours. The top diagram, labeled 'SGD', shows a jagged, oscillating path across the contours. The bottom diagram, labeled 'SGD with momentum', shows a smooth, straight path moving towards the center. Handwritten text 'Equipotential contours' with arrows points to the contours in both diagrams. The slide footer includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video inset of a speaker.

Now, one of the main problems of SGD is that let's say if you have so here in this figure, what we have discussed what we have describing are equipotential contours ok, so equipotential contours ok. So, basically all these lines here have the same value of the objective function. So, this is one way of visualizing a function ok.

Now, you see that the function is more or less flat in this direction. And the function is very steeply increasing or decreasing in this direction. Now, whenever you have a function with this kind of a property, what SGD does it is it oscillates a lot. This is also property of gradient descent, it not just a stochastic gradient descent ok. So, it oscillates a lot.

So, one way of stopping this is to use what is called a momentum term. So, let us say your objective function is J of θ ok. So, with normal SGD if for reference, if we go back with normal SGD, you have to so you just update, your parameter is now θ . And you update θ using η times gradient of J of θ at x_i and x_j .

Now, here your update is of this form that you use η times J of θ at the gradient of J of θ , but you also store the previous gradient or rather the previous update value, and you multiply that with γ ok. So, the whole update at time t is what is called v of t , which is the sort of a momentum update ok. So, momentum update, because you have a certain update coming from J of θ . And you have a certain existing update coming from the previous updates.

So, on the axis on which the updates are consistent; so, for example, on this axis the updates are consistent, it is always going towards the right ok. So, then on that axis the momentum will cause the update value to increase, whereas on the axis like this axis, where you are continuously overshooting the minimum ok, the update value will be low. So, this causes a lot of speed up of the stochastic gradient descent. So, this is one of the very important techniques of speeding up stochastic gradient descent. So, your update changes in this manner. So, every time instead of doing the previous update you do the momentum update.

(Refer Slide Time: 26:47)

Nesterov accelerated gradient

- Ideas:
 1. Big jump in the direction of the previous accumulated gradient & measure the gradient
 2. Then make a correction.
- Update:
 - $v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J(\theta - \gamma v_{t-1})$
 - $\theta = \theta - v_t$

The diagram illustrates the Nesterov accelerated gradient method. It shows a path starting from a point, moving in a direction (blue arrow), then making a correction (red arrow) to reach a point (green arrow). The path then continues in a direction (orange arrow) towards the minimum.

NPTEL ONLINE CERTIFICATION COURSES
IIT KHARAGPUR

Now, another so based on this momentum updates. Another very important type of update, which was discovered by Nesterov and this also has theoretical importance. But, this update is that you do sort of a prescient momentum update that is you know beforehand that you are going to do a momentum update. So, what you do is that you calculate the gradient at a point, where you would normally be after the momentum update.

So, instead of calculating the gradient where you are currently ok, you calculate you already know the momentum term. So, you calculate the gradient at a place where you would be, if you were to do update only based on the momentum, and then you use that as the corrected version of the gradient. So, instead of using the gradient you use the

corrected version of the gradient for updating ok. So, this is called the Nesterov accelerated gradient descent.

(Refer Slide Time: 28:30)

RMSprop

- Idea: Use the second moment of gradient vector to estimate the magnitude of update in a given direction.
- Update:

$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2$

$\Delta \theta = -\eta / \sqrt{E[g^2]_t + \epsilon} \odot g_t$

Handwritten notes on the slide include g_i and $\sqrt{E[g_i^2] + \epsilon}$.

The slide footer includes IIT KHARAGPUR, NPTEL ONLINE CERTIFICATE COURSES, and a small video feed of a presenter.

So, this was originally proposed by Nesterov and also proved by Nesterov to converge on convex functions. But, now people have also been using this in the deep learning setting for example for non-convex functions.

Now, another important update is to use the second moment of the gradient vector to estimate the magnitude of the update in a given direction. So, also sometimes called root mean square update, because what you do is you estimate the so for each so for each coordinate of the gradient. You estimate the, you try to estimate the square of the gradient in a momentum fashion.

So, you estimate the square of the gradient square of the current gradient and estimate the square of the existing gradient. And you use some fixed waiting of the two, you can use also any other waiting ok. So, the main idea here is that whatever gradient you have so if you have let us say the i th coordinate of the gradient, you divide it by square root of this expectation of g_i square or it is the same thing. So, you do it using expectation of square root of g_i square, and then in order to avoid division by 0 errors, you add epsilon.

So, this in some sense gives you a normalized update direction rather than so your update is not, very not very part term because of the magnitude of the gradient itself. And then

you do this update step. So, this is called this algorithm is called the RMS prop algorithm. This was first described by Geoffrey Hinton in one of the courses. So, many related algorithm have come up like Adadelta and Adagrad.

(Refer Slide Time: 31:29)

ADAM (Adaptive moment)

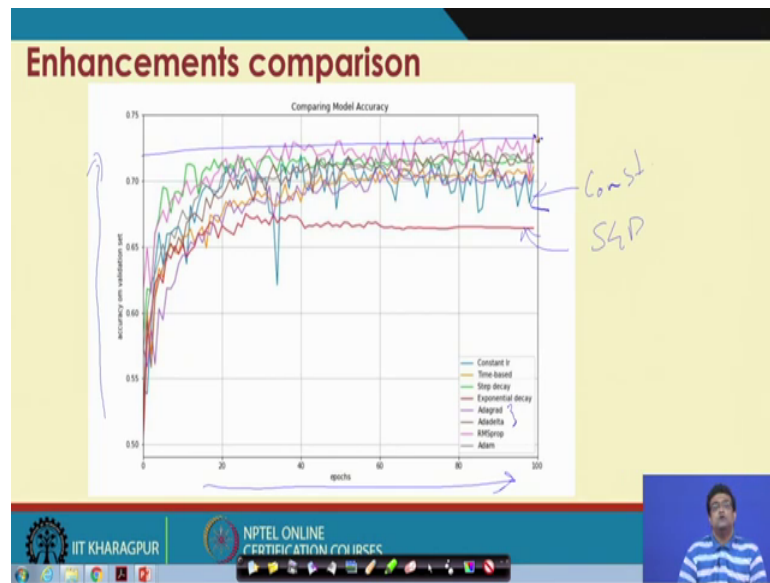
- Idea: In addition to storing an exponentially decaying average of past squared gradients like RMSprop, Adam also keeps an exponentially decaying average of past gradients.
- Updates:
 - $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
 - $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
 - $\hat{m}_t = m_t / (1 - \beta_1^t)$
 - $\hat{v}_t = v_t / (1 - \beta_2^t)$
 - $\vartheta_{t+1} = \vartheta_t - (\eta / (\sqrt{\hat{v}_t} + \epsilon)) \hat{m}_t$

The slide also features the IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES logos at the bottom, along with a small video inset of a presenter.

And then the final algorithm, which combines both the momentum and the root mean square updates is what is called the ADAM algorithm. So, basically in order to in addition to storing an exponentially decaying average of past squared gradients like RMS prop, ADAM also keeps an exponentially decaying average of past gradients ok.

So, m basically stores the exponentially decaying average of past gradients as we have already discussed. And v stores the exponentially decaying average of past gradients. And then you d bias them using by dividing them by 1 minus beta to the power t. And then this is the same RMS prop update step that you apply. And this is one of the most widely used optimizers today in deep learning community.

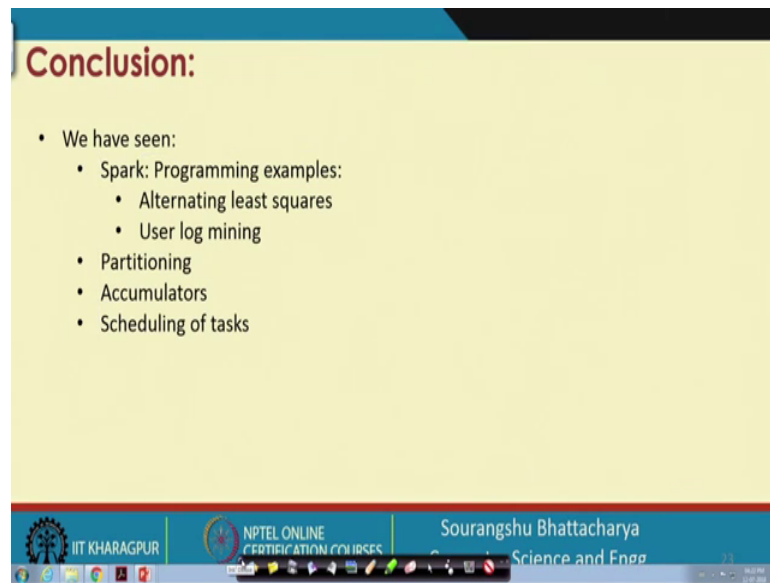
(Refer Slide Time: 32:50)



And this graph shows a comparison of many of these update algorithms. So, this so x axis is the number of epochs and y axis is the accuracy on the validation set ok. So, as the algorithm progresses the accuracy on the validation set increases. So, as you can see, so these are so the blue line is for constant a constant step size. So, this is simple SGD with constant step size. Red is SGD with exponentially decaying step size.

And there is there is step decay Adagrad and Adadelta. So, the thing to note is that RMS prop and ADAM reach the sort of the highest level. And though here it is not clear, but usually ADAM reaches the, reduces the cost a bit faster than RMS prop. But, both RMS prop and ADAM optimizer work very well in practice ok.

(Refer Slide Time: 34:36)

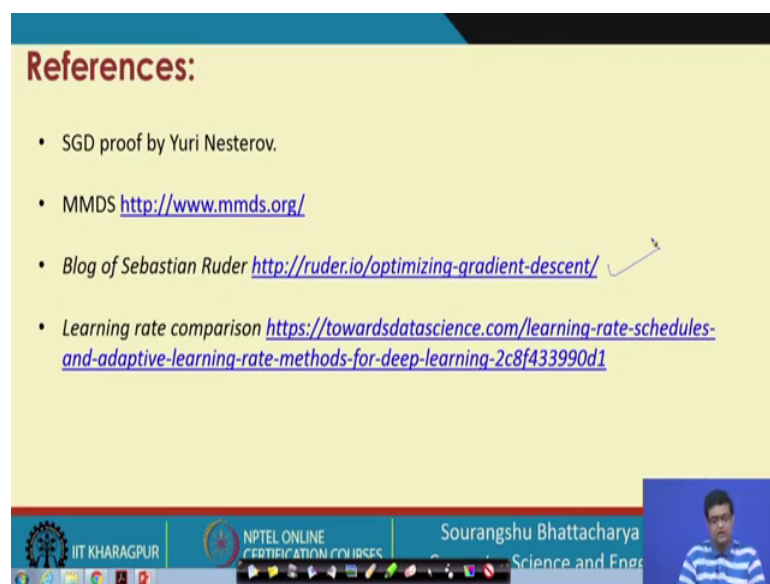


Conclusion:

- We have seen:
 - Spark: Programming examples:
 - Alternating least squares
 - User log mining
 - Partitioning
 - Accumulators
 - Scheduling of tasks

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Sourangshu Bhattacharya | Science and Engg

(Refer Slide Time: 34:43)



References:

- SGD proof by Yuri Nesterov.
- MMDS <http://www.mmds.org/>
- Blog of Sebastian Ruder <http://ruder.io/optimizing-gradient-descent/>
- Learning rate comparison <https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Sourangshu Bhattacharya | Science and Engg

So, these are the references. So, the proof of SGD that I showed was from a paper by Yuri Nesterov, the stochastic gradient descent algorithm itself. And the experiments by Leon Bottou were taken from the MMDS slides. And the material on the latest advancements on stochastic gradient descent algorithm was taken from an excellent Blog by Sebastian Ruder.

And I would recommend that you read this for the other algorithm. So, there are some algorithms which I have not described like the Adagrad algorithm and the Adadelta

algorithm. And these are some cutting edge things, which has happened, so for example, ADAM was invented in 2015. And then the last reference is for the graphs or the comparison empirical comparison between the different optimizers.

Thank you.