



STOCHASTIC OPTIMIZATION FOR LARGE SCALE ML

SOURANGSHU BHATTACHARYA

CSE, IIT KHARAGPUR

WEB: [HTTPS://CSE.IITKGP.AC.IN/~SOURANGSHU/](https://cse.iitkgp.ac.in/~sourangshu/)

EMAIL: SOURANGSHU@CSE.IITKGP.AC.IN



MUCH OF ML IS OPTIMIZATION

Linear Classification

$$\begin{aligned} \arg \min_w \sum_{i=1}^n \|w\|^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t. } 1 - y_i x_i^T w \leq \xi_i \\ \xi_i \geq 0 \end{aligned}$$

Maximum Likelihood

$$\arg \max_{\theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

K-Means

$$\arg \min_{\mu_1, \mu_2, \dots, \mu_k} J(\mu) = \sum_{j=1}^k \sum_{i \in C_j} \|x_i - \mu_j\|^2$$



STOCHASTIC OPTIMIZATION

- Goal of machine learning :
 - Minimize expected loss

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

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

- This is Stochastic Optimization
 - Assume loss function is convex



BATCH (SUB)GRADIENT DESCENT FOR ML

- Process all examples together in each step

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^n \frac{\partial L(w, x_i, y_i)}{\partial w} \right)$$

where L is the regularized loss function

- Entire training set examined at each step
- Very slow when n is very large



STOCHASTIC (SUB)GRADIENT DESCENT

- “Optimize” one example at a time
- Choose examples randomly (or reorder and choose in order)
 - Learning representative of example distribution

for $i = 1$ to n :

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

where L is the regularized loss function



STOCHASTIC (SUB)GRADIENT DESCENT

for $i = 1$ to n :

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

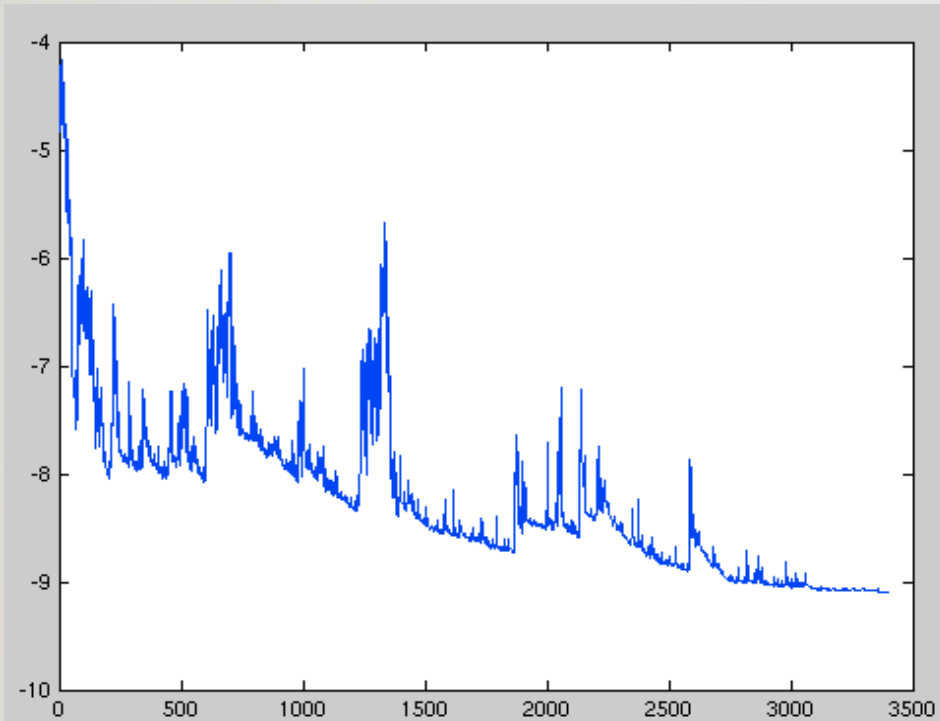
where L is the regularized loss function

- Equivalent to online learning (the weight vector w changes with every example)
- Convergence guaranteed for convex functions (to local minimum)



SGD CONVERGENCE

Objective function value



Iterations / updates

Objective function oscillates over the iterations.

Not a “Descent Method”

Maintain the **running minimum** loss and corresponding model parameters.



CONVERGENCE OF SGD

- Given dataset $D = \{(x_1, y_1), \dots, (x_m, y_m)\}$
- Loss function: $L(\theta, D) = \frac{1}{N} \sum_{i=1}^N l(\theta; x_i, y_i)$
- For linear models: $l(\theta; x_i, y_i) = l(y_i, \theta^T \phi(x_i))$
- Assumption D is drawn IID from some distribution \mathcal{P} .
- Problem:

$$\min_{\theta} L(\theta, D)$$



CONVERGENCE OF SGD

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

Algorithm:

- Initialize θ^0
- For $t = 1, \dots, T$

$$\theta^{t+1} = \theta^t - \eta_t \nabla_{\theta} l(y_t, \theta^T \phi(x_t))$$

- $\bar{\theta} = \frac{\sum_{t=1}^T \eta_t \theta^t}{\sum_{t=1}^T \eta_t}$.



CONVERGENCE OF SGD

- 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: $R = \|\theta^0 - \theta^*\|$
- $L = \max \nabla l(y, \theta^T \phi(x))$



SGD CONVERGENCE PROOF

- Define $r_t = \|\theta^t - \theta^*\|$ and $g_t = \nabla_{\theta} l(y_t, \theta^T \phi(x_t))$
- $r_{t+1}^2 = r_t^2 + \eta_t^2 \|g_t\|^2 - 2\eta_t (\theta^t - \theta^*)^T g_t$
- Taking expectation w.r.t $\mathcal{P}, \bar{\theta}$ and using $s^* - s(\theta^t) \geq E_{\mathcal{P}}[g_t]^T (\theta^* - \theta^t)$, we get:
$$E_{\bar{\theta}}[r_{t+1}^2 - r_t^2] \leq \eta_t^2 L^2 + 2\eta_t (s^* - E_{\bar{\theta}}[s(\theta^t)])$$

- Taking sum over $t = 1, \dots, T$ and using

$$E_{\bar{\theta}}[r_T^2 - r_0^2] \leq L^2 \sum_{t=0}^{T-1} \eta_t^2 + 2 \sum_{t=0}^{T-1} \eta_t (s^* - E_{\bar{\theta}}[s(\theta^t)])$$



SGD CONVERGENCE PROOF

- Using convexity of s :

$$\left(\sum_{t=0}^{T-1} \eta_t \right) E_{\bar{\theta}} [s(\bar{\theta})] \leq E_{\bar{\theta}} \left[\sum_{t=0}^{T-1} \eta_t s(\theta^t) \right]$$

- Substituting in the expression from previous slide:

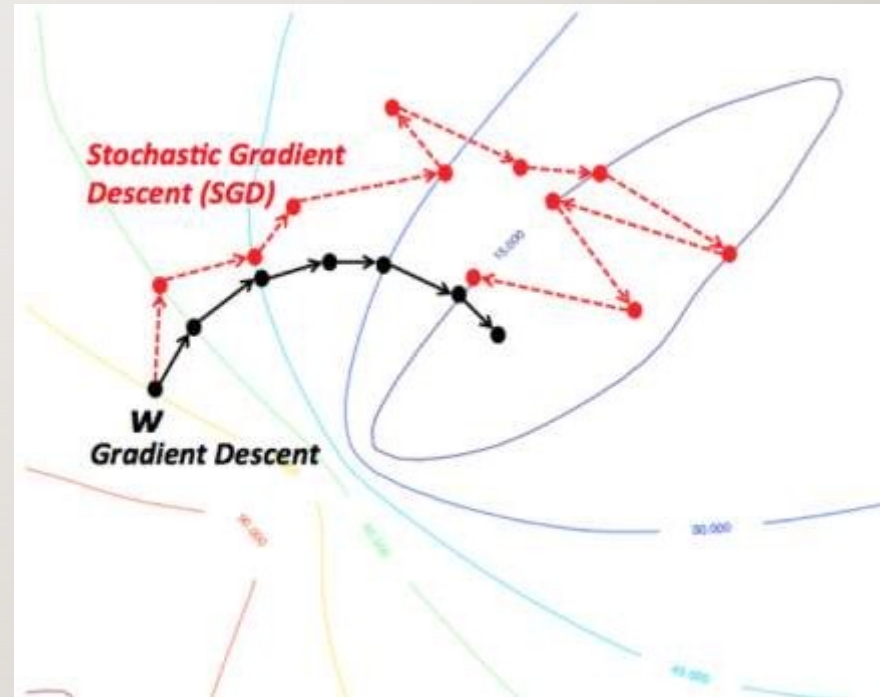
$$E_{\bar{\theta}} [r_T^2 - r_0^2] \leq L^2 \sum_{t=0}^{T-1} \eta_t^2 + 2 \sum_{t=0}^{T-1} \eta_t (s^* - E_{\bar{\theta}} [s(\bar{\theta})])$$

- Rearranging the terms proves the result.



SGD - ISSUES

- Convergence very sensitive to learning rate (η_t) (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!





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



EXAMPLE: TEXT CATEGORIZATION

- **Example by Leon Bottou:**
 - **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



EXAMPLE: TEXT CATEGORIZATION

• Questions:

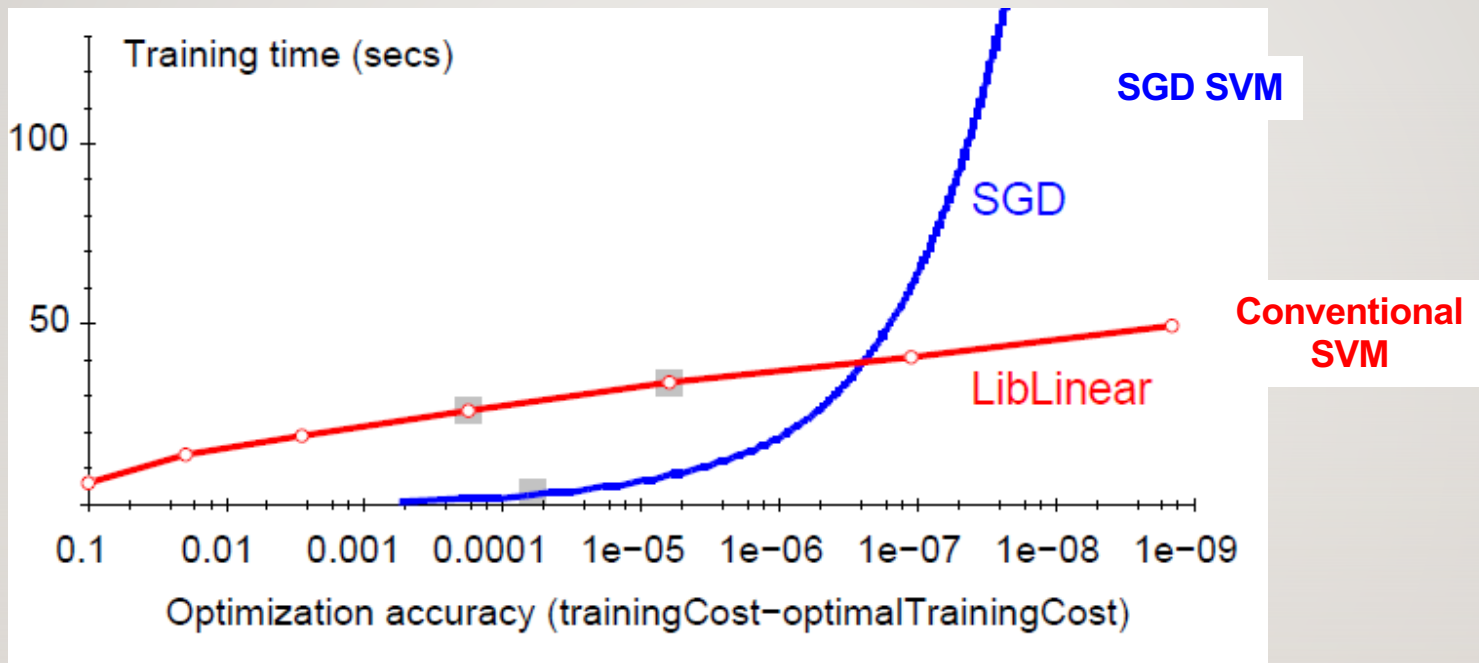
- (1) Is **SGD** successful at minimizing $f(w,b)$?
- (2) How quickly does **SGD** find the min of $f(w,b)$?
- (3) What is the error on a test set?

	<i>Training time</i>	<i>Value of $f(w,b)$</i>	<i>Test error</i>
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



OPTIMIZATION “ACCURACY”



Optimization quality: $| f(w,b) - f(w^{opt},b^{opt}) |$

For optimizing $f(w,b)$ within reasonable quality SGD-SVM is super fast



LEARNING RATE / STEP-SIZE SCHEDULE

- Need to choose learning rate η and t_0

$$w_{t+1} \leftarrow w_t - \frac{\eta_0}{t + t_0} \left(\frac{\partial L(x_i, y_i)}{\partial w} \right); \quad \eta = \frac{\eta_0}{t + t_0}$$

- Leon suggests:

- Choose t_0 so that the expected initial updates are comparable with the expected size of the weights
- Choose η_0 :
 - Select a **small subsample**
 - Try various rates η_0 (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
- Alternative form:

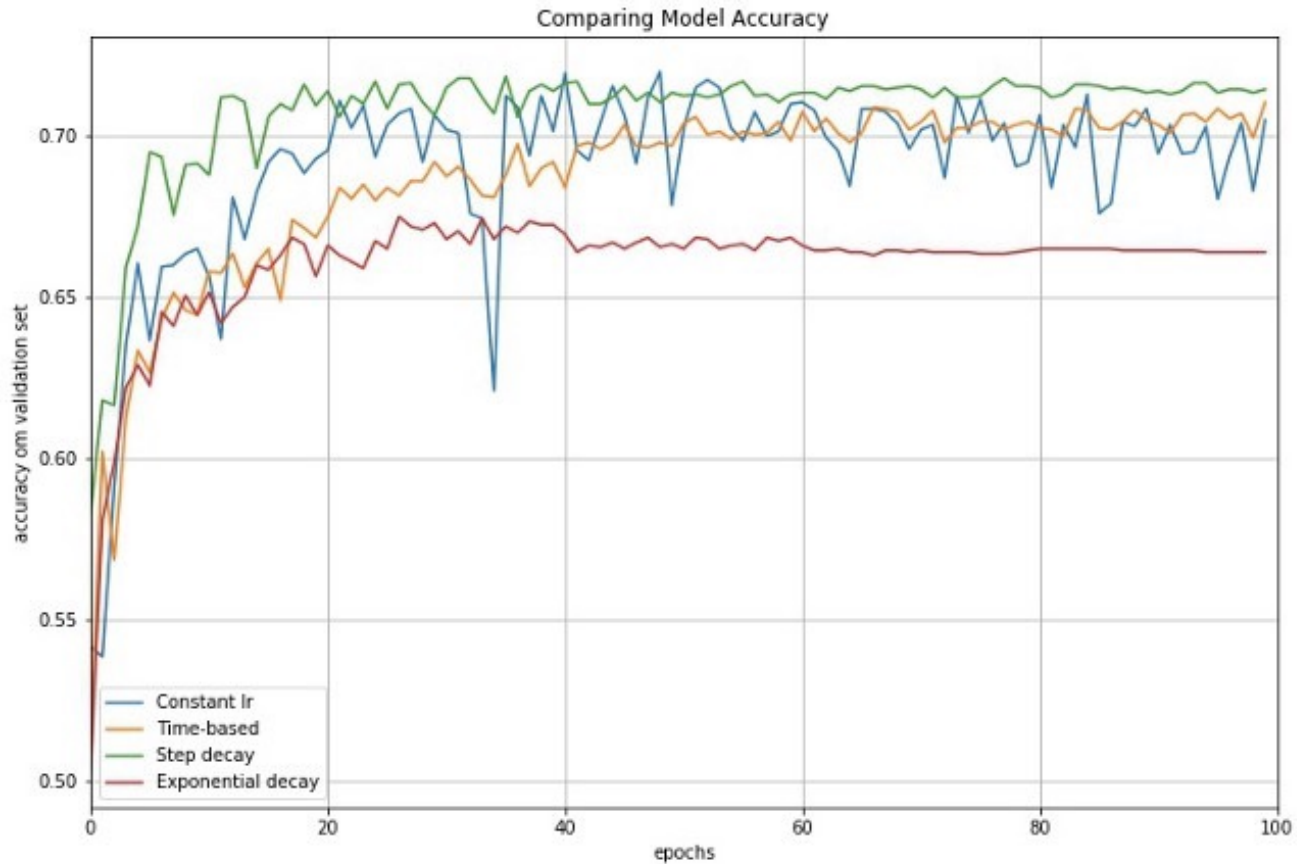
$$\eta = \frac{\eta_0}{1 + (\text{decay} * t)}$$

- Step decay schedule:

- Drop the learning rate by half every 10 epochs.

- $\eta = \eta_0 * (\text{drop})^{\text{floor}(\frac{t}{t_{\text{drop}}})}$

LEARNING RATE COMPARISON





ACCELERATED GRADIENT DESCENT



STOCHASTIC GRADIENT DESCENT

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



MOMENTUM GRADIENT DESCENT

- Idea: Overcome ravine oscillations by momentum
-

Update:

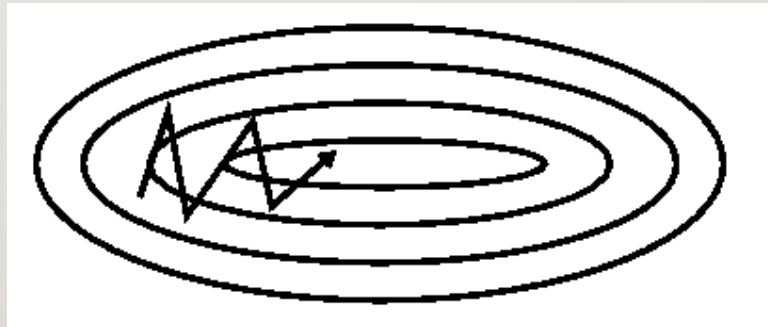
$$v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J(\theta)$$

$$\theta = \theta - v_t$$

SGD



SGD with
momentum



WHY MOMENTUM REALLY WORKS

The momentum term **reduces updates for dimensions whose gradients change directions.**



The momentum term **increases for dimensions whose gradients point in the same directions.**

Demo : <http://distill.pub/2017/momentum/>



NESTEROV ACCELERATED GRADIENT

- However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory.
- We would like to have a smarter ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.
- **Nesterov accelerated gradient** gives us a way of it.



NESTEROV ACCELERATED GRADIENT

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

Approximation of the next position of the parameters(predict)



NESTEROV ACCELERATED GRADIENT

Approximation of the next position of the parameters' gradient(**correction**)

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

Approximation of the next position of the parameters(**predict**)



NESTEROV ACCELERATED GRADIENT



Blue line : predict

Approximation of the next position of the parameters' gradient (correction)

Red line : correction

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

Green line : accumulated gradient

Approximation of the next position of the parameters (predict)



NESTEROV ACCELERATED GRADIENT



Blue line : predict

Approximation of the next position of the parameters' gradient (**correction**)

Red line : correction

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

Green line : accumulated gradient

Approximation of the next position of the parameters (**predict**)



NESTEROV ACCELERATED GRADIENT



Blue line : predict

Red line : correction

Green line : accumulated gradient

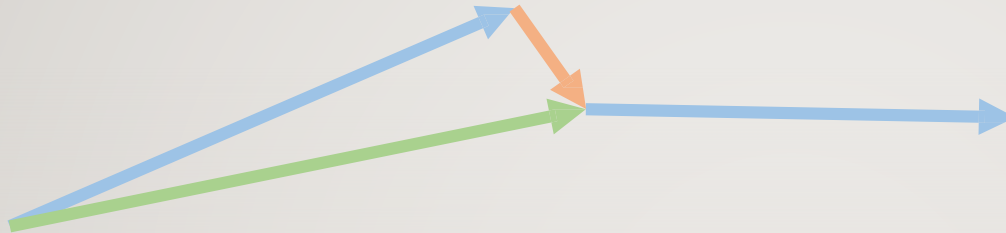
Approximation of the next position of the parameters' gradient (correction)

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

Approximation of the next position of the parameters (predict)



NESTEROV ACCELERATED GRADIENT



Blue line : predict

Red line : correction

Green line : accumulated gradient

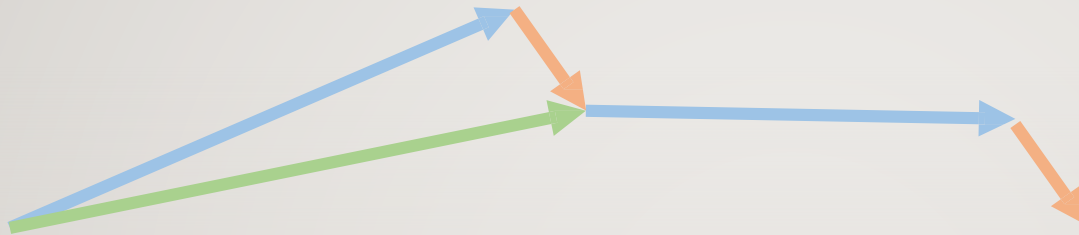
Approximation of the next position of the parameters' gradient (correction)

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

Approximation of the next position of the parameters (predict)



NESTEROV ACCELERATED GRADIENT



Blue line : predict

Red line : correction

Green line : accumulated gradient

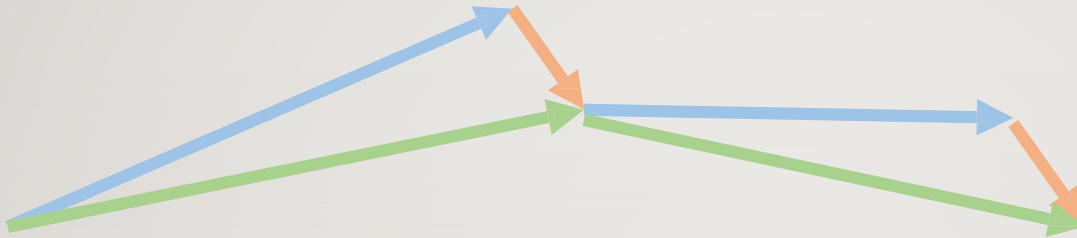
Approximation of the next position of the parameters' gradient (correction)

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

Approximation of the next position of the parameters (predict)



NESTEROV ACCELERATED GRADIENT



Blue line : predict

Approximation of the next position of the parameters' gradient (correction)

Red line : correction

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

Green line : accumulated gradient

Approximation of the next position of the parameters (predict)



NESTEROV ACCELERATED GRADIENT

- This anticipatory update **prevents** us from **going too fast** and **results in increased responsiveness**.
- Now , we can adapt our updates to the slope of our error function and **speed up SGD** in turn.



ADAPTIVE GRADIENTS

- Previous methods :
 - we used the same learning rate η for all parameters θ
- Adagrad :
 - It uses a **different learning rate** for every parameter θ_i at every time step t



WHAT'S NEXT ?

- We also want to adapt our updates to each individual parameter to perform larger or smaller updates **depending on their importance.**
 - Adagrad
 - Adadelta
 - RMSprop
 - Adam



ADAGRAD

- Adagrad adapts the learning rate to the parameters
 - Performing larger updates for infrequent
 - Performing smaller updates for frequent parameters.
- Ex.
 - Training large-scale neural nets at Google that learned to recognize cats in Youtube videos.



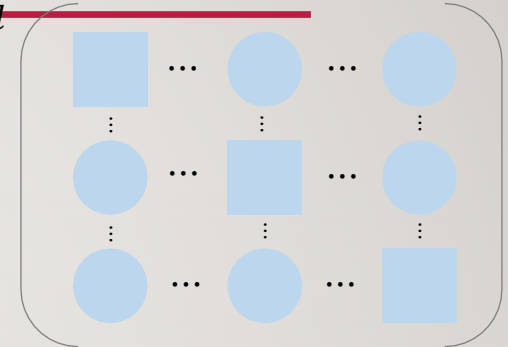
ADAGRAD

SGD

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$\mathbb{R}^{d \times d}$

$G_t =$



Adagrad modifies the general learning rate η based on the **past gradients** that have been computed for θ_i

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$g_{t,i} = \nabla_{\theta} J(\theta_i)$$

Vectorize

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$



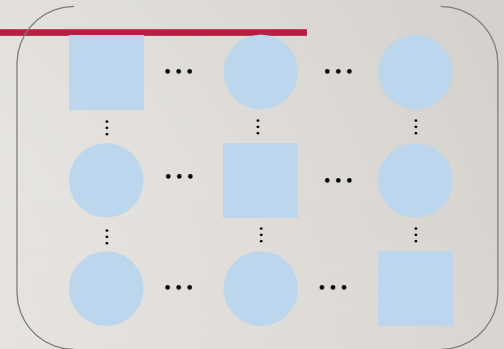
ADAGRAD

SGD

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$\mathbb{R}^{d \times d}$

$G_t =$



G_t is a diagonal matrix where each diagonal element (i,i) is the sum of the squares of the gradients θ_i up to time step t .

$$G_{t,ii} = \sum_{k=1}^t g_{k,i}^2$$

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Vectorize

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$



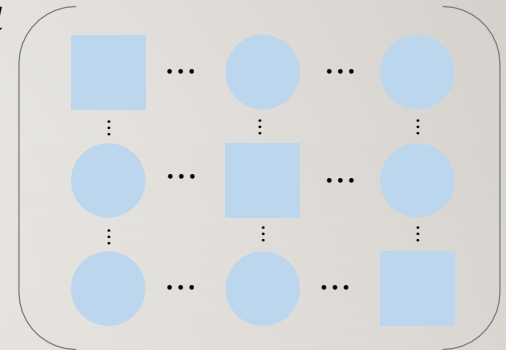
ADAGRAD

SGD

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$\mathbb{R}^{d \times d}$

$G_t =$



ϵ is a smoothing term that avoids division by zero (usually on the order of $1e - 8$).

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$g_{t,i} = \nabla_{\theta} J(\theta_i)$$

Vectorize

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$



ADAGRAD'S ADVANTAGE

- Advantages :
 - It is well-suited for dealing with sparse data.
 - It greatly improved the robustness of SGD.
 - It eliminates the need to manually tune the learning rate.



ADAGRAD'S DISADVANTAGE

- Disadvantage :
 - Main weakness is its accumulation of the squared gradients in the denominator.



ADAGRAD'S DISADVANTAGE

- The disadvantage causes the learning rate to shrink and become infinitesimally small. The algorithm can no longer acquire additional knowledge.
- The following algorithms aim to resolve this flaw.
 - Adadelta
 - RMSprop
 - Adam



ADADELTA

- The expected square sum of gradients is recursively defined as a decaying average of all past squared gradients.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2$$

- $E[g^2]_t$: The running average at time step t .
- γ : A fraction similarly to the Momentum term, around 0.9



ADADELTA

Adagrad

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

SGD

$$\begin{aligned}\Delta\theta_t &= -\eta \cdot g_{t,i} \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned}$$



Adadelta

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$



ADADELTA

Adagrad

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

SGD

$$\begin{aligned}\Delta\theta_t &= -\eta \cdot g_{t,i} \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned}$$



Replace the diagonal matrix G_t with the decaying average over past squared gradients $E[g^2]_t$

Adadelta

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$



ADADELTA

Adagrad

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

SGD

$$\begin{aligned}\Delta\theta_t &= -\eta \cdot g_{t,i} \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned}$$

Replace the diagonal matrix G_t with the decaying average over past squared gradients $E[g^2]_t$

Adadelta

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Adadelta

$$\Delta\theta_t = -\frac{\eta}{\text{RMS}[g]_t} g_t$$



ADADELTA

- The units in this update do not match and the update should have the same hypothetical units as the parameter.
 - As well as in SGD, Momentum, or Adagrad
- To realize this, first defining another exponentially decaying average

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma)\Delta\theta_t^2$$



ADADELTA

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

Adadelta

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}g_t$$

Adadelta

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t}g_t$$



ADADELTA UPDATE RULE

- Replacing the learning rate η in the previous update rule with $RMS[\Delta\theta]_{t-1}$ finally yields the Adadelta update rule:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

- Note : **we do not even need to set a default learning rate**



RMSPROP

RMSprop and Adadelta have both been developed independently around the same time to resolve Adagrad's radically diminishing learning rates.

RMSprop

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}g_t$$



RMSPROP

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients.

RMSprop

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}g_t$$

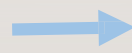
Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.



ADAM

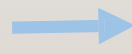
- Adam's feature :
 - Storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop
 - Keeping an exponentially decaying average of past gradients m_t , similar to momentum.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$



The first moment (the mean)

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



The second moment (the uncentered variance)



ADAM

- As m_t and v_t are initialized as vectors of 0's, they are biased towards zero.
 - Especially during the initial time steps
 - Especially when the decay rates are small
 - (i.e. β_1 and β_2 are close to 1).
- Counteracting these biases in Adam

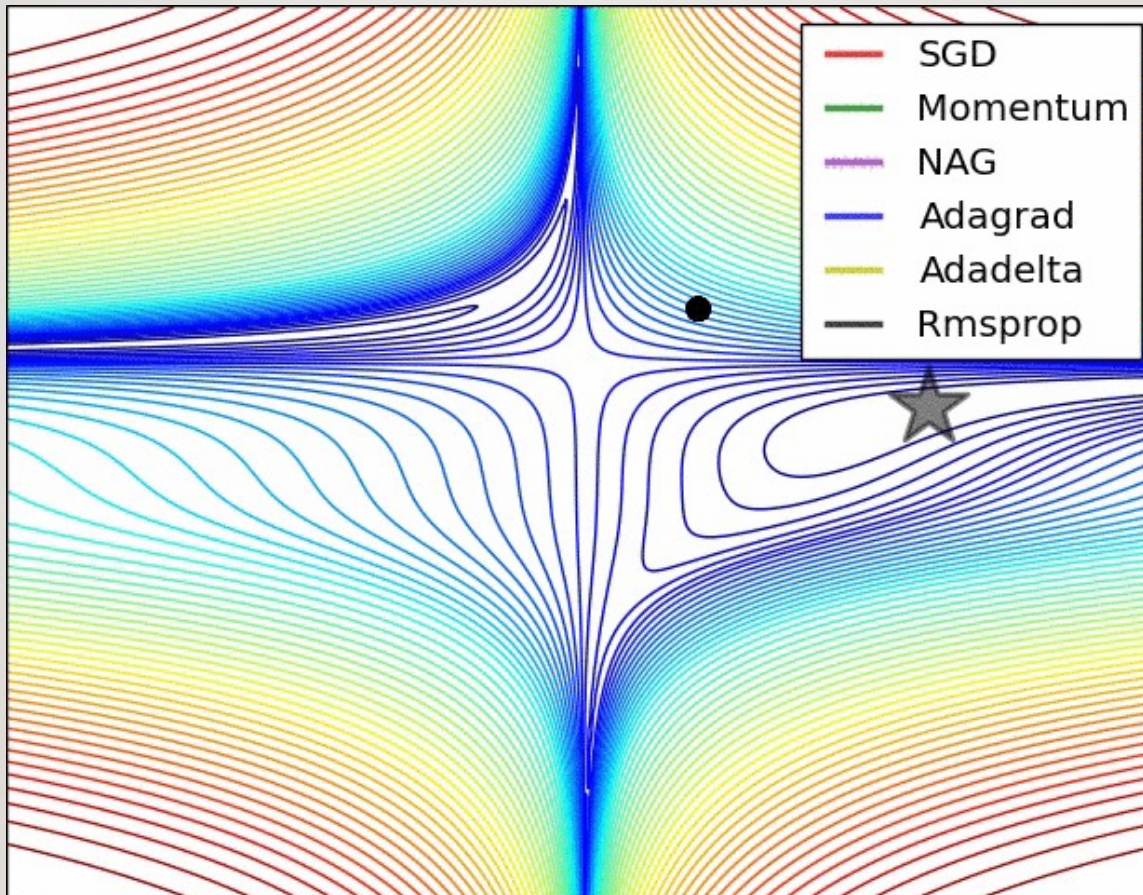
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Adam

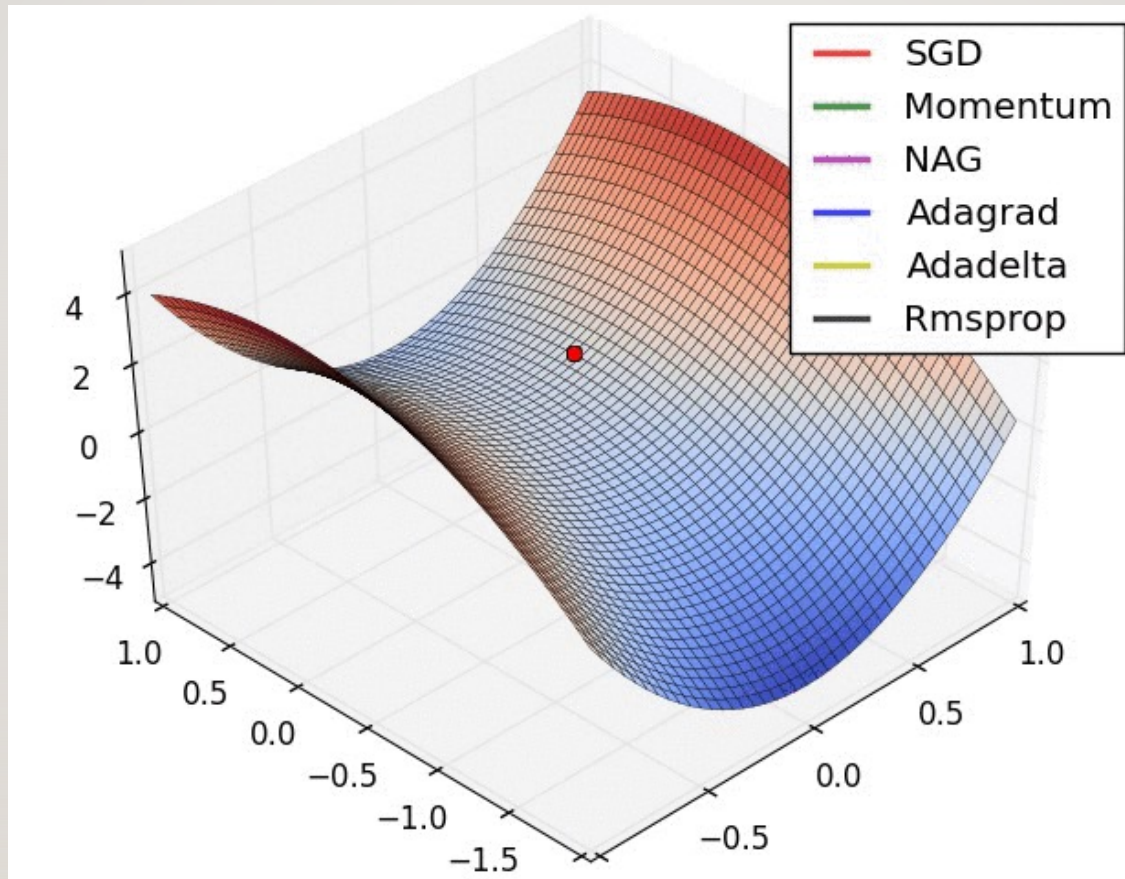
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Note : default values of 0.9 for β_1 , 0.999 for β_2 , and 10^{-8} for ϵ

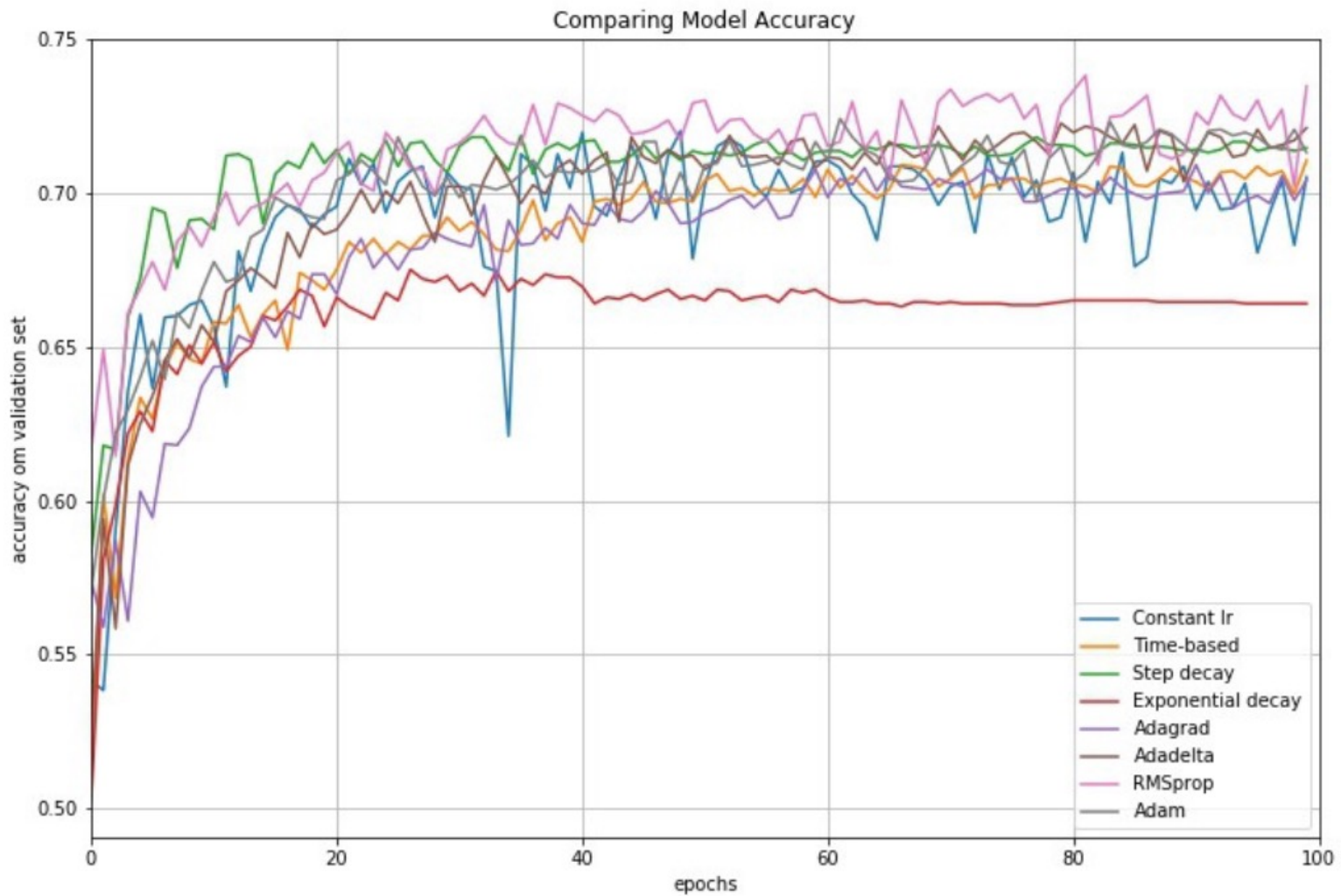
VISUALIZATION



VISUALIZATION



ENHANCEMENTS COMPARISON





SUMMARY

- There are two main ideas at play:
 - **Momentum** : Provide consistency in update directions by incorporating past update directions.
 - **Adaptive gradient** : Scale the scale updates to individual variables using the second moment in that direction.
 - This also relates to adaptively altering step length for each direction.



References:

- **SGD convergence** proof: “Confidence level solutions for stochastic programming” by Y. Nesterov, P. Vial, *Automatica*, 2008.
- **Accelerated SGD**: Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747* (2016).
- **First SGD in ML paper**:
Léon Bottou and Olivier Bousquet: **The Tradeoffs of Large Scale Learning**, *Advances in Neural Information Processing Systems*, 20, MIT Press, Cambridge, MA, 2008.



THANKS

QUESTIONS?

Email: sourangshu@cse.iitkgp.ac.in