CS60021: Scalable Data Mining

Large Scale Machine Learning

Sourangshu Bhattacharya

– Optimization → "Mathematical Optimization"

θ – parameters

Much of ML is optimization

Linear Classification

f(x) — min.

software.

$$\arg\min_{w} \sum_{i=1}^{n} ||w||^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{s.t. } \; 1 - y_i x_i^T w < \xi_i$$

$$\xi_i \geq 0$$

"solved"

Maximum Likelihood

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

$p_\theta(x_1, \ldots, x_n)$

K-Means

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

Combinatorial

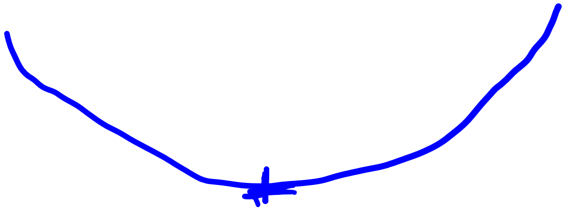
$\min_x f(x)$

s.t. $g_1(x) \leq 0$

$\vdots$

$g_m(x) \leq 0$

x^*



Convex opti.

Stochastic optimization

- Goal of machine learning :
  - Minimize expected loss

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

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

- This is Stochastic Optimization
  - Assume loss function is convex

x_1 $\quad$ x_2

$(\lambda(x_1) + (1-\lambda)x_2)$

$\lambda \in [0,1]$

RHS $\quad$ LHS

$f(\lambda x_1 + (1-\lambda)x_2) \le \boxed{\lambda f(x_1) + (1-\lambda) f(x_2)}$

$\forall x_1, x_2$ $\quad$ f as convex

$\lambda(x_1) + (1-\lambda)x_2$

5000 a

25 ← 25×10 ← grad. w — 500

5×10^6

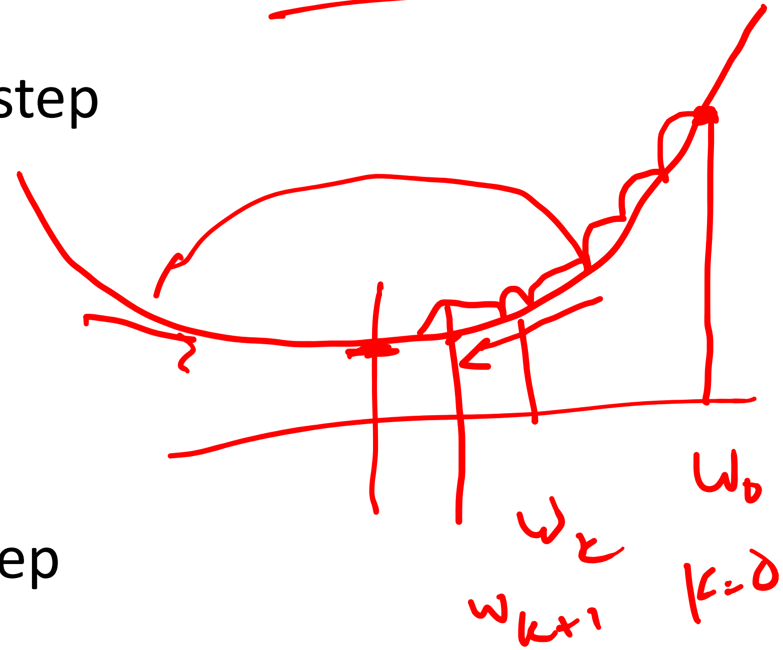
10,000

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

step towards -ve grad. direction.

SGD ← stochastic grad.

robust to training data

"stochastic gradient"

Stochastic (sub)gradient descent

stochastic optimization

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

for $k=1$ to # epochs.

for $i = 1$ to n:

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

E

where L is the regularized loss function

$\frac{\partial L(w,x,y)}{\partial w}$

$n \times$ # epochs

$\text{loss} = E_{(x,y)\sim P}\left[L(w,x,y)\right]$

$\nabla \text{loss} = E_{(x,y)\sim p}\left[\frac{\partial L(w,x,y)}{\partial w}\right]$

Stochastic (sub)gradient descent

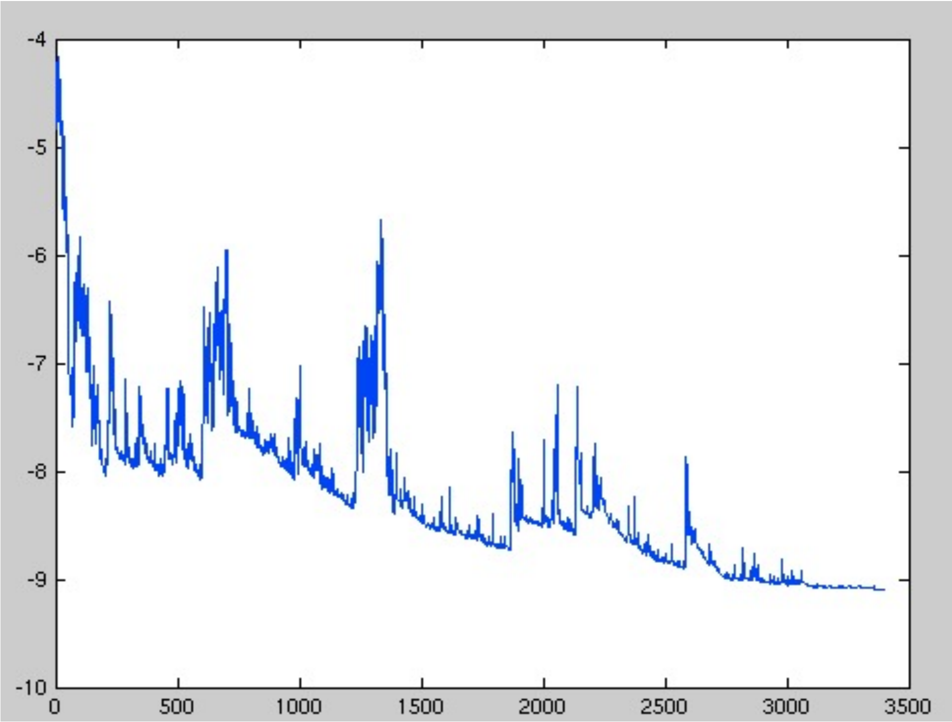
for $i = 1$ to n:

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

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

Stochastic gradient descent

$P_i = \frac{1}{N}$ P

- Given dataset $D = \{(x_1, y_1), \ldots, (x_m, y_m)\}$

N N

- 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:

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

$\eta_t \rightarrow$ $E(l(\bar{\theta})) \rightarrow E(l(\theta^*))$

Stochastic gradient descent

$\frac{1}{N}$

$(x_t, y_t) \sim \text{I.I.D.} (\mathcal{D})$

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

Algorithm:

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

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

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

$$w_t = \frac{\eta_t}{\sum \eta_t}$$

$$\bar{\theta} = \sum_{t=1}^{T} w_t \theta^t$$

$\frac{1}{n}\left(l(x_1)+l(x_2)+\dots+l(x_n)\right)\ \theta \qquad \sum \frac{1}{t^2}$

$\eta_t \qquad \eta_t = \frac{1}{t}$

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:

$\eta_t = \frac{1}{\sqrt{t}}$

$$\boxed{E_{\bar{\theta}}[s(\bar{\theta})] - s^* \le \frac{R^2 + L^2 \sum_{t=1}^{T} \eta_t^2}{2 \sum_{t=1}^{T} \eta_t}} \qquad T \to 0$$

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

$$\frac{\sum \eta_t^2}{\sum \eta_t}$$

$s(\bar{\theta})$

$$\boxed{\sum \eta_t^2 < \infty}$$

$$\boxed{\sum \eta_t \to \infty}$$

SGD convergence proof

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

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

SGD convergence proof

- Using convexity of s:

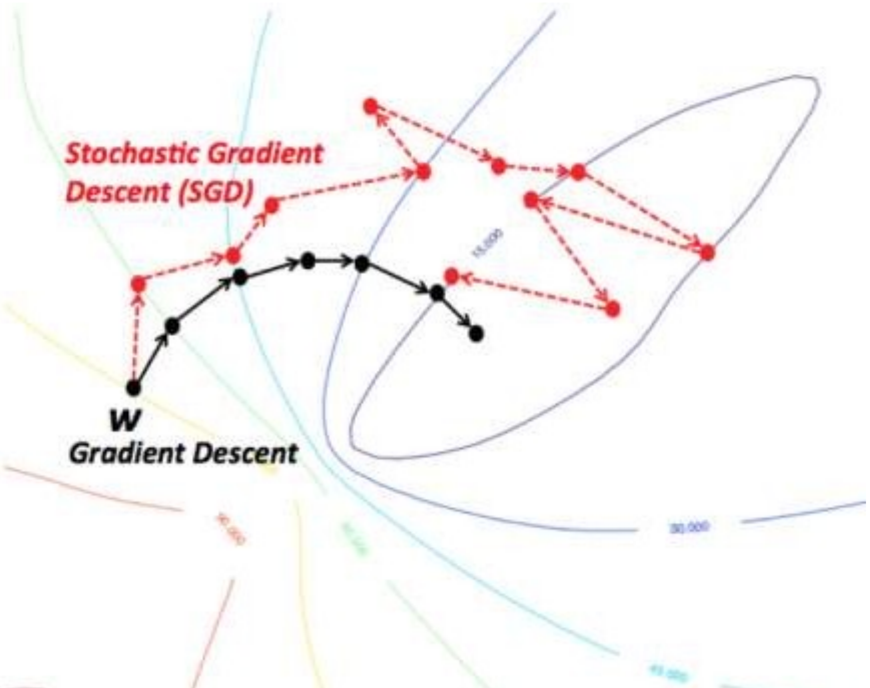
$$\left(\sum_{t=0}^{T-1}\eta_t\right)E_{\overline{\theta}}[s(\bar{\theta})] \leq E_{\overline{\theta}}[\sum_{t=0}^{T-1}\eta_t s(\theta^t)]$$

- Substituting in the expression from previous slide:

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

- Rearranging the terms proves the result.

The fluctuation : Batch vs SGD



https://wikidocs.net/3413

Batch gradient descent converges to the minimum of the basin the parameters are placed in and the **fluctuation is small**.

SGD's fluctuation is large but it enables to jump to new and potentially better local minima.

However, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting

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

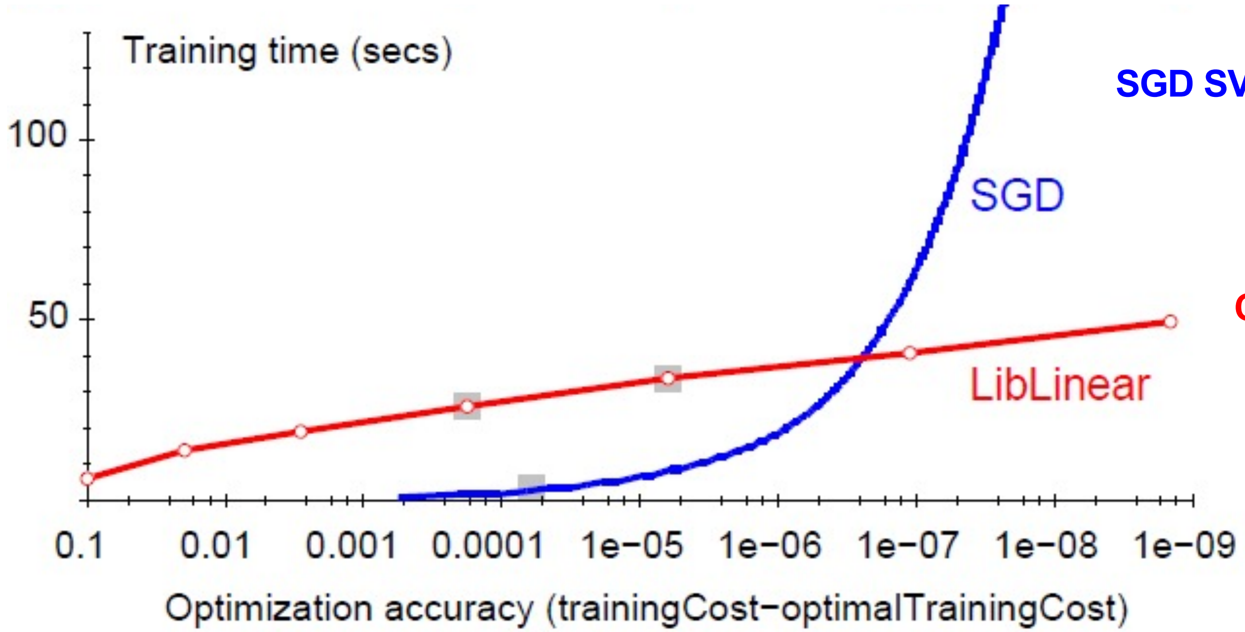
Example: Text categorization

- **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?

	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

Optimization "Accuracy"



SGD SVM

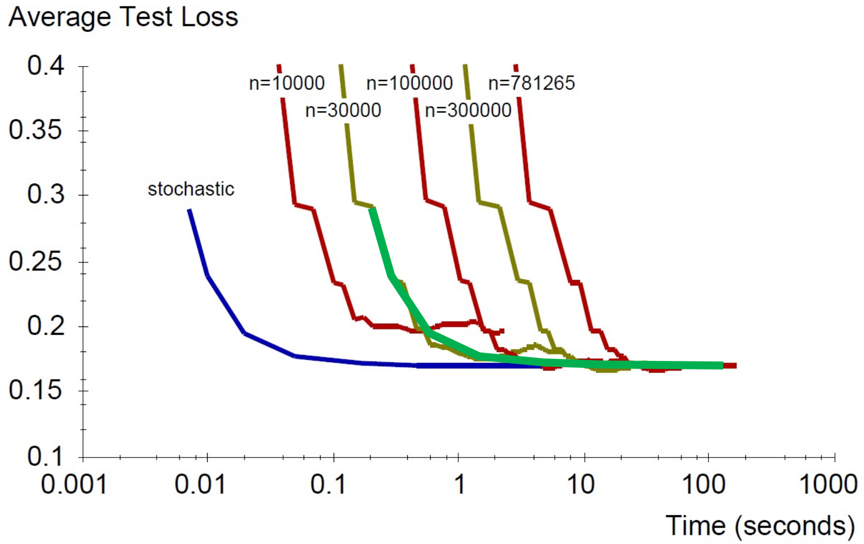
Conventional SVM

Optimization quality: | *f(w,b)* – *f* *(*w^{opt}*,*b^{opt}*)* |

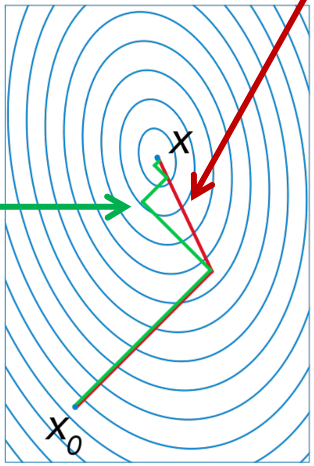
For optimizing *f(w,b) within reasonable* quality *SGD-SVM* is super fast

SGD vs. Batch Conjugate Gradient

SGD on full dataset vs. **Conjugate Gradient** on a sample of ***n*** training examples



Theory says: Gradient descent converges in linear time k. **Conjugate gradient** converges in $\sqrt{k}$.
k… condition number



Bottom line: Doing a simple (but fast) SGD update many times is better than doing a complicated (but slow) CG update a few times

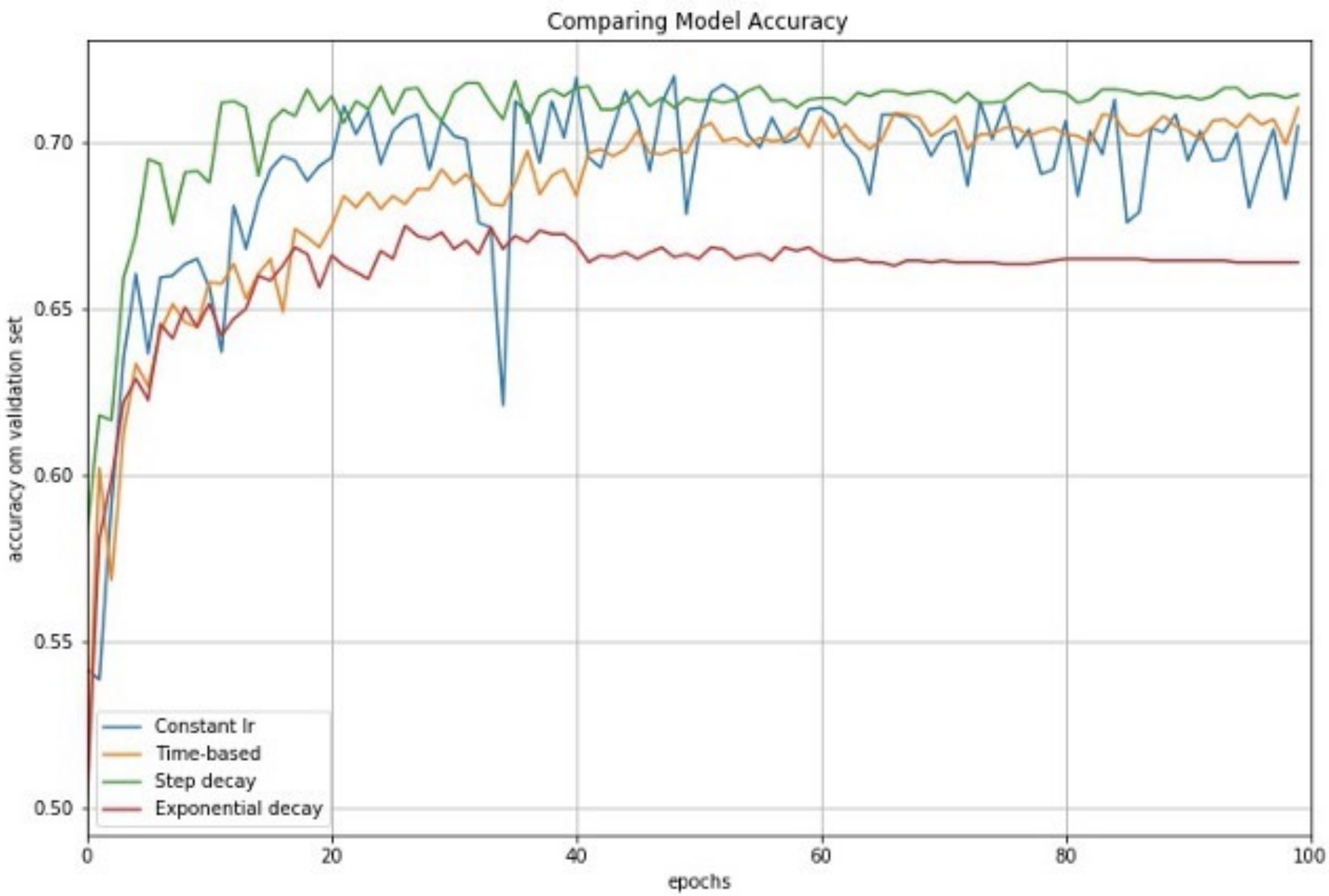
Practical Considerations

- **Need to choose learning rate η and t_0**

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

- **Leon suggests:**
  - Choose $\mathbf{t_0}$ 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

Learning rate comparison



Practical Considerations

- **Sparse Linear SVM:**
  - **Feature vector x_i is sparse (contains many zeros)**
    - Do not do: $\boldsymbol{x_i} = [0,0,0,1,0,0,0,0,5,0,0,0,0,0,0,\ldots]$
    - But represent $\mathbf{x_i}$ as a sparse vector $\mathbf{x_i}$**=[*(4,1)*, *(9,5)*, …]**
  - **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: $\boldsymbol{x_i}$ is sparse and so few coordinates $\boldsymbol{j}$ of $\boldsymbol{w}$ will be updated

$$w \leftarrow w(1-\eta)$$

expensive: $\boldsymbol{w}$ is not sparse, all coordinates need to be updated

Practical Considerations

- **Solution 1:** $\boldsymbol{w = s \cdot v}$
  - Represent vector ***w*** as the product of scalar ***s*** and vector ***v***
  - Then the update procedure is:
    - *(1)* $\boldsymbol{v = v - \eta C \frac{\partial L(x_i, y_i)}{\partial w}}$
    - *(2)* $\boldsymbol{s = s(1 - \eta)}$

Two step update procedure:

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

(2) $w \leftarrow w(1-\eta)$

- **Solution 2:**
  - Perform only step **(1)** for each training example
  - Perform step **(2)** with lower frequency and higher $\boldsymbol{\eta}$

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

ACCELERATED GRADIENT DESCENT

Stochastic gradient descent

Idea: Perform a parameter update for each training example *x(i)* and label *y(i)*

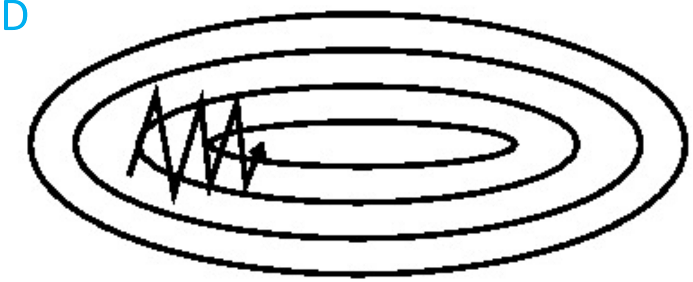
Update: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x(i), y(i))$

Performs redundant computations for large datasets

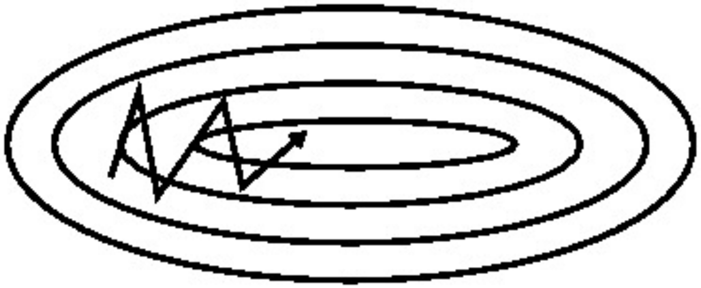
Momentum gradient descent

- Idea: Overcome ravine oscillations by momentum
- Update:
  - $v_t = \gamma v_{t-1} + \eta \cdot \nabla_\theta J(\theta)$
  - $\theta = \theta - v_t$

SGD

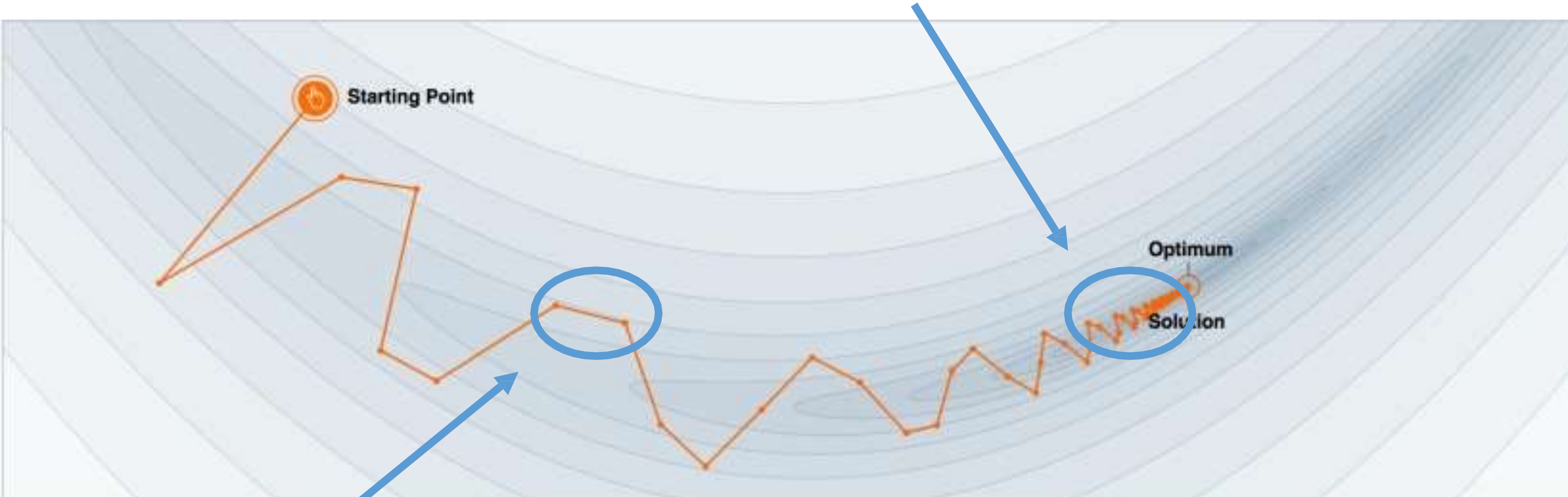


SGD with momentum



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)

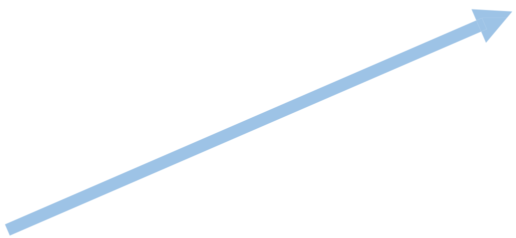
Nesterov accelerated gradient

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

Red line : correction

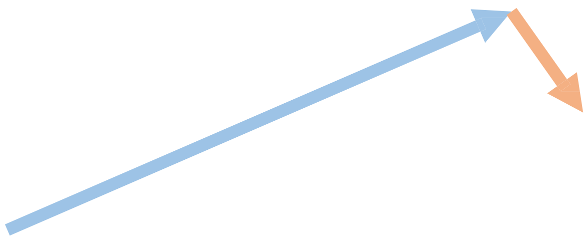
Green line :accumulated gradient

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

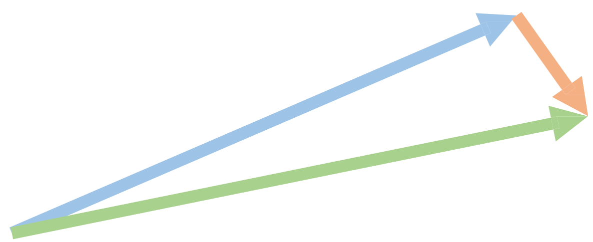
Green line :accumulated gradient

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

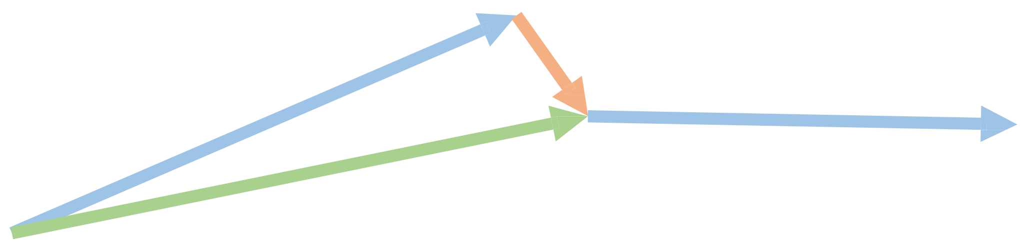
Green line :accumulated gradient

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

Red line : correction

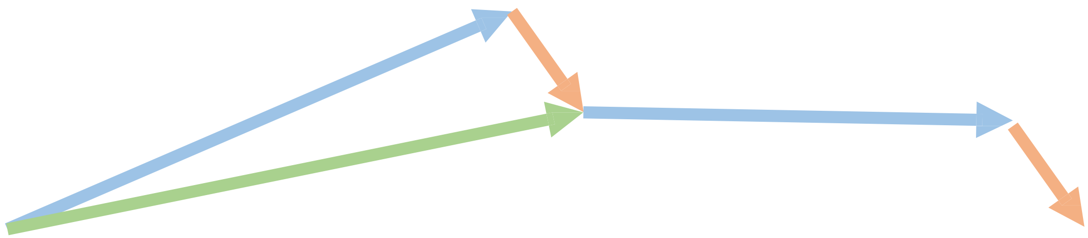
Green line :accumulated gradient

Approximation ofthe 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

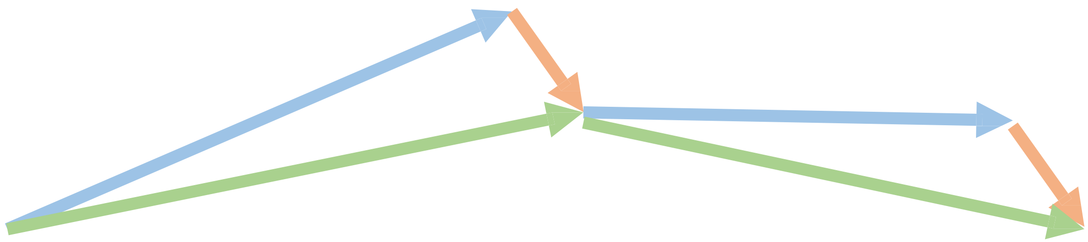
Green line :accumulated gradient

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

- 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.

AdaGrad

- Adapts the learning rate to the parameters
- Smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features

larger updates (i.e. high learning rates) for parameters associated with infrequent features

Update:

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

RMSprop

- Idea: Use the second moment of gradient vector to estimate the magnitude of update in a given direction.
- Update:
  - $E[g^2]_t = 0.9\, E[g^2]_{t-1} + 0.1\, g_t^2$
  - $\Delta\theta_t = -\, \eta / \sqrt{(E[g^2]_t + \epsilon)} \odot g_t$

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.

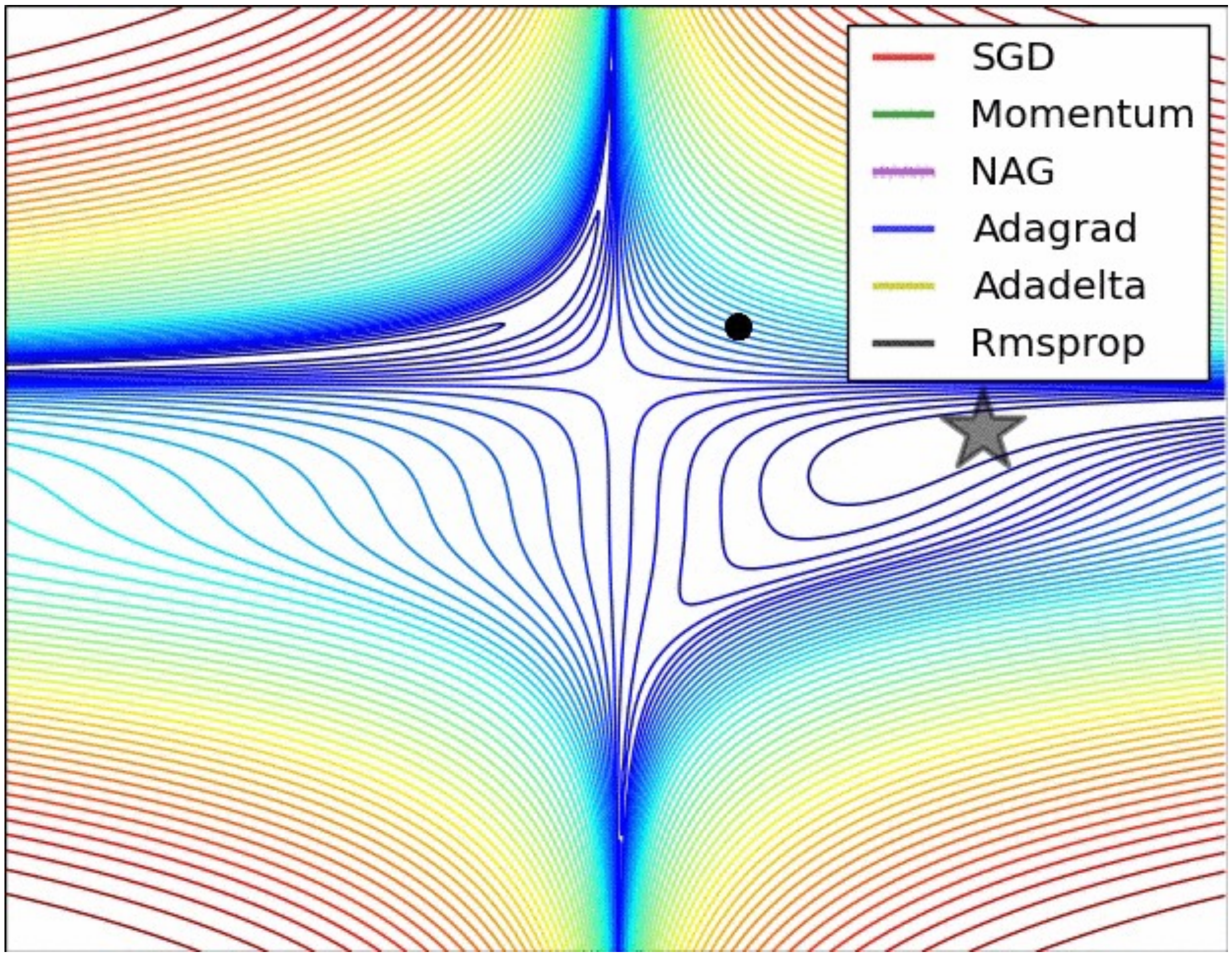
- Updates:

  - $m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t$
  - $v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2$

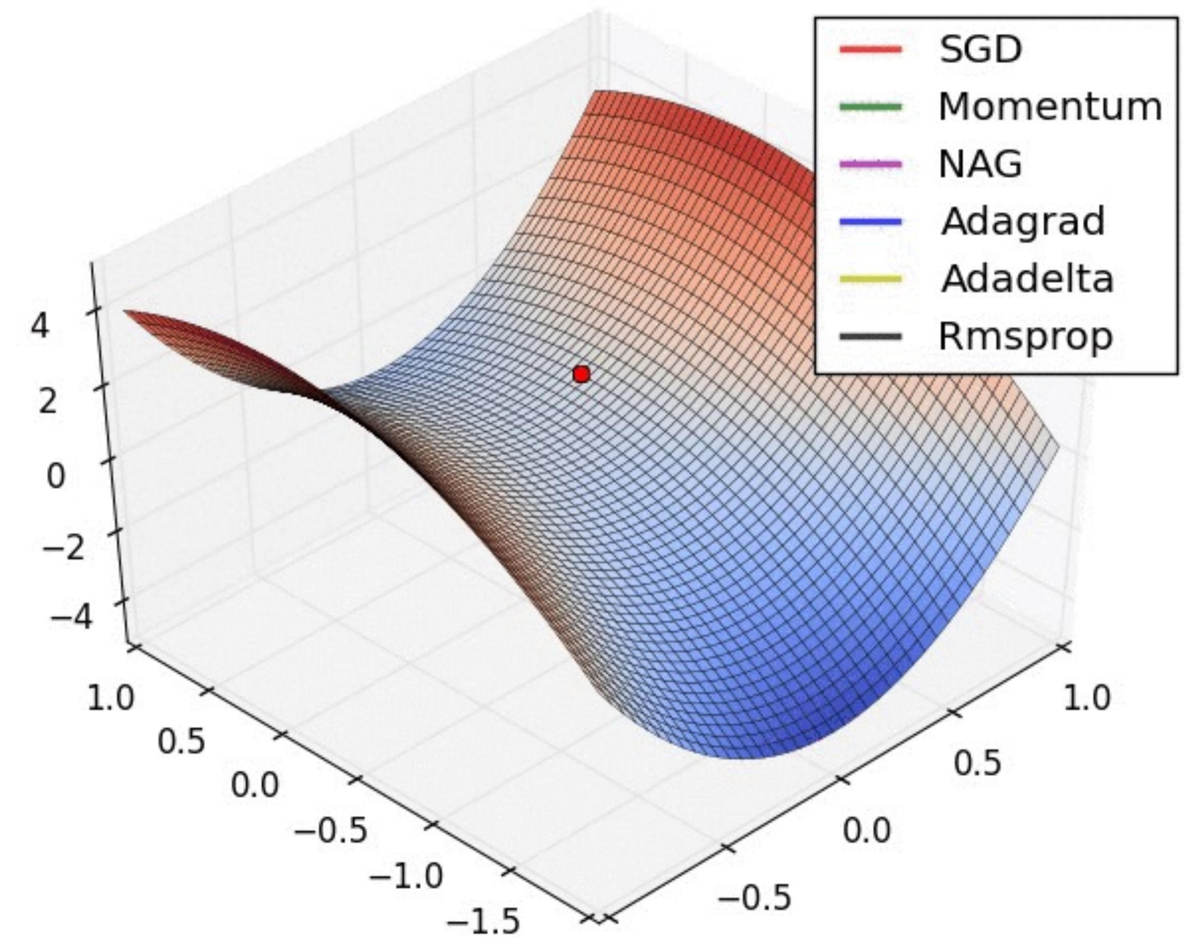
  - $\hat{m}_t = m_t / (1 - \beta_1^t)$
  - $\hat{v}_t = v_t / (1 - \beta_2^t)$

  - $\vartheta_{t+1} = \vartheta_t - (\eta / (\sqrt{\hat{v}_t} + \epsilon))\, \hat{m}_t$

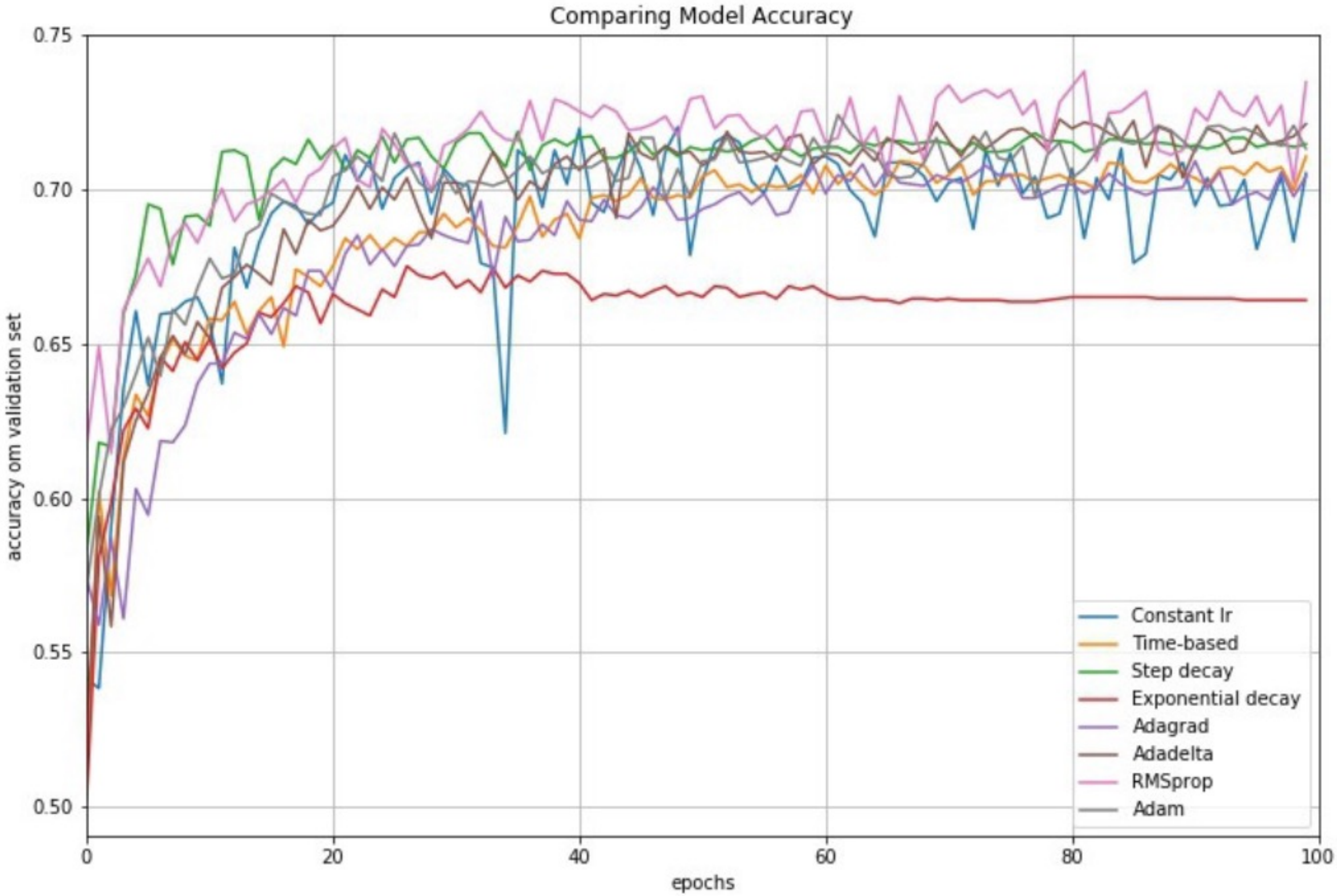
Visualization



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 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*