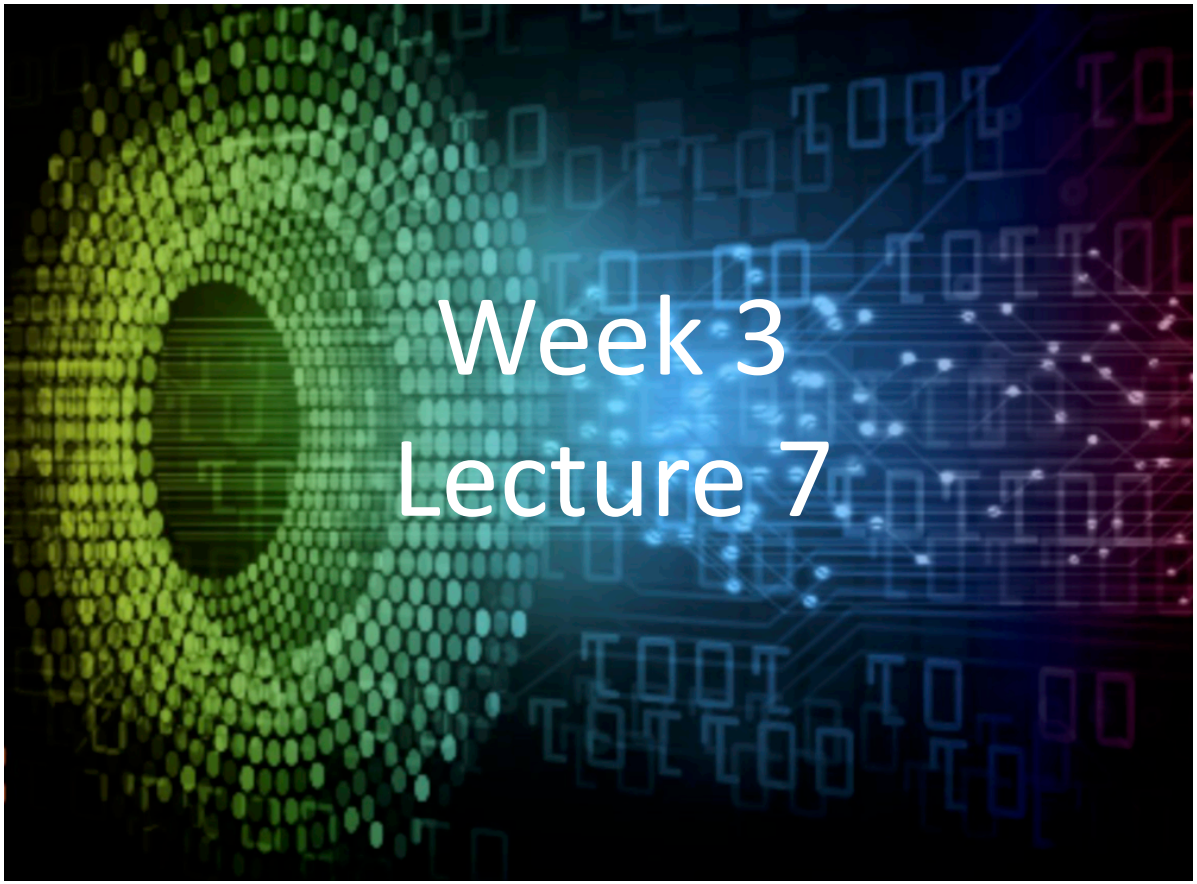
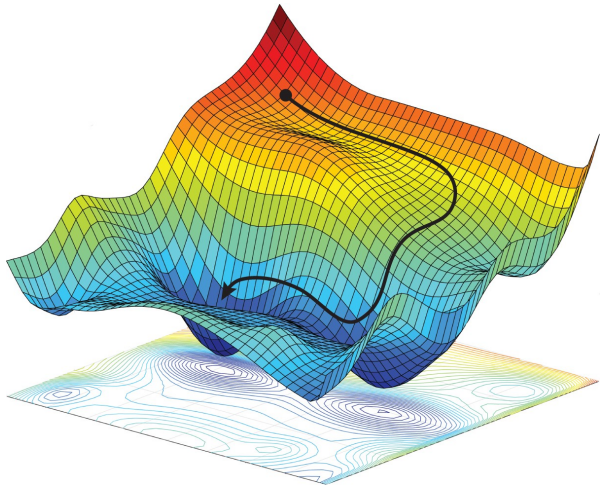
Introduction to Deep Learning
Applications and Theory



ECE 596 / AMATH 563

Previous Lecture/Week: **Network Training/Optimization**

1. Model Optimization
    1. Gradient Descent
    2. SGD
    3. Extensions
2. Training NN



Current Week: **The Data Driven Deep Learning Process**

Andrew Ng

Current Week: **The Data Driven Deep Learning Process**

1. Hyperparameters and Optimization
    1. Processing Stochastic Data (EMA)
    2. Optimization with Stochastic Data

rederive

Parameters

- **Parameters**

  Model Parameters:

  W, b, activation, output, cost

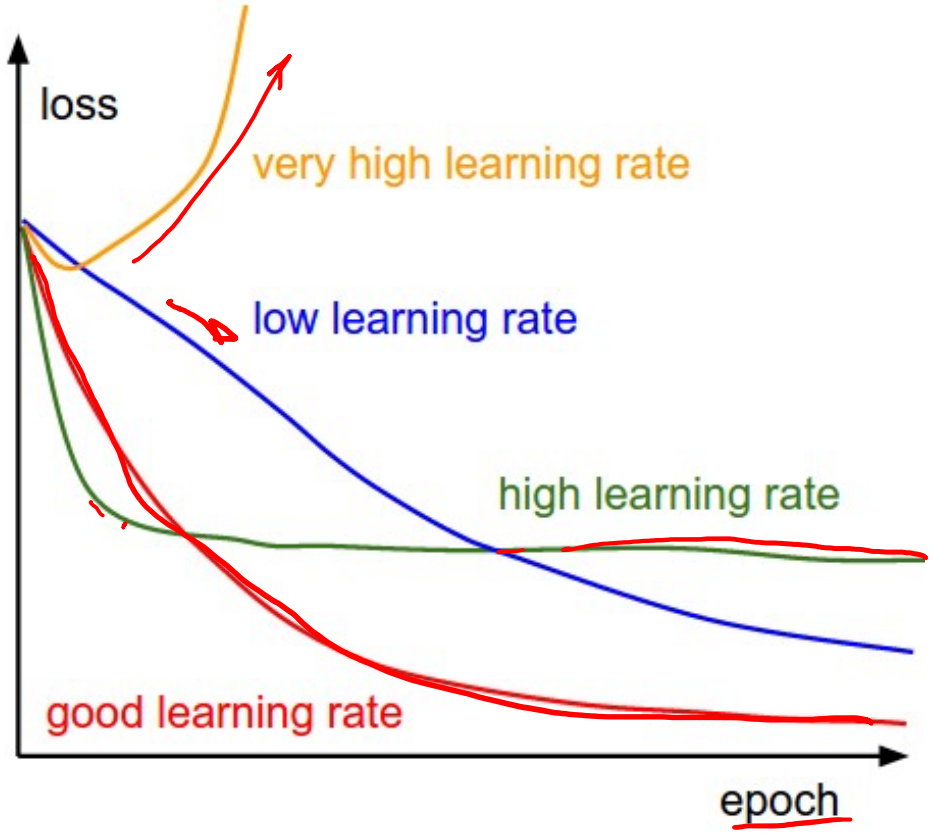
- **Hyper-parameters**

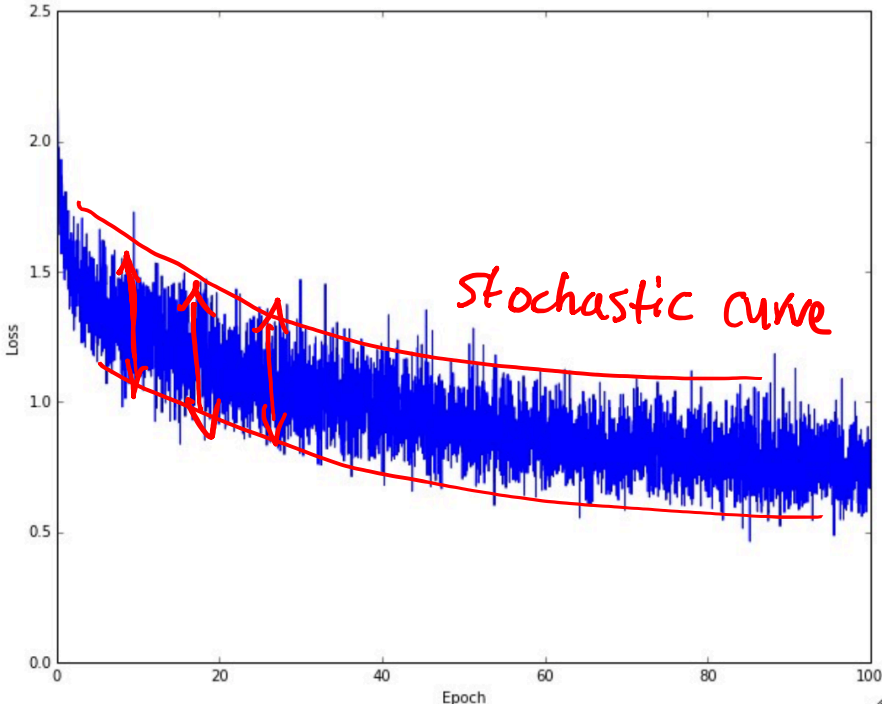
  External Parameters:

  Batch/minibatch size

  Learning parameters

Experimenting with GD





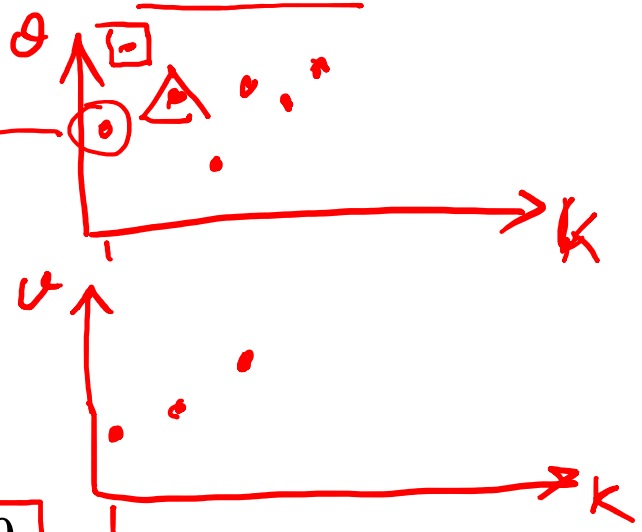
Hyperparameters = α

Stochastic Data – TSLA Stock

Exponentially Moving Average

(EMA)

$$v_k = \gamma v_{k-1} + (1-\gamma)\theta_k \qquad v_0 = 0$$



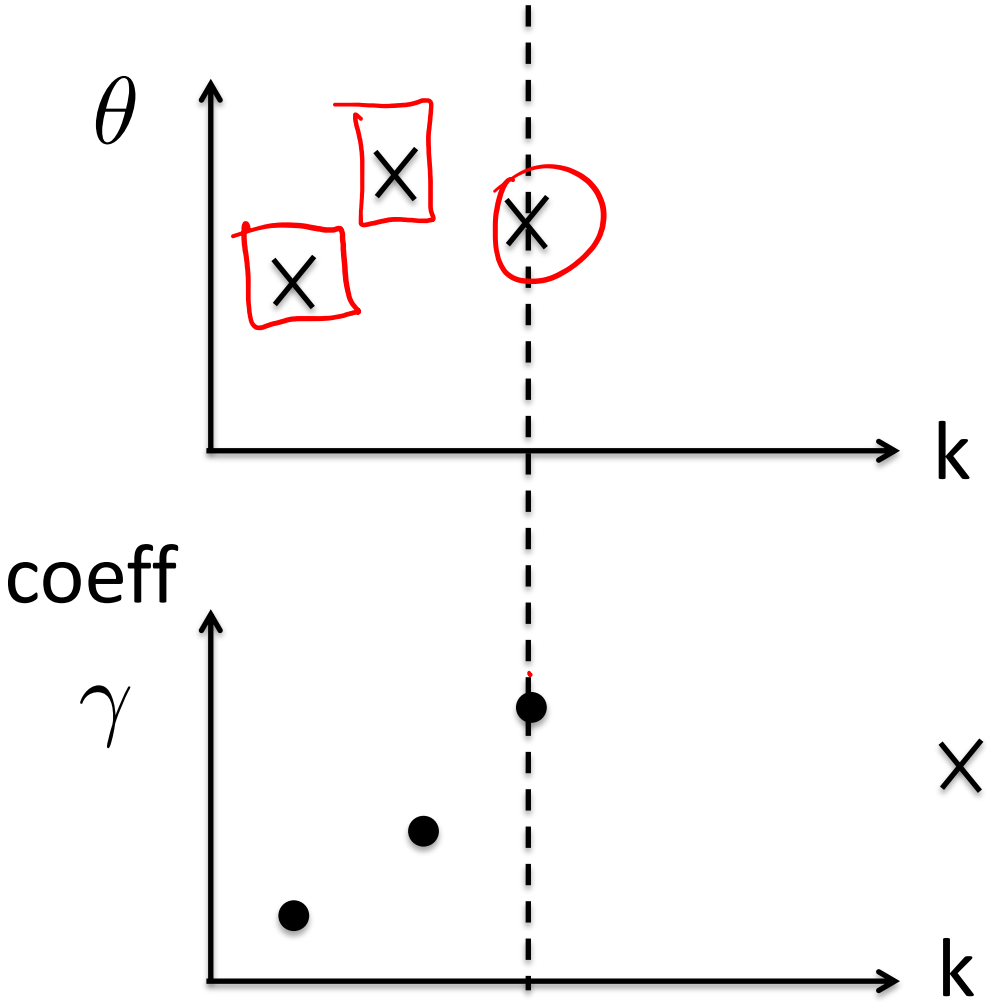
Let's iterate it:

$$v_1 = (1-\gamma)\theta_1$$

$$v_2 = \gamma(1-\gamma)\theta_1 + (1-\gamma)\theta_2$$

$$v_3 = \gamma^2(1-\gamma)\theta_1 + \gamma(1-\gamma)\theta_2 + (1-\gamma)\theta_3$$

Exponentially Moving Average



$$\times (1 - \gamma)$$

Window width ~

$$\frac{1}{1-\gamma}$$

$\gamma = 0.9$ $\approx$ 10 steps window

$\gamma = 0.98$ $\approx$ 50 steps window

Exponentially Moving Average

Momentum Method

- Let's apply it to the **gradient** values

$$\check{v}_k = \gamma \check{v}_{k-1} + (1-\gamma)\nabla_{w\,k}$$

$$w_{k+1} = w_k - \alpha \check{v}_k$$

1

Smoothens the gradient

Hyperparameters = α, γ

β

RMS Prop

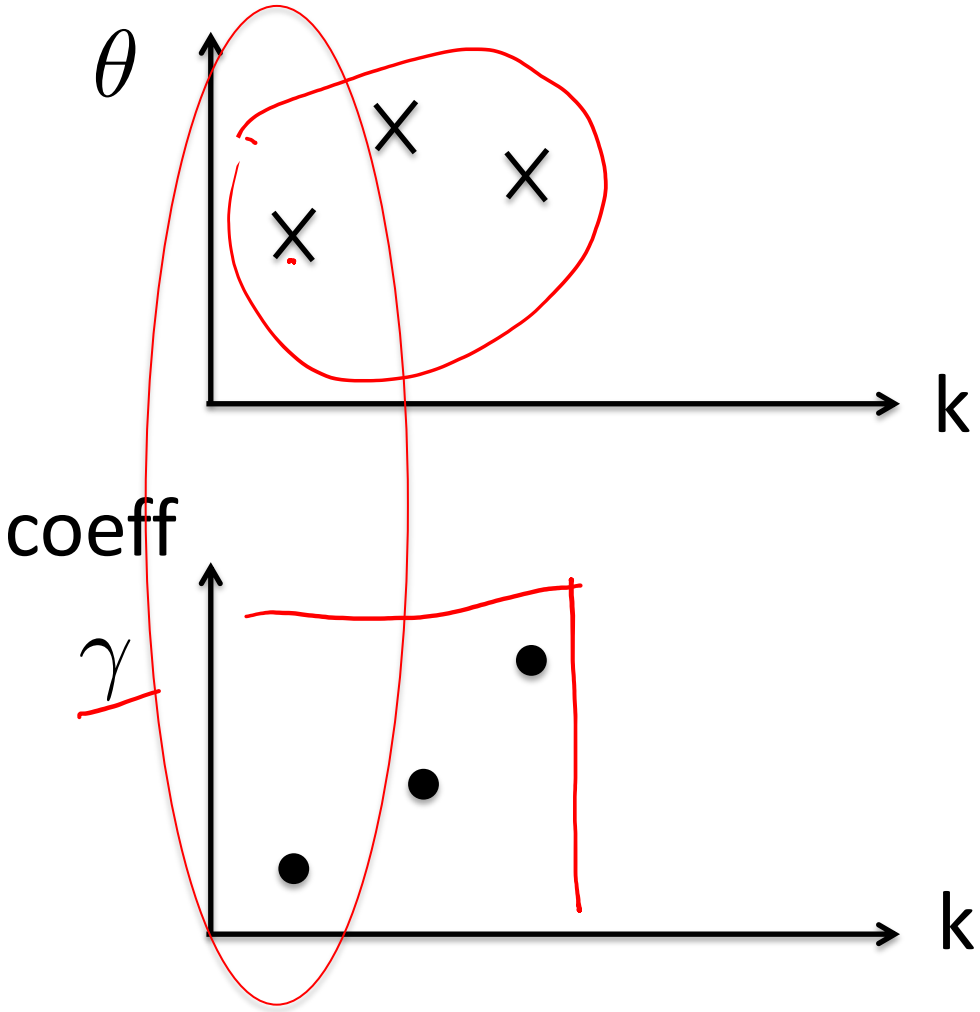
- Let's apply EMA to **square gradient** values

$$S_k = \gamma S_{k-1} + (1-\gamma)(\nabla_{w\,k})^2$$

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{S_k + \epsilon}} \nabla_{w\,k}$$

Hyperparameters = α, γ; ϵ

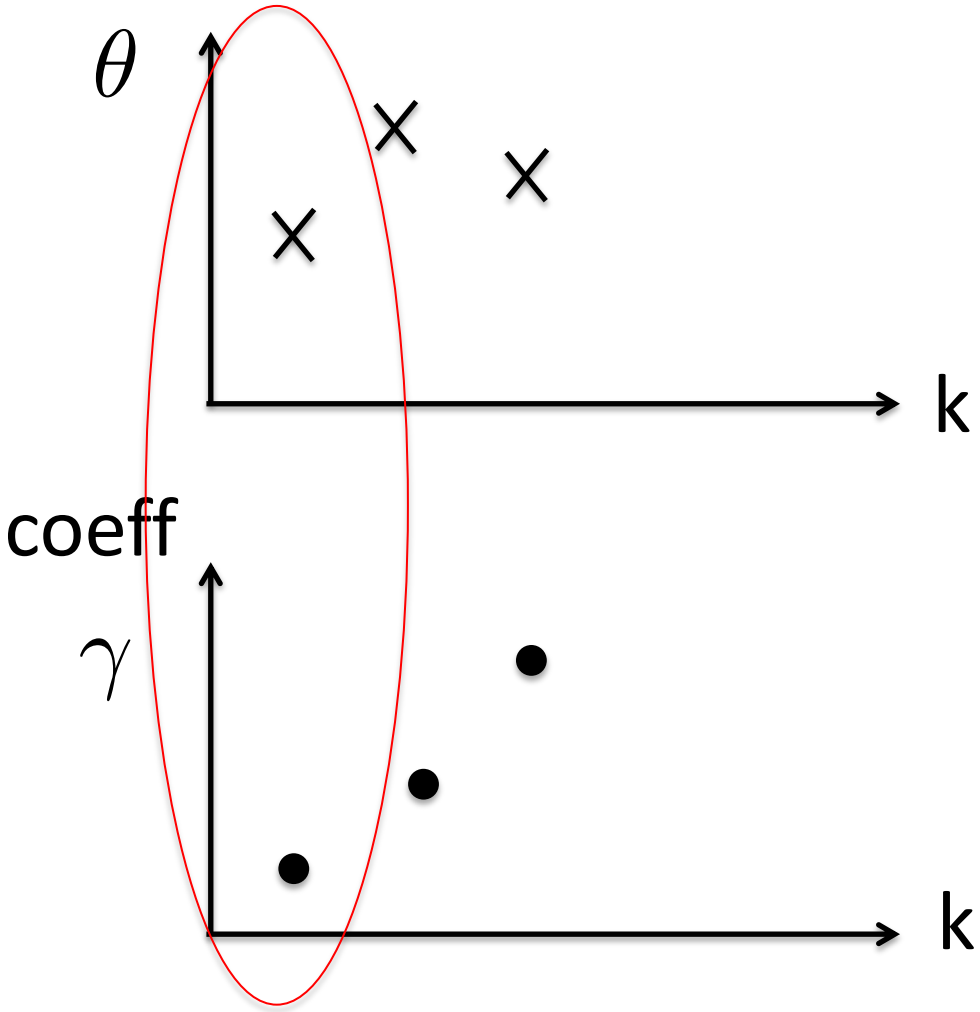
Biases in EMA



$$v_0 = 0$$

Exponentially Moving Average

Biases in EMA Correction



$$v_0 = 0$$

Correction:

$$\tilde{v}_k = \frac{v_k}{1 - \gamma^k}$$

small k small

1 k large

Adam

- What if we combine Momentum with RMSProp

$$\check{v}_k = \gamma_1 \check{v}_{k-1} + (1-\gamma_1)\nabla_{w\,k}$$

$$S_k = \gamma_2 S_{k-1} + (1-\gamma_2)(\nabla_{w\,k})^2$$

- And do bias correction

$$\check{v}_k^c = \frac{\check{v}_k}{1-\gamma_1^k}$$

$$S_k^c = \frac{S_k}{1-\gamma_2^k}$$

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{S_k^c} + \epsilon}\check{v}_k^c$$

Hyperparameters = $\alpha, \gamma_1, \gamma_2; \epsilon$

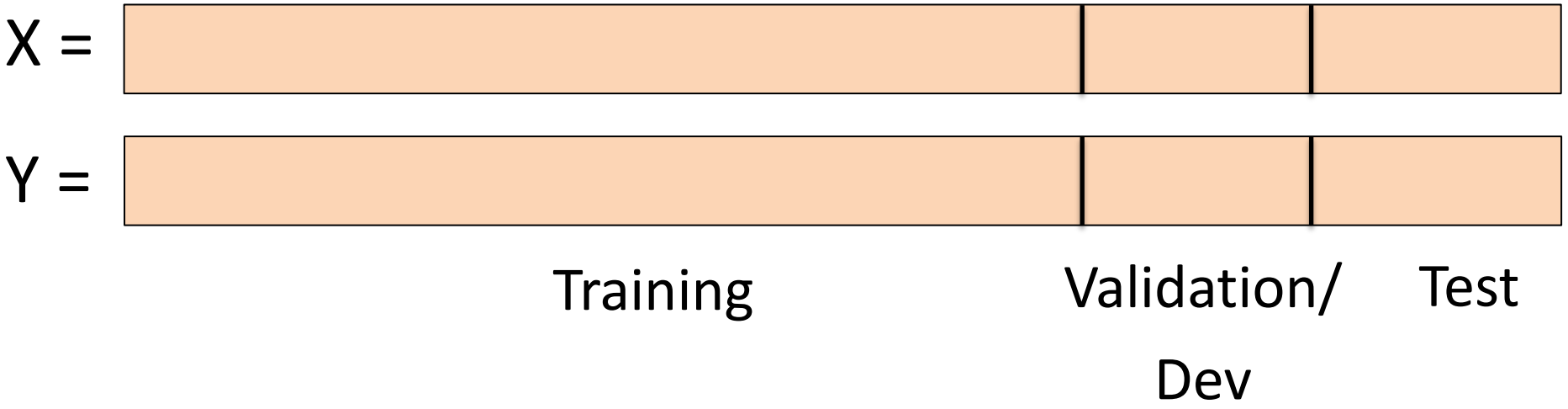

DISPLAY TOOLTIP
EMA(10)
$750
$500
$250
$0
$-250
May 19
Jul 19
Sep 19
Nov 19
Jan 20
Mar 20
May 20
Jul 20
Sep 20
Nov 20
Jan 21
Mar 21
EMA

2019

2021

Supervised Learning

Dataset D =



- Plausible divisions
- Distributions of each set

Over-fitting

- **Discrepancy** between

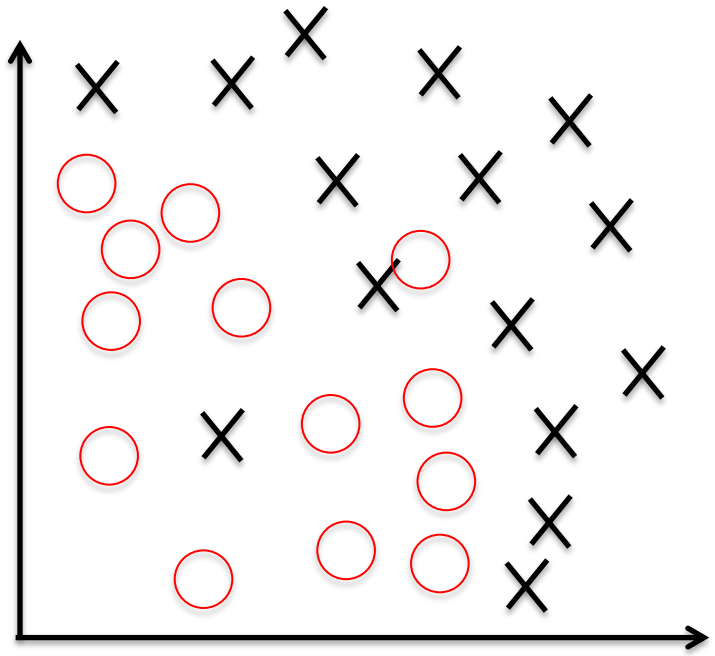
  Training performance and

  Testing performance

High **variance** – over-fitting

High **bias** – under-fitting

High Variance vs High Bias



Gradient Descent

$$\vec{w}_{k+1} = \vec{w}_k - \alpha \nabla_{\vec{w}} J(\vec{w}_k; b)$$

$$\vec{b}_{k+1} = \vec{b}_k - \alpha \frac{\partial}{\partial b} J(\vec{w}; b_k)$$

- Assume $i = 1$, and initial $\vec{w}_0, b_0$
- How do we compute $\vec{w}_1, b_1$?

Our cost: $J = L(\hat{y}, y)$

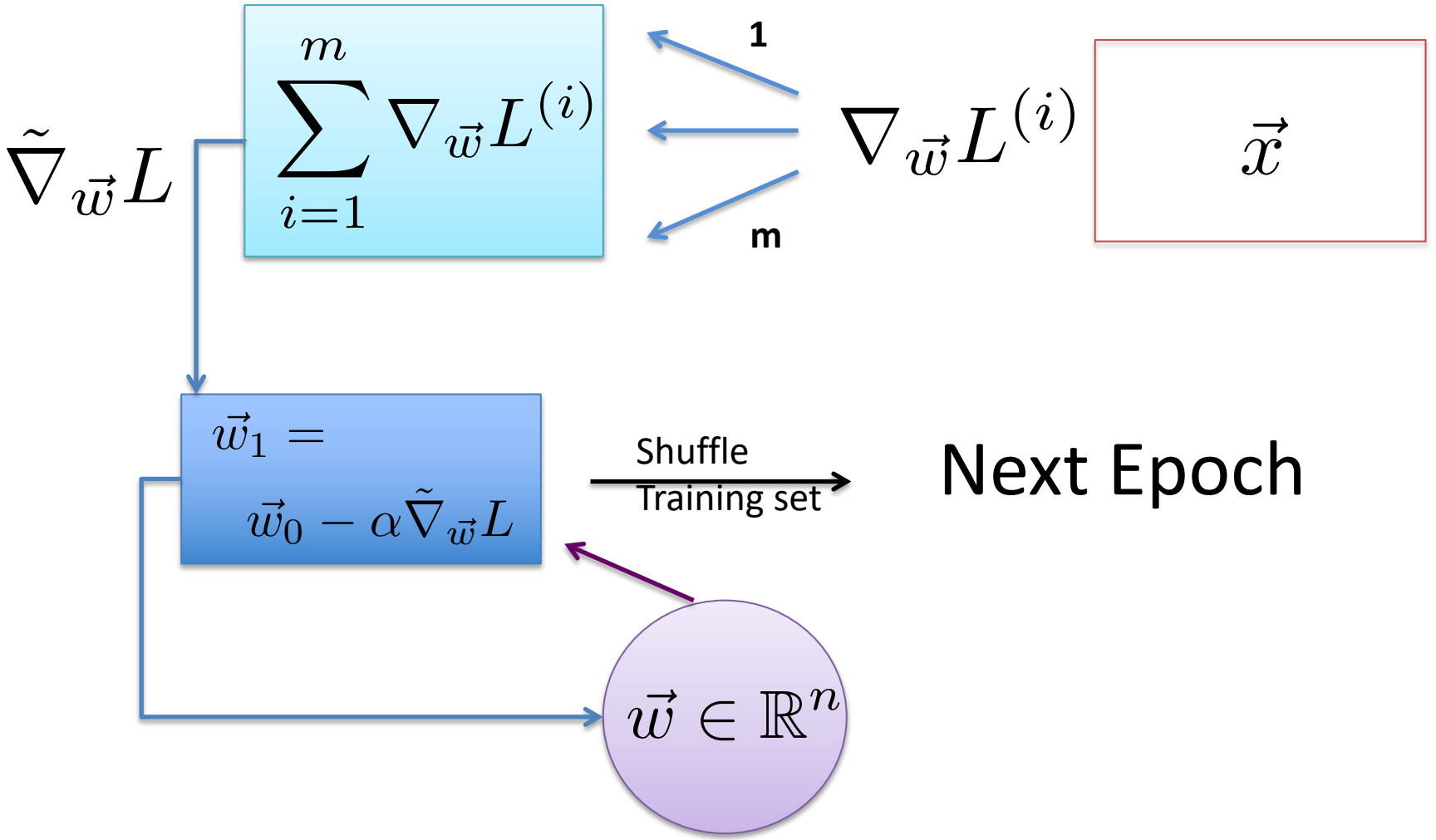
Back Propagation – Chain Rule

$$\nabla_{\vec{w}} L(\hat{y}, y) = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \nabla_{\vec{w}} z$$

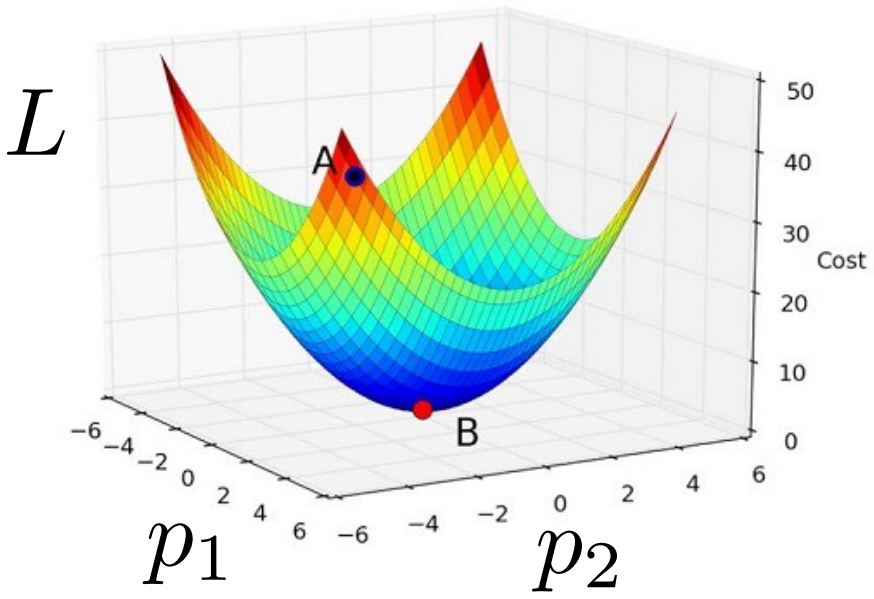
$z = \vec{w}^T \vec{x} + b$ → $\hat{y} = \sigma(z)$ → $L(\hat{y}, y)$

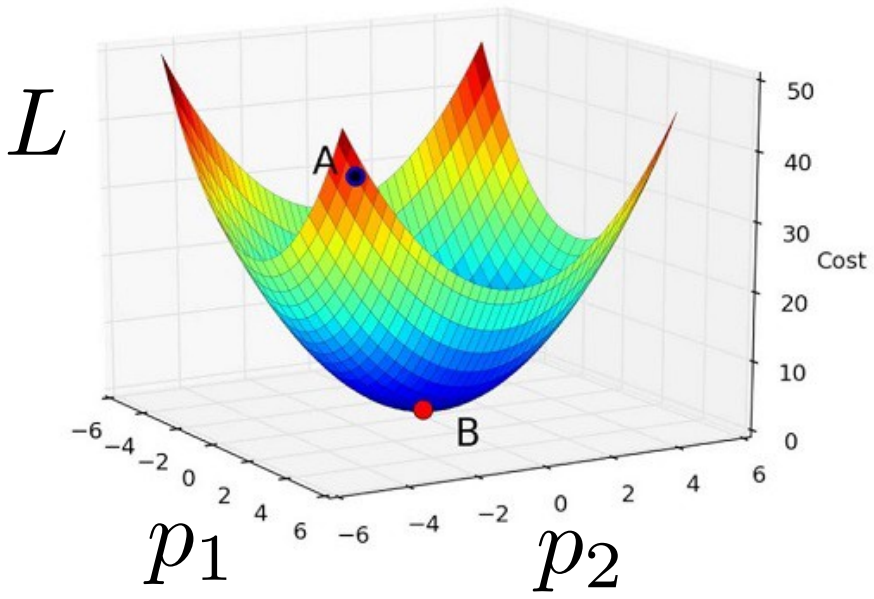
$\nabla_{\vec{w}} z$ ← multiply $\frac{\partial \hat{y}}{\partial z}(z)$ ← multiply $\frac{\partial L}{\partial \hat{y}}(\hat{y}, y)$

Graph


$\tilde{\nabla}_{\vec{w}} L$
$\sum_{i=1}^{m} \nabla_{\vec{w}} L^{(i)}$
1
m
$\nabla_{\vec{w}} L^{(i)}$
$\vec{x}$
$\vec{w}_1 = \vec{w}_0 - \alpha \tilde{\nabla}_{\vec{w}} L$
Shuffle
Training set
Next Epoch
$\vec{w} \in \mathbb{R}^n$

SGD Convergence





Journal of Mathematical Analysis and Applications
Volume 114, Issue 2, March 1986, Pages 512-527

On convergence of the stochastic subgradient method with on-line stepsize rules

Andrzej Ruszczyński *, Wojciech Syski

Almost surely convergence

Stochastic (Online) GD

$$w_{k+1} = w_k - \alpha \cdot \nabla_w J(w; x^{(i)}; y^{(i)})$$

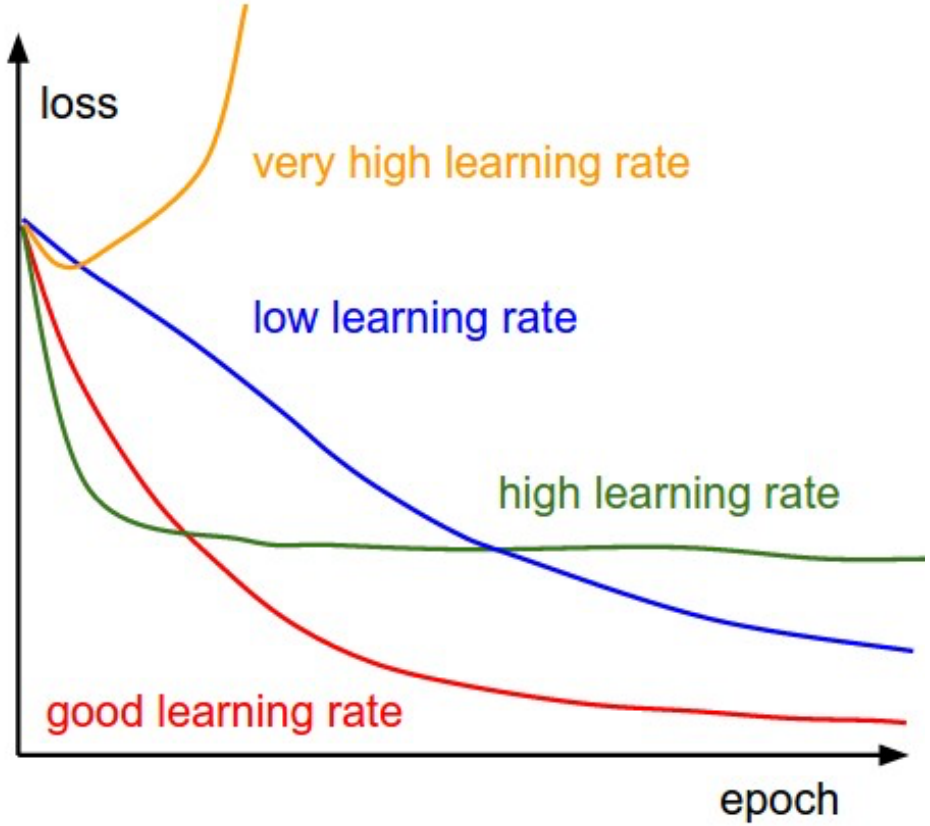
- Performs an update for **each training example $x^{(i)}$ and label $y^{(i)}$**
- The values of the loss and parameters will fluctuate.

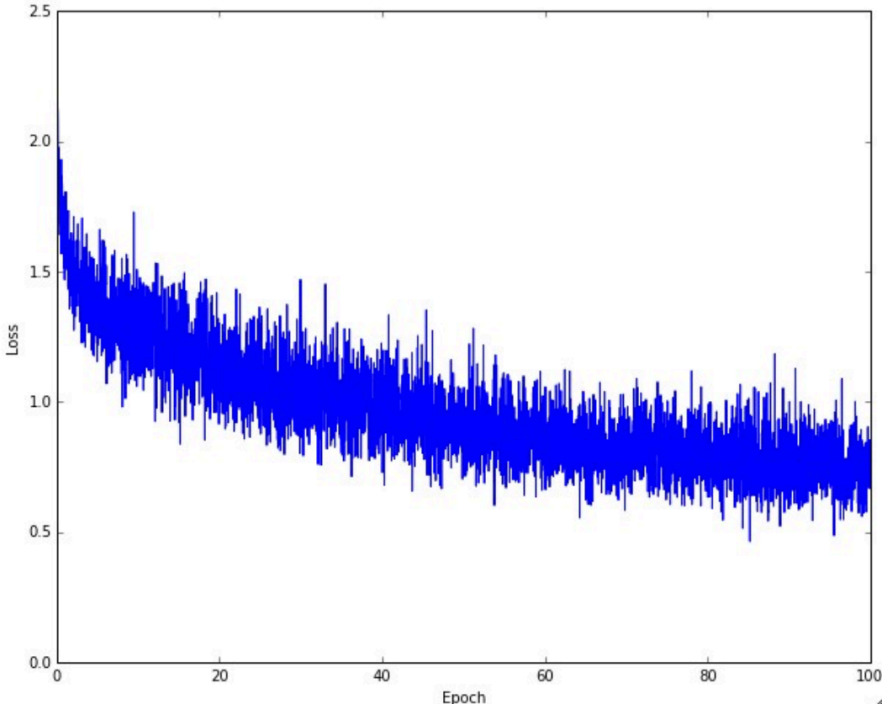
  (+) will discover better minimums

  (-) convergence to chosen minimum will keep overshooting
- **Learning rate** plays a very important role

Need Metrics

Experimenting with GD





Hyperparameters = α

Mini-batch GD

$$w_{k+1} = w_k - \alpha \cdot \nabla_w J(w; x^{(i:i+n)}; y^{(i:i+n)})$$

- Mini-batch GD is a hybrid method between GD and SGD.
- Performs an update for every mini-batch of **n training examples**.

(+) reduces the variance of the parameter updates

(+) efficient in computing the gradient w.r.t. a mini-batch

- mini-batch sizes range between 50-256.

Challenges

- Choosing a proper learning rate can be difficult
- Learning rate smart schedule
  - Annealing
  - Change of J below threshold
- Variable learning for different parameters
- Suboptimal local (saddle points)

Comparing GD variants

SGD	Mini-batch GD	Batch GD
mbsize = 1	**mbsize = m/R**	**mbsize = m**
(-) Can loose speedup from oscillations (-) hard to parallelize	(+) The whole minibatch is evaluated in parallel (+) Mostly consistent convergence	(+) Consistent convergence (-) Too long per iteration

Hyperparameters = mbsize (powers of 2, CPU/GPU memory)

Parameters

- **Parameters**

  Model Parameters:

  W, b, activation, output, cost

- **Hyper-parameters**

  External Parameters:

  Batch/minibatch size

  Learning parameters

Learning rate decay

$$\alpha = \frac{1}{1 + decr \cdot epnum} \alpha_0$$

$$\alpha = d^{epnum} \cdot \alpha_0$$

$$\alpha = \frac{d}{\sqrt{epnum}} \cdot \alpha_0$$

Momentum Method

$$v_{k+1} = \gamma v_k + \alpha \cdot \nabla_w J(w_k)$$

$$w_{k+1} = w_k - v_{k+1}$$

Image 2: SGD without momentum

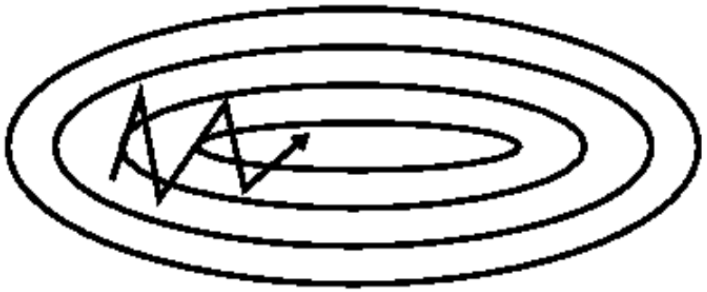


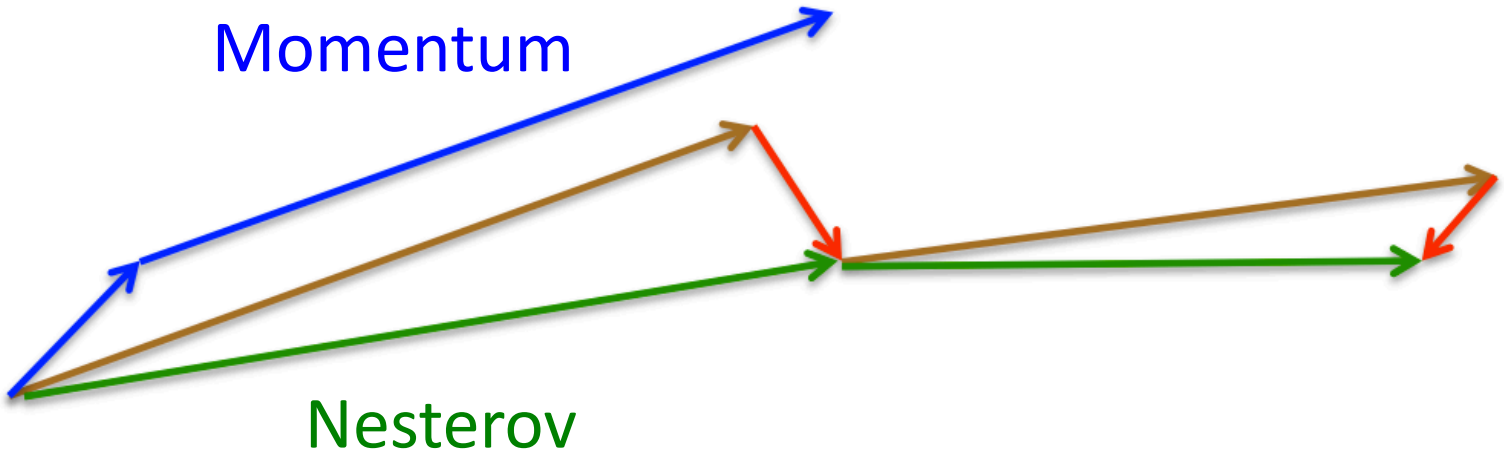
Image 3: SGD with momentum

$$\gamma \approx 0.9$$

Nesterov Accelerated Gradient

$$v_{k+1} = \gamma v_k + \alpha \cdot \nabla_w J(w_k - \gamma v_k)$$

$$w_{k+1} = w_k - v_{k+1}$$



Adagrad

$$w_{k+1,j} = w_{k,j} - \frac{\alpha}{\sqrt{G_{k,jj} + \epsilon}} \cdot g_{k,j}$$

- Adagrad uses a different learning rate for every parameter w_j at every step k. G is diagonal matrix of sum squared gradient values.

- Performs **smaller updates** (i.e. low learning rates) for parameters associated with **frequently occurring features**, and **larger updates** (i.e. high learning rates) for parameters associated with **infrequent features**.

RMSProp

$$E[g^2]_k = \gamma E[g^2]_{k-1} + (1-\gamma) g_k^2$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{E[g^2]_k + \epsilon}} g_k$$

- Prevents accumulation by adding regularizing term in the running average (exponentially decaying)
- Beneficial for RNNs

Adadelta

$$E[\Delta w^2]_k = \gamma E[\Delta w^2]_{k-1} + (1-\gamma)\Delta w_k^2$$

$$RMS[\Delta w]_k = \sqrt{E[\Delta w^2]_k + \epsilon}$$

$$\Delta w_k = -\frac{RMS[\Delta w]_{k-1}}{RMS[g]_k} g_k$$

$$w_{k+1} = w_k + \Delta w_k$$

- Generalizes RMSProp / Adagrad for considering RMS instead of accumulation of grad
- No learning rate parameter

AdaM- Adaptive Moment Estimation

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{v}_k} + \epsilon} \hat{m}_k$$

- Keeps track of 2 moments: mean and variance
- Normalizes them to prevent biases

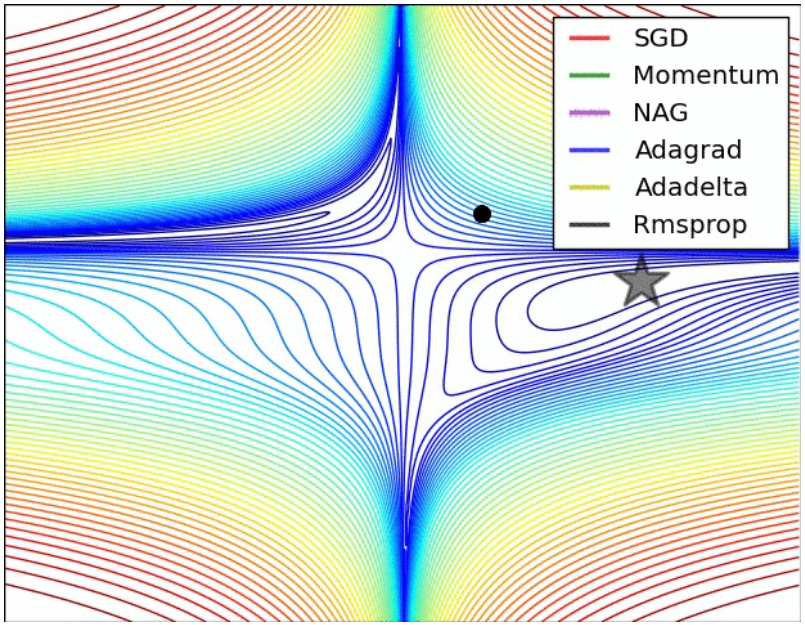
$$\hat{m}_k = \frac{m_k}{1 - \beta_1^k}$$

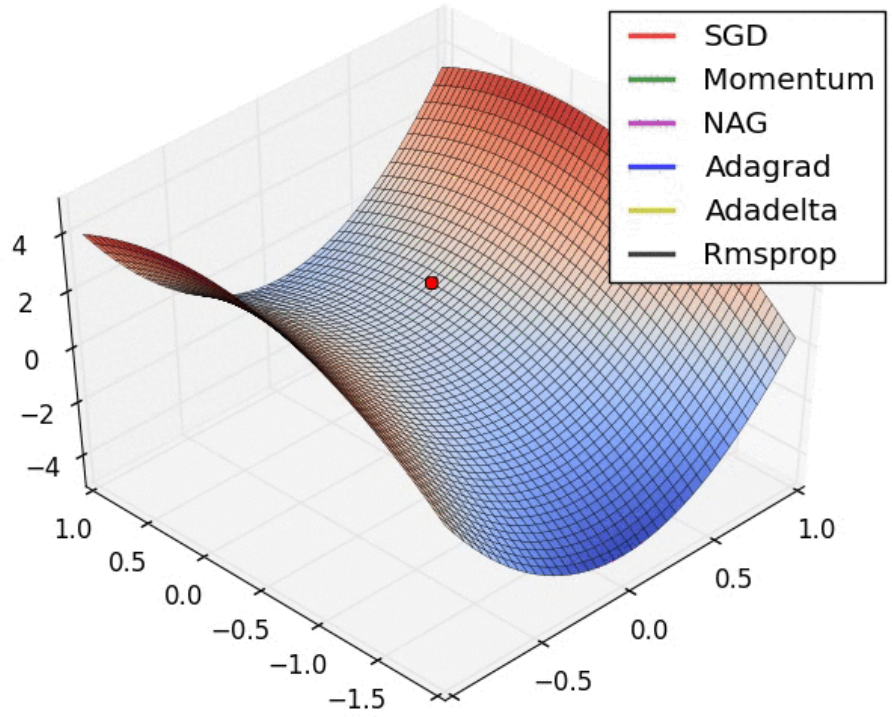
$$\hat{v}_k = \frac{v_k}{1 - \beta_2^k}$$

Additional

- AdaMax
  - Generalization of AdaM to L-inifnity norm
- Nadam
  - Nesterov Adam
- AMSgrad
  - Max normalization instead of exponenetial in Adam

Visual Comparison





Which optimizer to use?

- RSMProp & AdaDelta **adaptive**
- Adam **adaptive + momentum -> robust**
- SGD as **a first pass**

The Right Optimization is Key