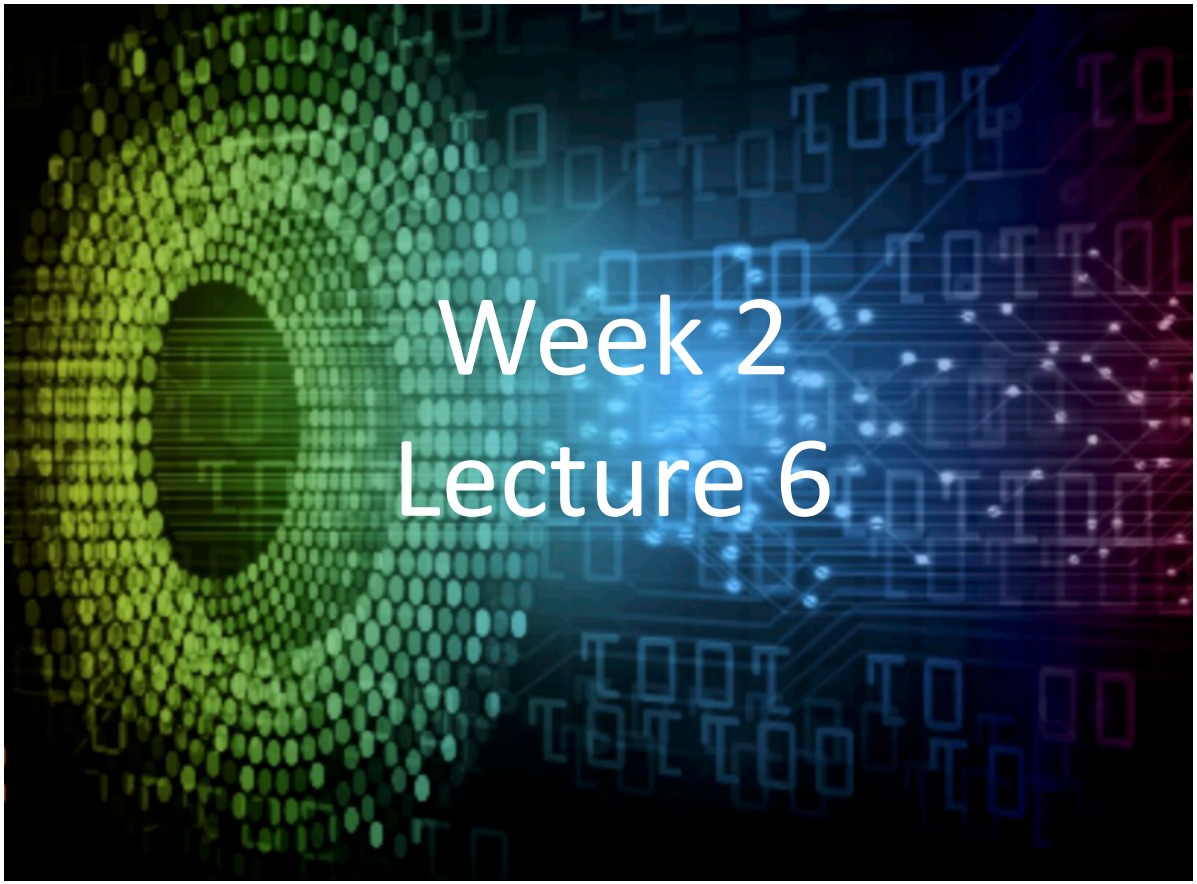
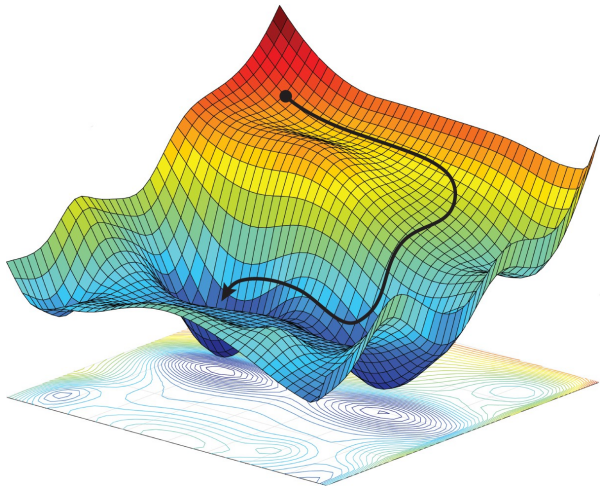
Introduction to Deep Learning
Applications and Theory



ECE 596 / AMATH 563

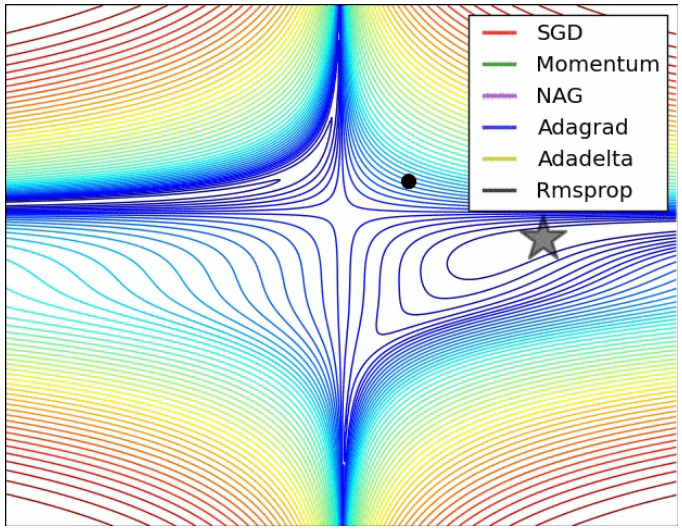
Previous Lecture: **Optimization and Gradient Descent**

1. Gradient Descent
2. Training the Logistic Regression Loss
3. Classification/ Regression with NN



Current Lecture: **Network Training/Optimization**

1. Stochastic Gradient Descent
2. Extensions
3. Training NN



Gradient Descent

learning rate

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

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

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

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

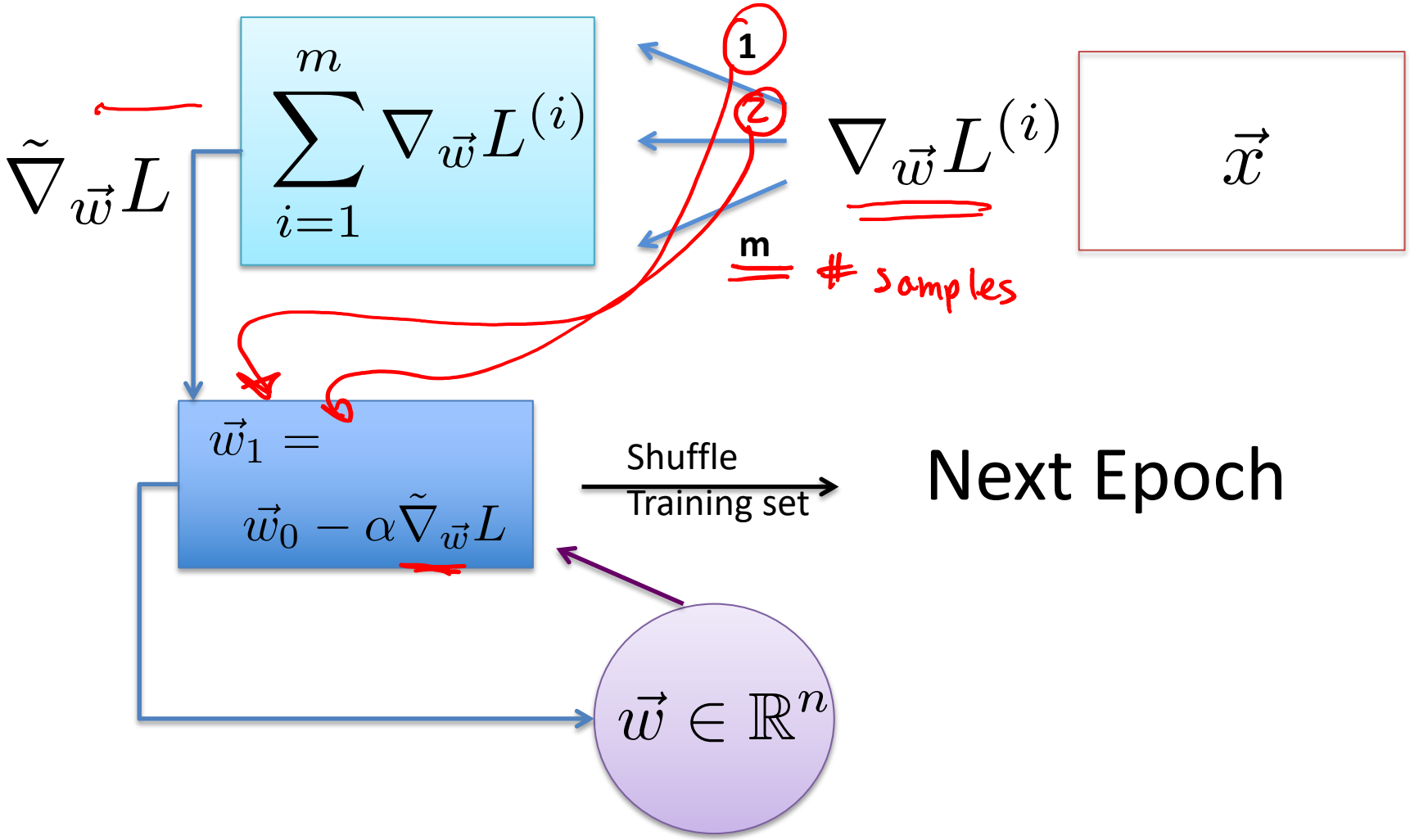
Back Propagation – Chain Rule

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

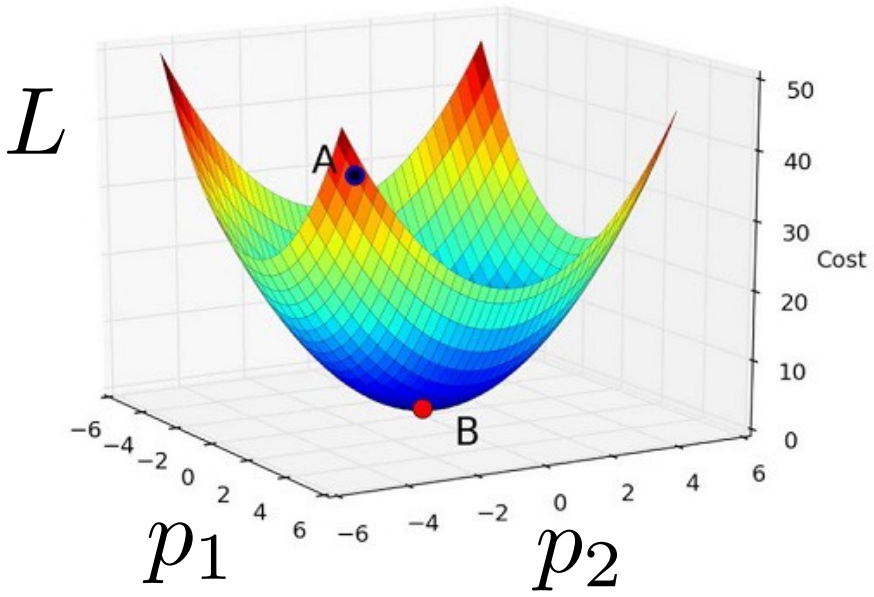
$z = \vec{w}^T \vec{x} + b$ → $\hat{y} = \sigma(z)$ → $L(\hat{y}, y)$

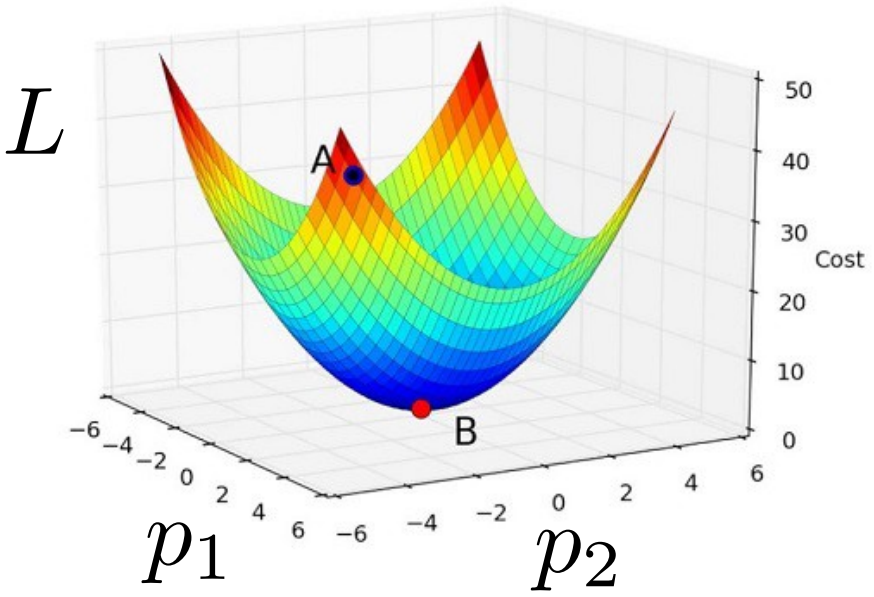
$\frac{\partial L}{\partial \hat{y}}(\hat{y}, y)$ **multiply** $\frac{\partial \hat{y}}{\partial z}(z)$ **multiply** $\nabla_{\vec{w}} z$

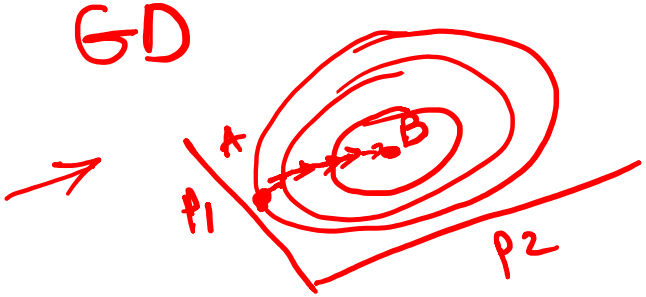
Graph

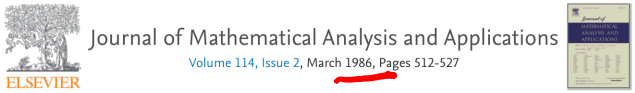

$\tilde{\nabla}_{\vec{w}} L$
$\sum_{i=1}^{m} \nabla_{\vec{w}} L^{(i)}$
1
2
$\nabla_{\vec{w}} L^{(i)}$
$\vec{x}$
m
samples
$\vec{w}_1 = \vec{w}_0 - \alpha \tilde{\nabla}_{\vec{w}} L$
Shuffle
Training set
Next Epoch
$\vec{w} \in \mathbb{R}^n$

SGD Convergence









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

Andrzej Ruszczyński *, Wojciech Syski

Almost surely convergence

⇒ neurips 2020

Stochastic (Online) GD

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

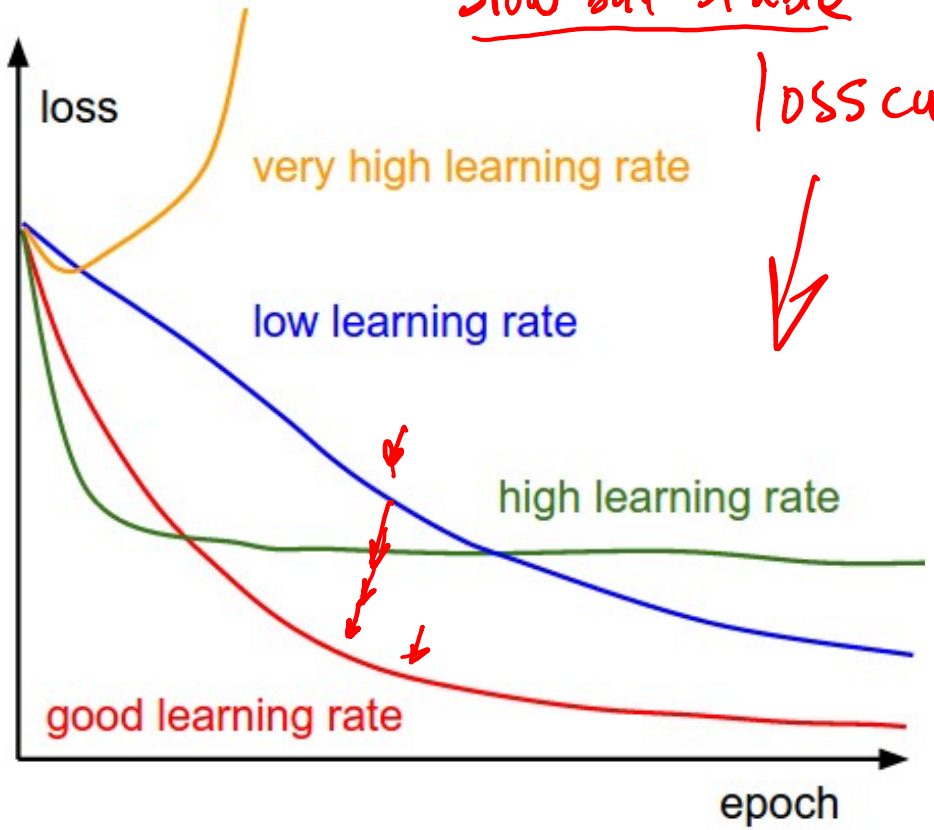
- Performs an update for **each training example $x^{(i)}$ and label $y^{(i)}$**
- The values of the loss and parameters will fluctuate.

  (+) will discover better minimums

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

Need Metrics

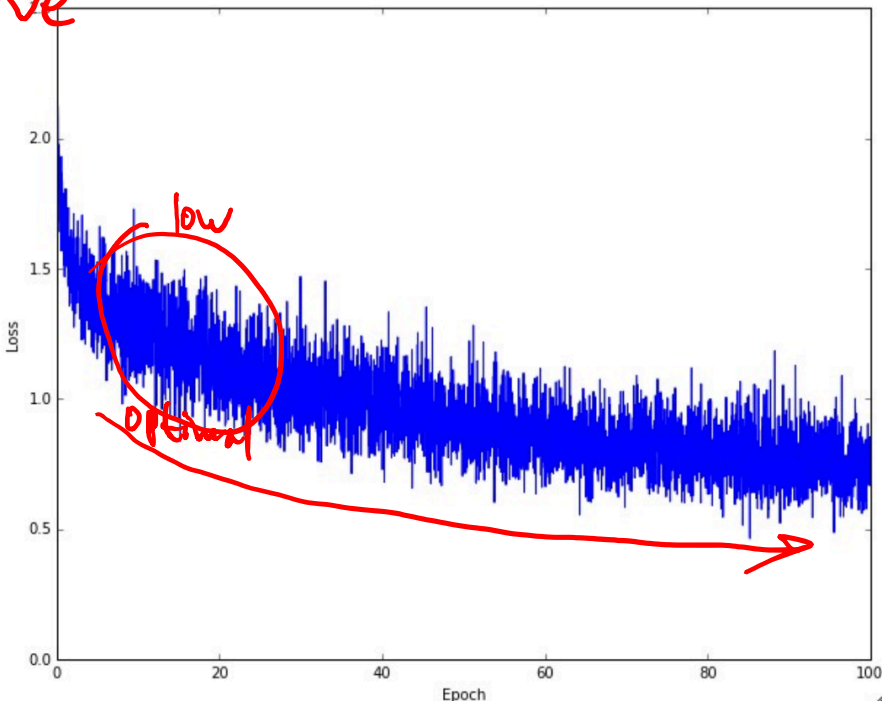
Experimenting with GD



Slow but stable

loss curve

Fluctuating



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)

extensions

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

1. Learning rate decay

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

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

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

2. Momentum Method

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

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

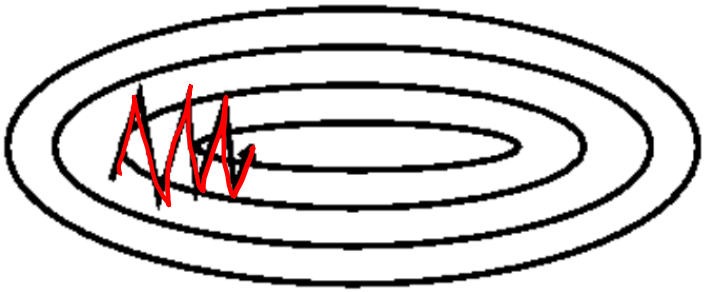


Image 2: SGD without momentum

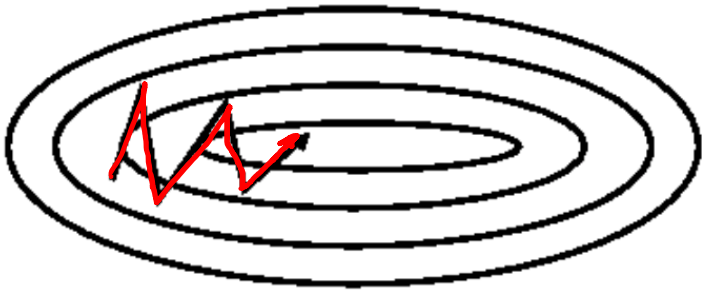


Image 3: SGD with momentum

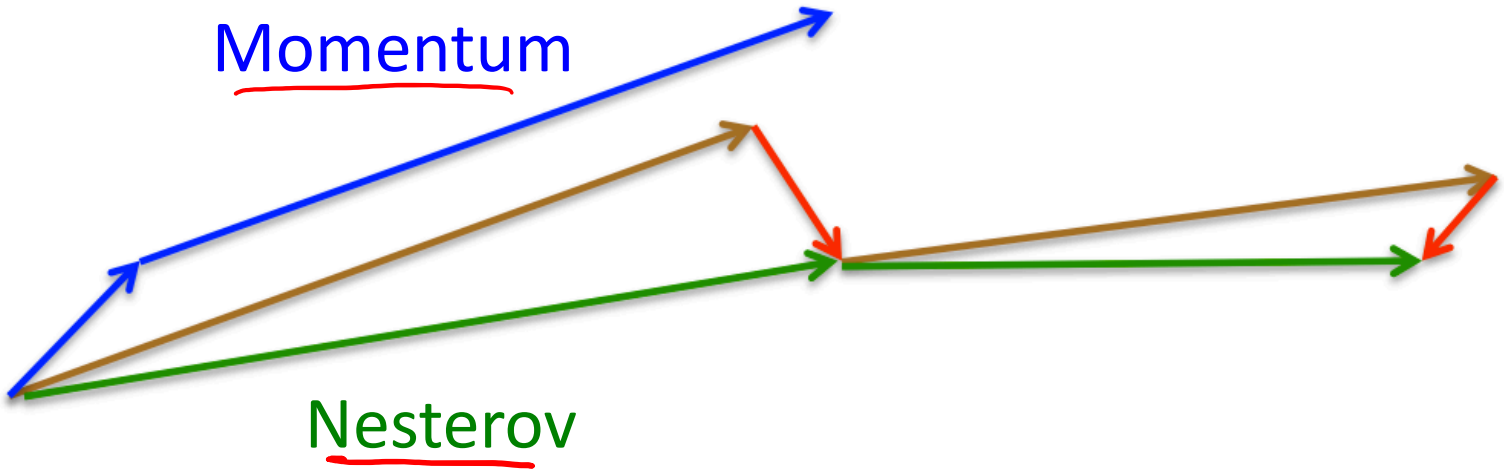
$$\gamma \approx 0.9$$

3. Nesterov Accelerated Gradient

(NAG)

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

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



4. Adagrad

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

gradient

- 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**.

5. 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

6. 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

7. 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

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

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

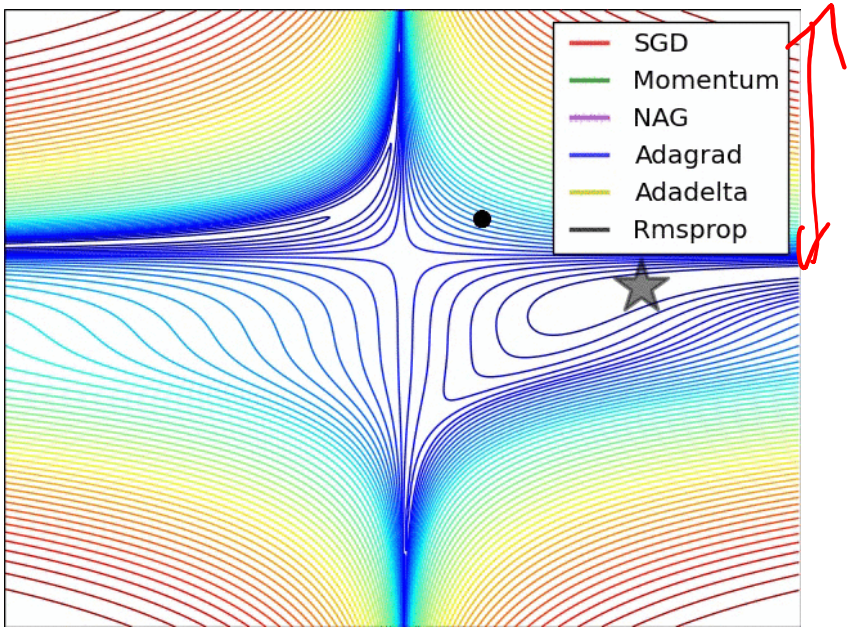
Additional

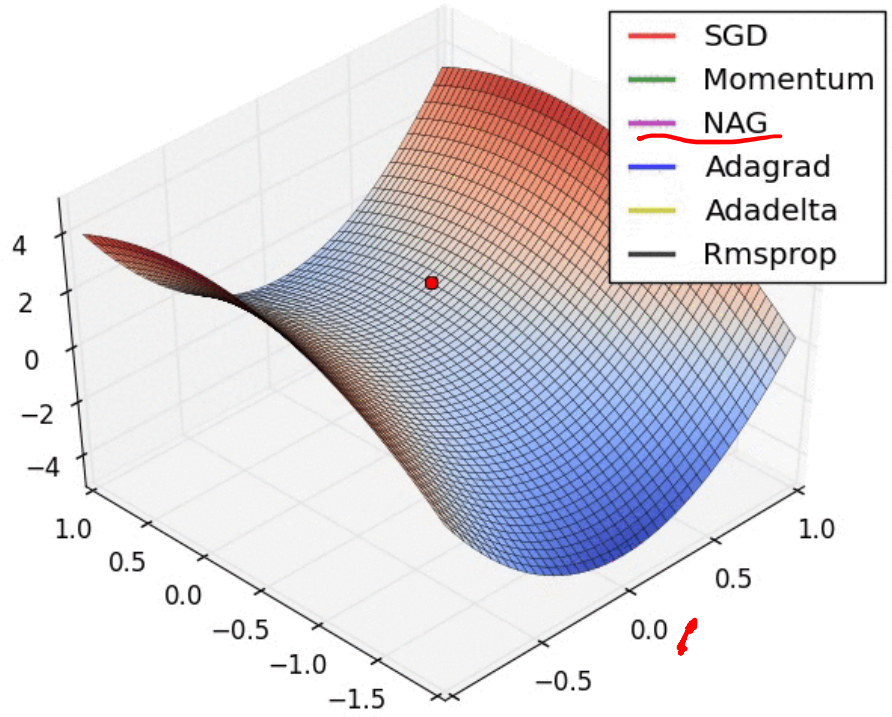
- 8. AdaMax
  - Generalization of AdaM to L-inifnity norm

- 9. Nadam
  - Nesterov Adam

- 10. 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

Lab This Week

The MNIST Dataset

- Adjust your setup to achieve a testing accuracy of 90% or better

Next Week - **Deep Learning Practices**

- Practical Optimization
- Operators
- Drop out
- Initialization
- Normalization
- Project Cycle with Deep Learning Methodology
- Introduction to CNN