Optimization for machine learning

Gradient

Optimization

Training

Loss Function

- $\sum_{x,y} l(x, y; w)$
  - x,y from training dataset
  - w model parameters
- Alternative viewpoint
  - Function that transforms x close to y
  - $f(x; w) \approx y$
- $\sum_{x,y} g(f(x; w), y) = \sum_{x,y} l(x, y; w)$
  - Function g captures penalty for not being perfect

-ln$\frac{1}{1+e^{-wxy}}$

$(y - g(x; w))^{\,2}$

$\max(0, 1 - wxy)$

$$\min_{w} \sum_{xy} \ell(x, y; w)$$

Predictions

- Let x be a new data point
  - Not in train or validate
- Compute probability distribution f(x;w*)
  - Probability vector
  - Predicted class

$$\max_i f_i(x; w^*)$$

Brewers	Cardinals	Cubs	Pirates	Reds
0.20	0.20	0.43	0.13	0.04

Loss Function

- KL divergence
  - View labels as probability distributions by means of one-hot encoding
  - This is empirical distribution Q
  - Often non-point distribution
- Machine learning model yields distribution $P = f(x; w)$
  - In logistic regression, probability
  - Otherwise softmax
- Want P to be close to Q
- Kullback–Leibler divergence

$$\xcancel{\frac{z_i}{\sum_j z_j}} \qquad \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$KL(Q||P) = \sum_i Q(i) \cdot \log \frac{Q(i)}{P(i)}$$

Training

KL Loss Function

- Not symmetric
- Non-negative
  - Follows from Jensen’s inequality with r.v. Q and function = log Q/P
- Equal to zero if and only if P=Q (almost surely)

$$-\sum_i Q(i)\cdot \log\frac{Q(i)}{P(i)} = \sum_i Q(i)\cdot \log\frac{P(i)}{Q(i)} \leq \log\sum_i Q(i)\,\frac{P(i)}{Q(i)} = 0$$

Training

Loss Function

$$\min_{w} \sum_{\{x,y\}} KL(Q(y)||P(x;w)) = \min_{w} \sum_{\{x,y\}} KL(Q(y)\big|\big|f(x;w)\big) =$$

$$\min_{w} \sum_{\{x,y\}} \sum_{i} Q_i(y) \log(Q_i(y)/ f_i(x;w))$$

$$\min_{w} -\sum_{x,y} \sum_{i} Q_i(y) \log f_i(x;w)$$

$$\max_{w} \sum_{x,y} \sum_{i} Q_i(y) \log f_i(x;w)$$

- Match P with Q

Loss Functions

- L2 loss
  - Regression

$$\sum_i (y_i - f(x_i; w))^2$$

- Cross-entropy with softmax
  - KL divergence
    - Softmax

$$P_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$\min_w - \sum_{x,y} \sum_i Q_i(y) \log P_i(x; w)$$

  - Max likelihood
    - Cross entropy
    - Same as KL

$$H(Q, P) = -\sum_i Q_i \log P_i$$

Binary Case

- Two classes

$$QlogP_0 + (1 - Q)log(1 - P_o)$$

  - Q is either 0 or 1

$$logP_0 \text{ or } \log(1 - P_0)$$

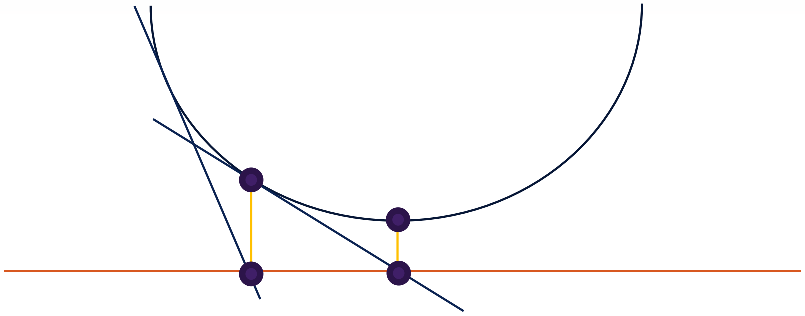
  - Same as maximum likelihood
    - Logistic regression

Optimization

Single Variate Newton

$$\min_{w} f(w)$$

$$w_{m+1} = w_m - f'(w_m)/f''(w_m)$$



Northwestern | ENGINEERING

Gradient Optimization

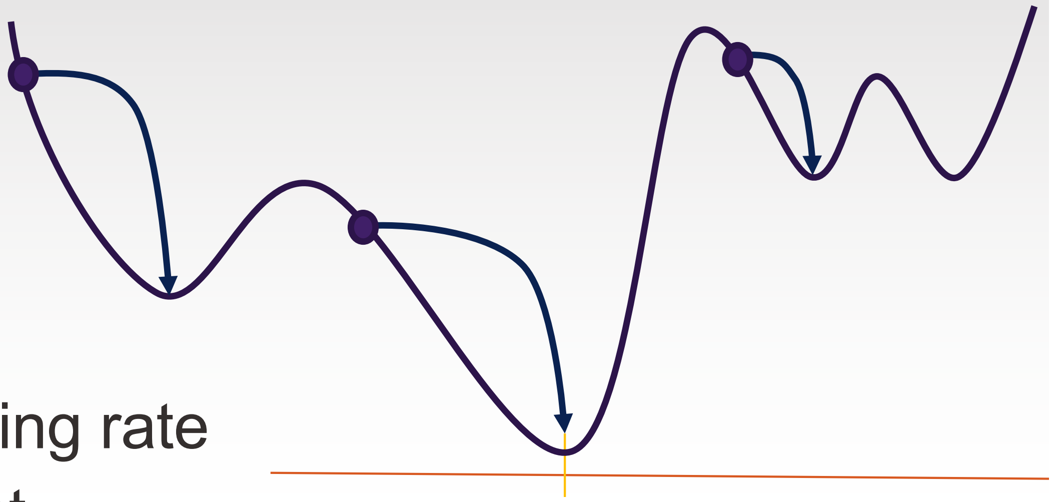
- Multiple variables
  - Derivate = gradient
  - Second derivate = Hessian

$$w_{m+1} = w_m - H(w_m)^{-1} \nabla f(w_m)$$

- What is the problem in the ML (loss minimization) context?
  - Size of Hessian grows
    - Computing it and taking the inverse

Gradient Optimization

$$w_{m+1} = w_m - t_m \nabla f(w_m)$$



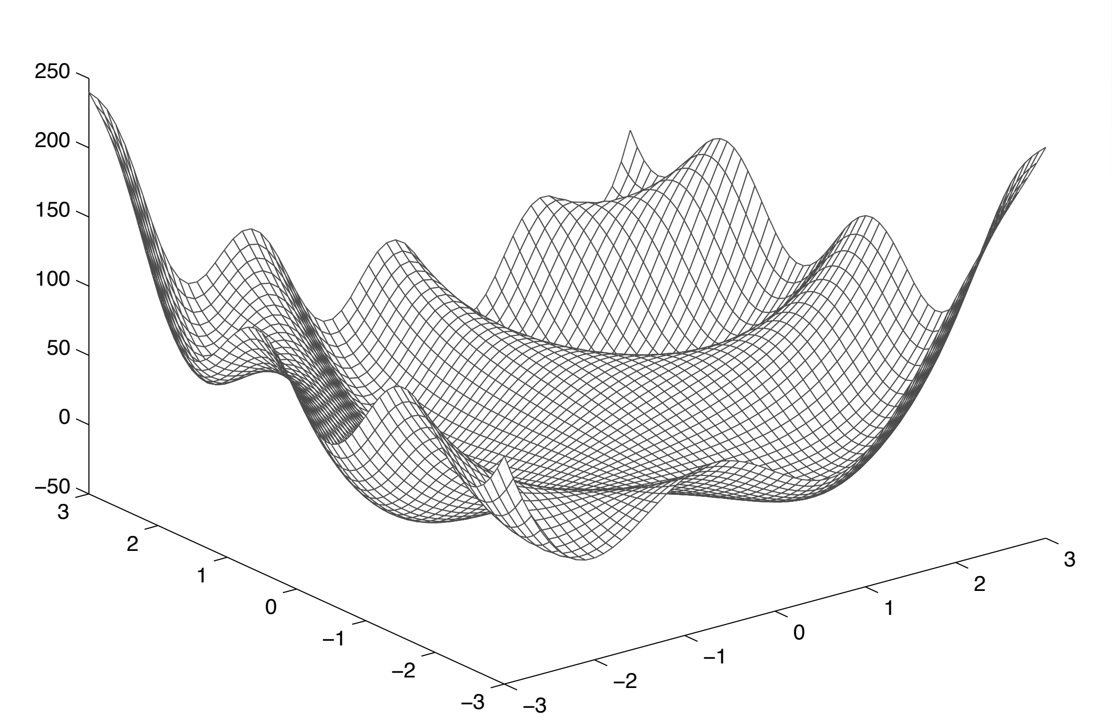
- Algorithm very sensitive to learning rate
- Starting point also very important
- Convergence
  - Certain learning rates lead to convergence
  - No guarantee to optimal solution
    - True only if function convex

Convergence Rate

- Let w^* be optimal
- Let w_m be current iterate
- Convergence
  - $\lim_{m \to \infty} w_m = w^*$?
  - $\lim_{m \to \infty} f(w_m) = f(w^*)$?
- Rate of convergence
  - $||w_m - w^*|| \leq \rho^m ||w_0 - w^*||$
  - $\rho < 1$ is convergence rate
- Studying
  - $||w_m - w_{m-1}|| \leq \rho ||w_{m-1} - w_{m-2}||$
  - Are iterates getting closer and closer

Hard Cases

- Many local minimums or maximums
- Depends on starting point



Optimization

Gradient

- Square error

$$\frac{\partial w}{\partial w_k} \sum_i (y_i - wx_i)^2 = -2 \sum_i x_{ik}(y_i - wx_i)$$

- Logistic regression

$$\frac{\partial w}{\partial w_k} \sum_i -ln(1 + \exp(-y_i w x_i))^{-1}$$

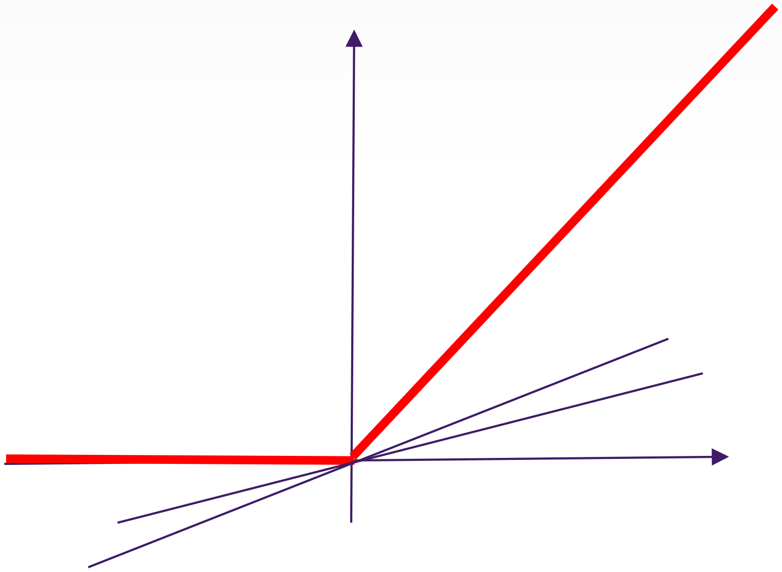
$$= \sum_i -y_i x_{ik} \exp(-y_i w x_i)\,(1 + \exp(-y_i w x_i))^{-1}$$

Optimization

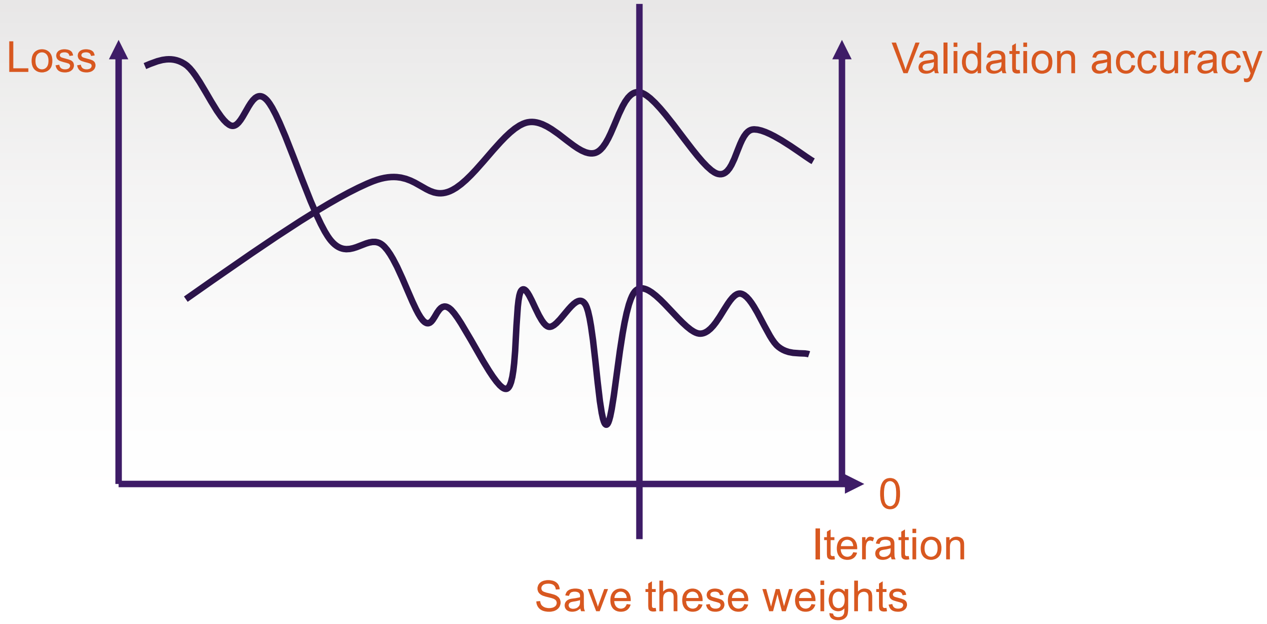
Gradient

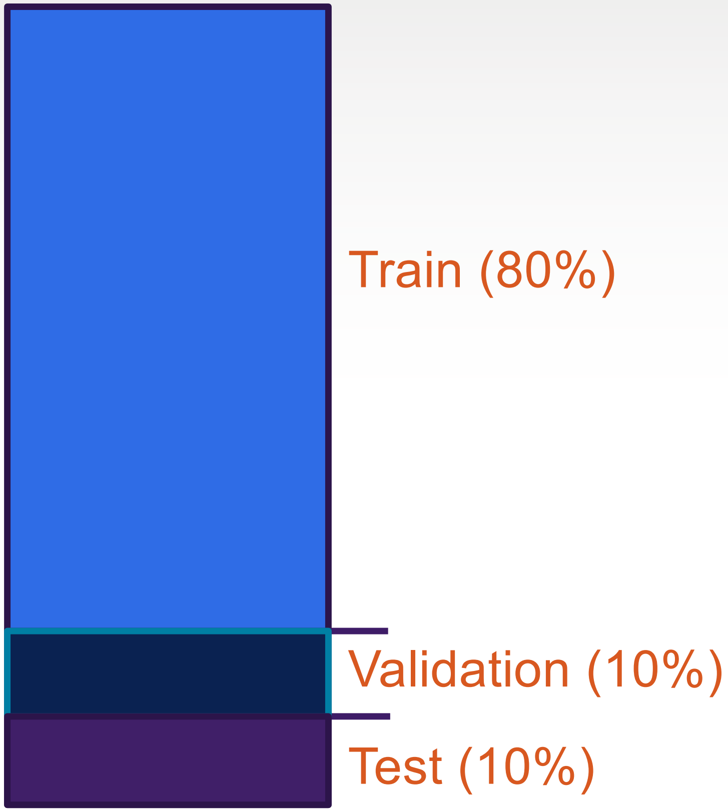
- Hinge loss

$$\frac{\partial w}{\partial w_k}\sum_i \max(0,1-y_i w x_i) = \sum_i \begin{cases} -y_i x_{ik} & y_i w x_i < 1 \\ 0 & y_i w x_i \geq 1 \end{cases}$$



Datasets and Loss Function

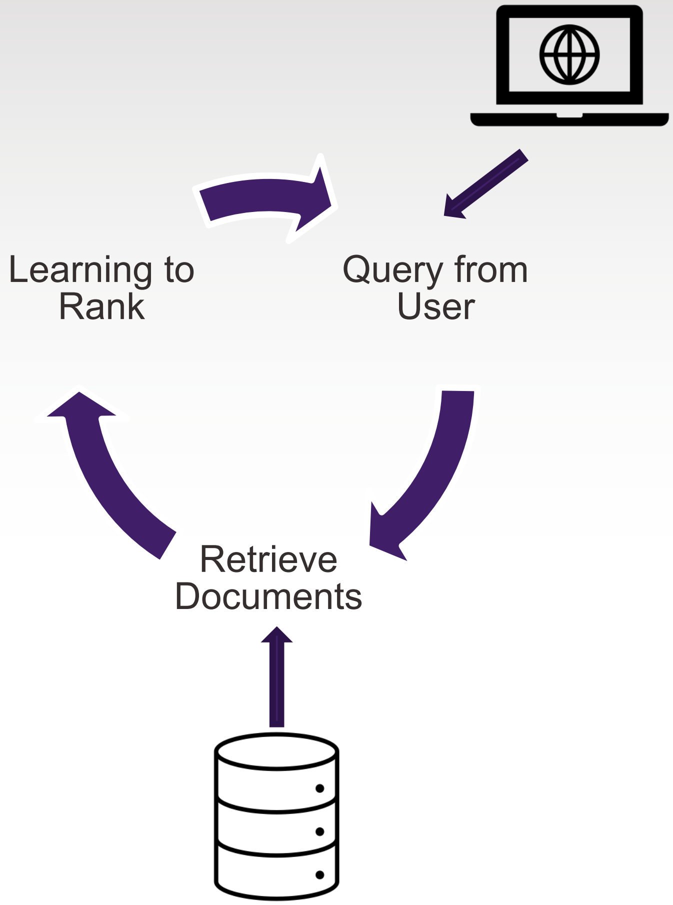




- Compute w from train + validate
- Use computed w* on test to evaluate
  - Use in the same way in production

Online Setting

- Examples
  - Starting a web search engine
    - No historical/training data
    - Train and inference at the same time
  - Patterns change frequently
    - Train and inference at the same time
  - Learning to rank
    - Order items
    - Individual preferences change frequently



Online Gradient

- Algorithm
  - Loop $t = 0,1,2,\cdots$
    - Get sample x_t
    - Make prediction $f_t(x_t, w_t)$
    - Observe label y_t
      - User acts
    - $w_{t+1} = w_t - \lambda \nabla l_t(x_t, y_t; w_t)$
- Comments
  - Gradient based on a single sample
  - Learning rate needs to be constant if process repeats many times

Quality

- Regret
  - Compare offline setting vs online setting
    - Get all samples at once vs one-by-one
- $R(T) = \sum_{t=1}^{T} l_t(x_t, y_t; w_t) - \min_{w} \sum_{t=1}^{T} l_t(x_t, y_t; w)$
- Regret should be small
- Hard to get low regret
  - Adversary to select the order of $x's$
- Under some conditions (convexity) gradient
  - $R(T) \leq {}^{c}/_{\sqrt{T}}$

Role of Convex Functions

Convexity

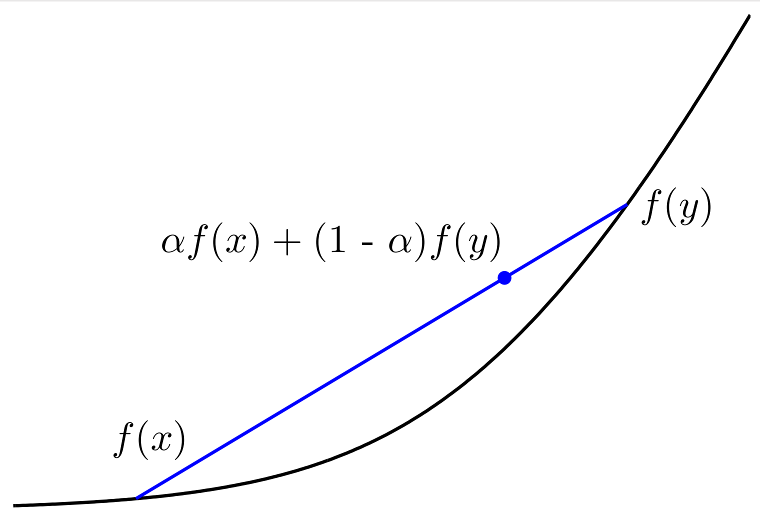
Convex Functions and Sets

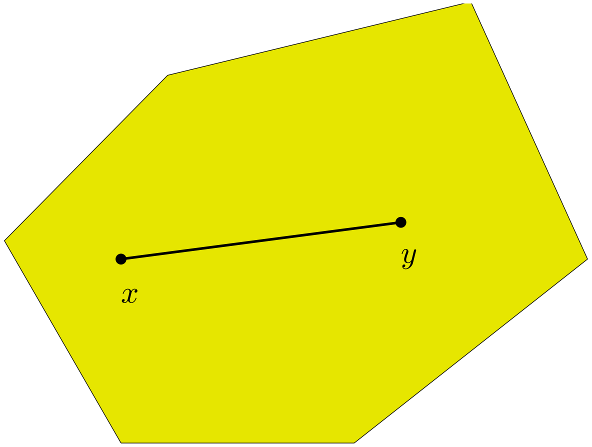
- Tangent above function
  - For any x,y and $1 \geq \lambda \geq 0$

$$\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda) y)$$

  - Hessian is positive semidefinite
- Set
  - For any $x \in S, y \in S, 1 \geq \lambda \geq 0$

$$\lambda x + (1-\lambda) y \in S$$



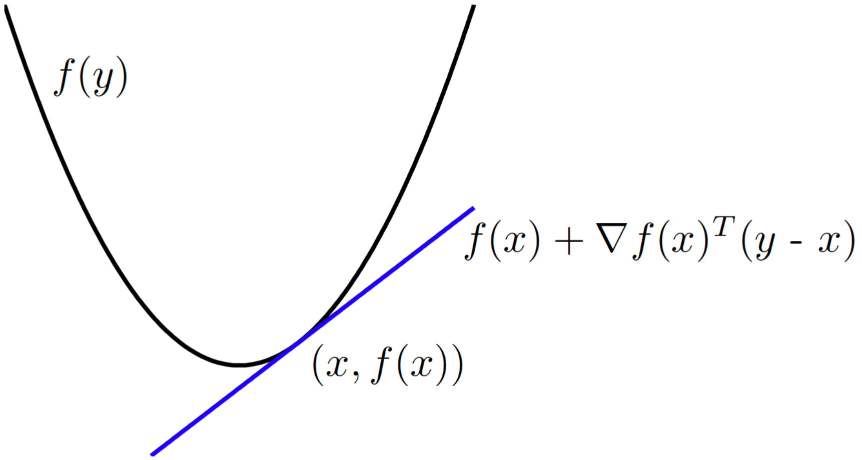


Sub-gradient

- Convex function always has a sub-gradient

$$f(y) \geq f(x) + \nabla f(x)(y - x) \quad \text{for every } y$$

- All function values are above the gradient line
  - Sub-gradient is a set
- Sub-gradient does not exist for general functions
- Single variate function
  - Convex if and only if
    - Second derivative is nonnegative
    - First derivative is increasing

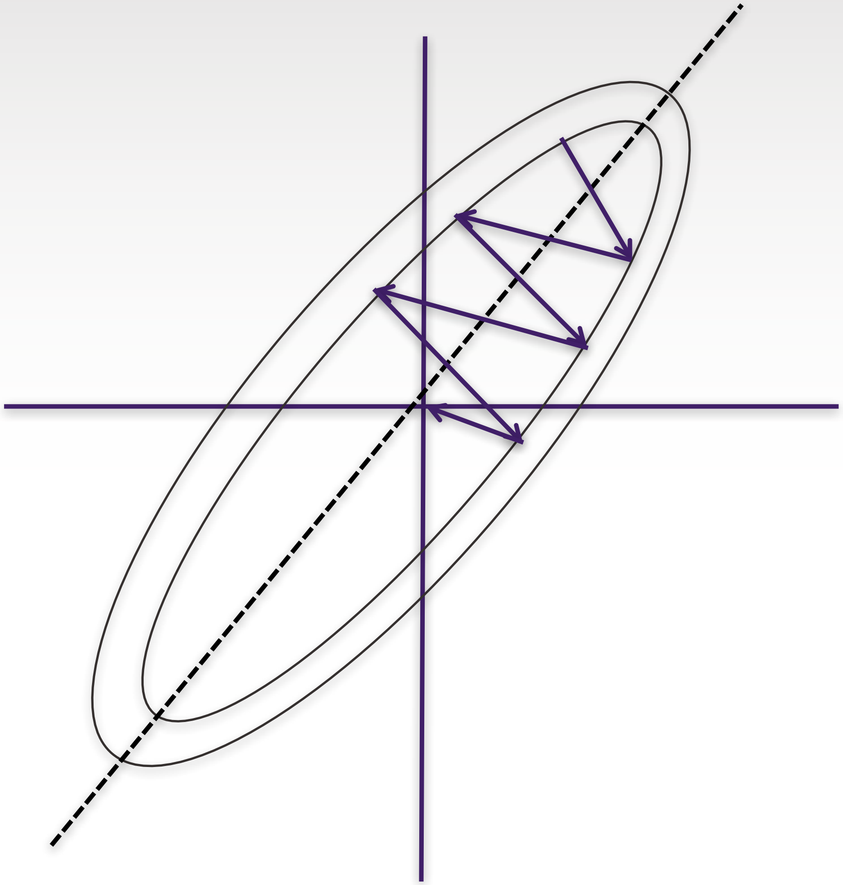


Convergence

- Convex function continuous in the ‘interior’
  - Counter example on the boundary?
- The sub-gradient (support vector) always exists
- Fairyland
  - Convex function has no local minimum or maximum
    - Every local point is also global
  - Bowl in 2D
- Most common loss functions are convex

Convergence on Level Curves

- Function is quadratic
- Level curves are ellipses



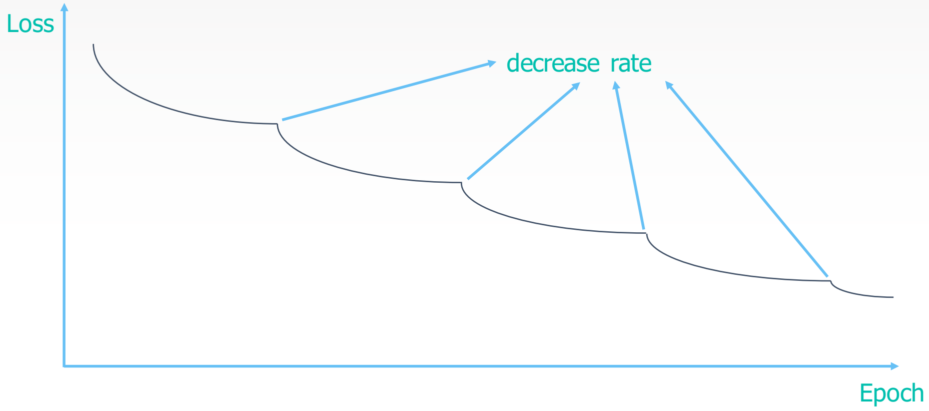
Optimization Convergence

$$w_{m+1} = w_m - t_m \nabla f(w_m)$$

- If $\sum_{m=1}^{\infty} t_m = \infty, \sum_{m=1}^{\infty} t_m^2 < \infty,$
  - Sequence converges to an optimal solution
  - Learning rate must go to zero
    - But not too quickly
      - Converge to a suboptimal point
- Typical rate $t_m = \frac{a}{b+m}$
  - Does not work well in practice

Learning Rate

- Every K iterations
  - Reduce the rate by factor f
- Works great in practice
  - If K and f chosen correctly
- Challenge
  - Selection of K and f
  - Trial-and-error



Stochastic Gradient Descent

SGD

Distributed

Gradient Optimization for ML

- Loss function involves all samples
  - Gradient is the sum of all gradients
  - Computationally intractable
- Solution
  - Randomly select subset S of samples
  - Compute the gradient for each selected sample
  - Average them
- Allows parallelization
  - Embarrassingly parallel

$$w_{m+1} = w_m - t_m \frac{1}{|s|} \sum_{i \in S} \nabla l(x_i, y_i; w_m)$$

Gradient Computation

- Often can be written by linear algebra matrix, vector operations
  - $w_m^i = \nabla l(x_i, y_i; w_m)$
- Instead of loop for all samples in S
  - All matrix operations can be written as tensor operations
  - Tensor = higher dimension than matrix
- Consequence
  - Pass to GPU w as tensor
  - All operations as tensor operations on GPU
    - Disguise fact that there are several samples

GPU

From Slides to Practice

```
Load all to memory
Loop
  Select S
  Send to GPU
```

```
Loop
  Read S from file
  Send to GPU
```

```
Loop
  Read large batch
            from file
  Loop
    Select S from batch
    Send to GPU
```

- Data stored in DB or csv
- Ten million records in csv

Python Yield

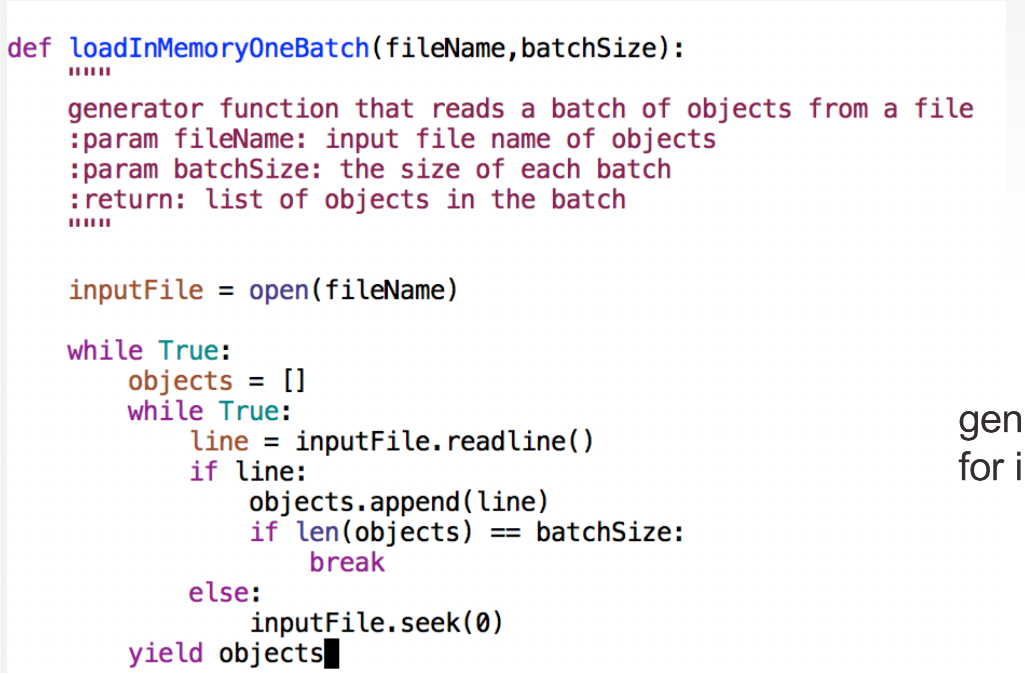
- Return from function
  - Destroy function stack/state
- Generators are iterators
  - You can iterate through them only once
- Yield in function
  - Retain state and continue from state in subsequent calls
  - As return but returns a generator

- getFib is essentially an infinite function

```
def getFib():
    number = 1
    while True:
        if isFib (number):
            yield number
        number += 1

gen = getFib
num = next(gen) # smallest Fibonacci number
    # getFib loops for un unspecified number of

    iterations
next(gen) # next Fibonacci number
```

Python Yield

- Large file
  - Want to read batches on a continuous basis

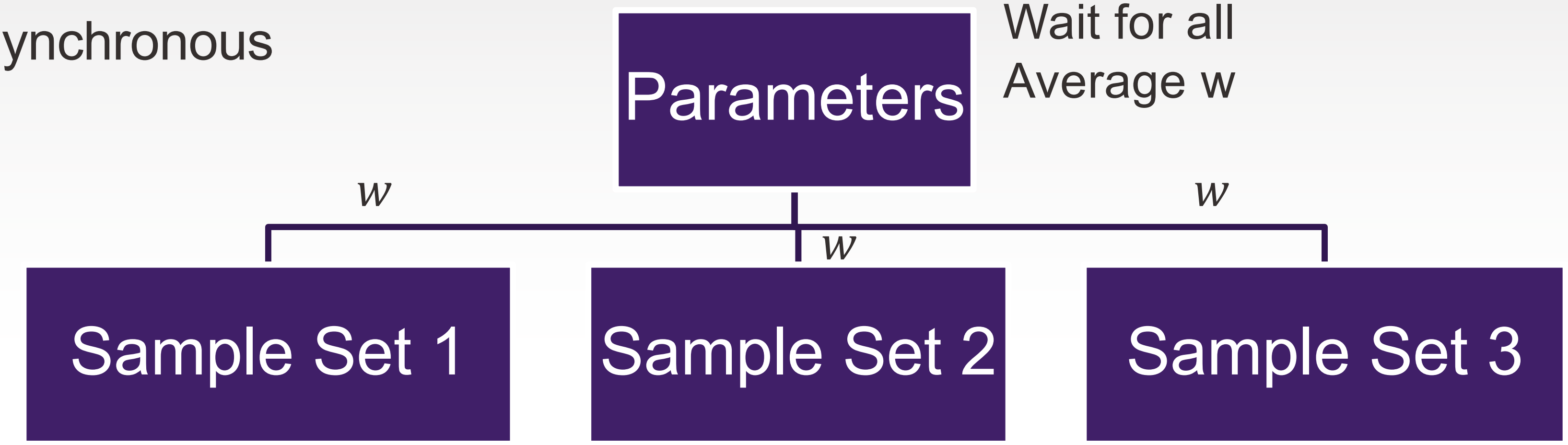


```python
def loadInMemoryOneBatch(fileName,batchSize):
    """
    generator function that reads a batch of objects from a file
    :param fileName: input file name of objects
    :param batchSize: the size of each batch
    :return: list of objects in the batch
    """

    inputFile = open(fileName)

    while True:
        objects = []
        while True:
            line = inputFile.readline()
            if line:
                objects.append(line)
                if len(objects) == batchSize:
                    break
            else:
                inputFile.seek(0)
        yield objects
```

```
gen = loadInMemoryOneBatch(“file.csv”,50000)
for i in range(100):
        data = next(gen)
```

Parameter Server

- Data distributed
  - Synchronous



  - Asynchronous – Wildhog!
    - Server updates parameters as soon as one is done

Pros and Cons

Full gradient

High per iteration time

Low number of iterations

Most stable

Very low variance

Stochastic gradient descent

Lower per iteration time

Higher number of iterations

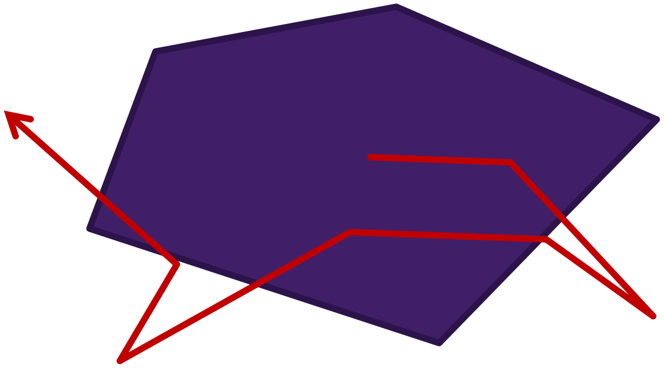
Harder to set learning rate

Higher variance

Constrained Optimization

- $\min_{w \in S} f(w)$
    - S convex set
- Apply gradient step
- Project the new vector to *S*

$$w_{m+1} = proj_S(\overline{w}_{m+1}) = \min_{x \in S} ||x - \overline{w}_{m+1}||^2$$



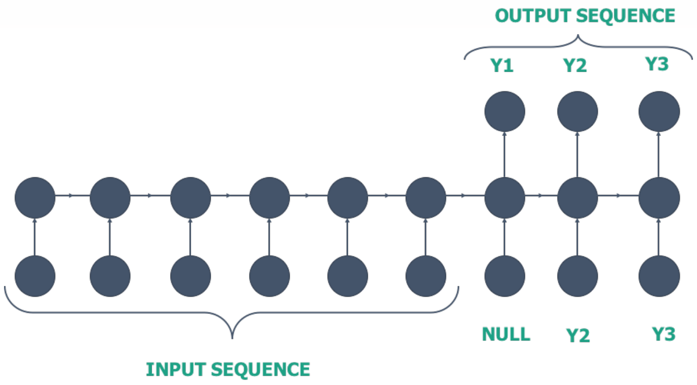
Frank-Wolfe Projection Algorithm

- $\min_{x} f(x)$ subject to $x \in S$
- First order approximation of $f(x)$
- Loop
  - Solve $\min_{s} s\nabla f(x_k)$ subject to $s \in S$
  - $\gamma' \in arg\min_{0 \leq \gamma \leq 1} f((1-\gamma)x_k + \gamma s_k)$
  - $x_{k+1} = (1-\gamma')x_k + \gamma' s_k$
- Second step can be replaced by
  - $\gamma = \frac{2}{s+k}$
  - Guarantees convergence

Lag

Lagrangian Relaxation

- $\min_x f(x)$ subject to $f_i(x) \leq 0 \; i = 1,2,\ldots,k \; h_j(x) = 0 \; j = 1,2,\ldots,l$
- Examples
  - Lasso regression
    - min MSE subject to L1 loss $\leq \delta$
  - In regression impose signs on coefficients
  - Language generation
    - Do not repeat words
    - Article cannot be followed by an article
  - Inverse classification / Counterfactuals
    - Total amount of perturbation
    - Number of perturbed



Lagrangian Relaxation

- Impose penalties for violation

$$L(x, \lambda, \nu) = f(x) + \sum_i \lambda_i f_i(x) + \sum_j \nu_j h_j(x)$$

$$\lambda \geq 0$$

- Dual formulation

$$g(\lambda, \nu) = \min_x f(x) + \sum_i \lambda_i f_i(x) + \sum_j \nu_j h_j(x)$$

$$\max_{\lambda \geq 0, \nu} g(\lambda, \nu)$$

  - Max min problem

Lagrangian Duality

- Original problem
  - $\min_{x} \max_{\lambda,\nu} f(x) + \sum_i \lambda_i f_i(x) + \sum_j \nu_j h_j(x)$
  - Switched max and min
  - Do not get the same value
- Weak duality
  - For any $\lambda \geq 0$ we have $g(\lambda, \nu) \leq f(x^*)$
  - x^* optimal solution to constrained problem
- For reasonable convex problems
  - $\max_{\lambda \geq 0, \nu} g(\lambda, \nu) = f(x^*)$

Lag

Lagrangian Duality

- Solve the dual problem to find λ, ν, x'
- Note that x' might not satisfy all of the constraints
  - Use ad-hoc techniques to find a feasible x close to x'
  - Use post-processing projection algorithm
  - Lasso do not do anything

Lagrangian Algorithm

- Solve the dual problem by gradient
- Loop
  - By gradient optimization solve

    $$\min_{x} L(x, \lambda, \nu)$$

    - Use gradient with respect to x
    - Let x' be optimal
  - By gradient solve

    $$\max_{\lambda \geq 0, \nu} L(x', \lambda, \nu)$$

    - Projection of λ
    - Gradient with respect to λ, ν
    - $\nabla_{\lambda,\nu} L(x', \lambda, \nu) = (f_1(x'), \dots, f_k(x'), h_1(x'), \dots, h_l(x'))$

Momentum

- SGD struggles with ravines
  - In one dimension descent is much steeper than in another
  - SGD oscillates
  - Also when all regions have gentle slopes
    - Gradient is small
- Idea
  - Continuously keep going in the same direction
  - Gain confidence and start making bigger steps
  - Ball gaining momentum going down a slope

Momentum

- Stabilize the gradient by combining with previous gradients

$$v_m = \gamma v_{m-1} + t_m \nabla f(w_m)$$
$$w_{m+1} = w_m - v_m$$

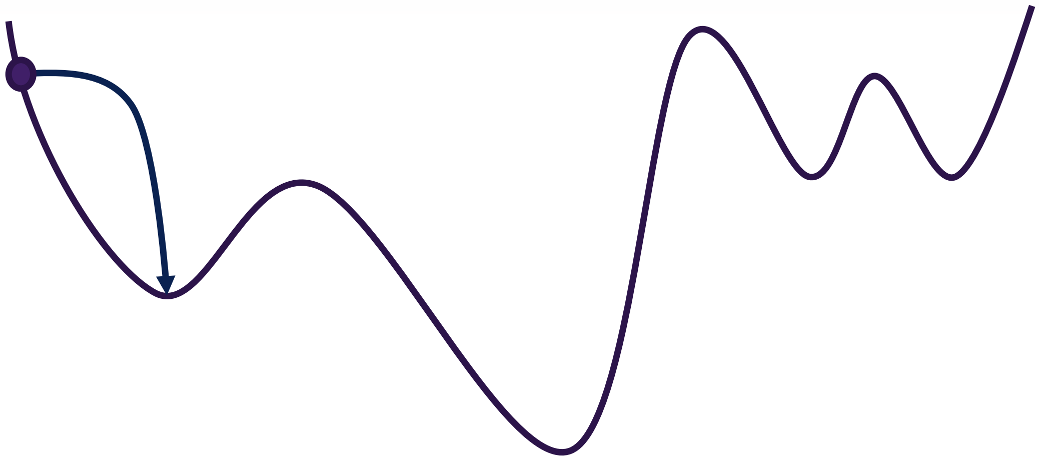
- Unwinding we obtain
  - $v_t = \sum_{i=0}^{t} \gamma^i \eta \nabla f(w_{t-i})$
  - Assumption $t_m = \eta$

Stochastic Gradient Descent

General Functions

General Functions

- f not convex
  - Deep learning
- Apply SGD in the same way
- Can find saddle point
  - Gradient zero but Hessian singular



Second Order

- $w_{m+1} = w_m - \alpha_m H(w_m)^{-1} \nabla f(w_m)$
- Find $H(w_m)u = -\nabla f(w_m)$
- Formal derivation
  - $f(w_m + s_m) \approx f(w_m) + \nabla f(w_m) s_m + \frac{1}{2} s_m H_m s_m$
  - Set derivative of right-hand side to zero
  - Same as minimize right-hand size with respect to s_m

Hessian-Free

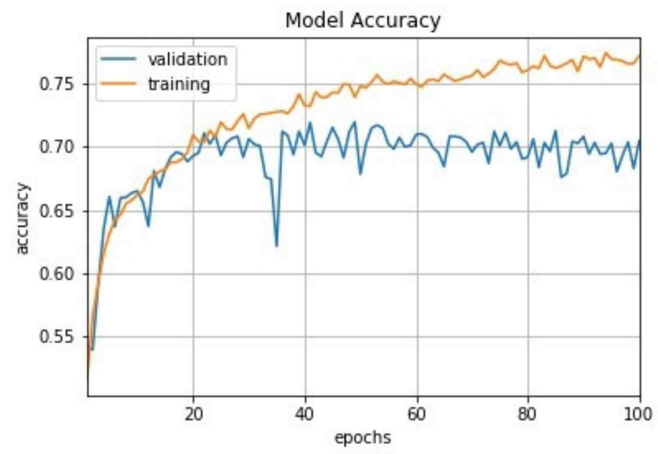
- Avoid singularity of Hessian
  - $H(w_m) + \lambda I$
- Observation
  - $H(w_m)u \approx \frac{\nabla f(w_m+\epsilon u)-\nabla f(w_m)}{\epsilon}$
  - Often possible to solve exactly
- Idea
  - $H(w_m)u = -\nabla f(w_m)$ the same as $\min_u ½\, uH(w_m)u + \nabla f(w_m)u$
  - Iteratively solve by conjugate-gradient
  - Requires only $H(w_m)v$ computations in an iterative setting

The Art of Learning Rate

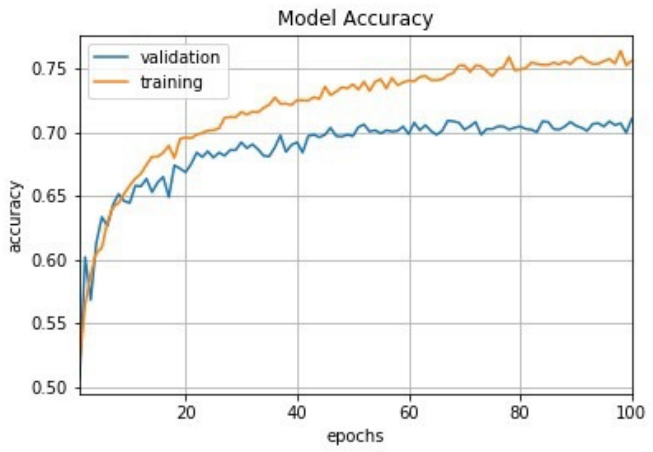
Learning Rate

Learning SGD

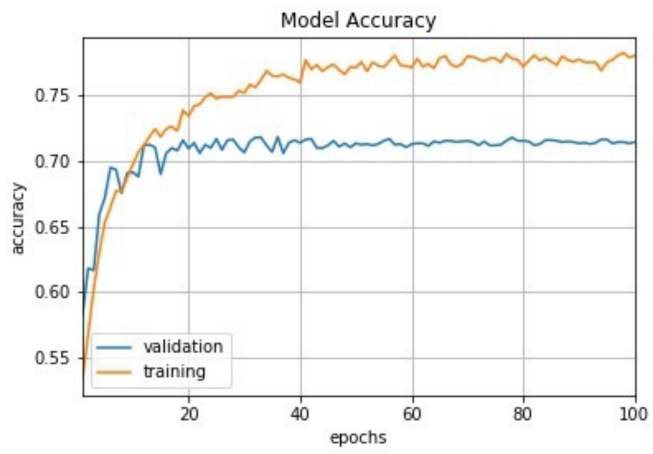
Constant



Time-based decay



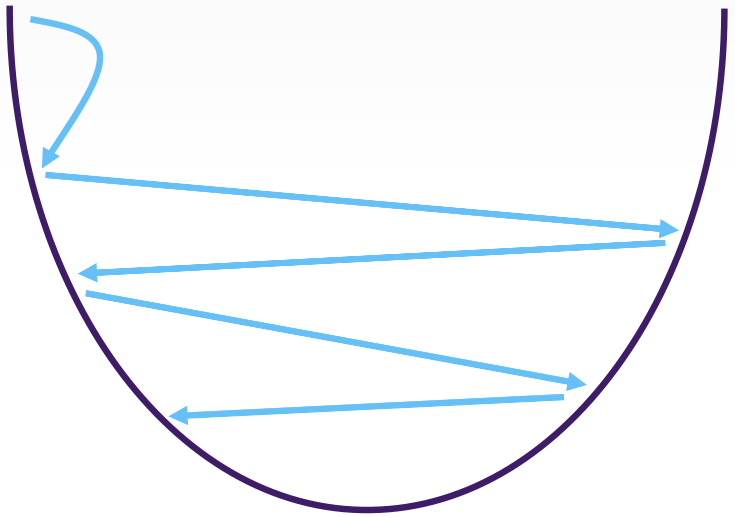
Step decay



Learning

SGD

- Too small learning rate
  - Convergence is very slow
- Too big
  - Jumping around too much



Adaptive: Adagrad

- Same learning rate for all weights
- Sparse feature (rare words in NLP) can be very important
  - Should get high learning rate
  - Likely to get small derivatives
- Weight j
  - Derivatives small (rare feature)
    - Larger learning rate to give them more importance
  - Derivatives large, small learning rate
- No longer moving in the gradient direction
- Learning rate always goes to zero
  - Not necessarily desirable

$$\lambda_t^j = \frac{\alpha}{\sqrt{\sum_{i=1}^{t-1} \nabla l(w_i)_j^2}}$$

Auxiliary Formula

$$u_t = \vartheta u_{t-1} + (1 - \vartheta) z_t$$

$$<=>$$

$$u_t = (1 - \vartheta)(z_t + \vartheta z_{t-1} + \vartheta^2 z_{t-2} + \cdots + \vartheta^t z_0)$$

- Exponential decay
- Can be easily tracked recursively

Adaptive: RMSprop

- Similar but weigh more recent iterates more
- Two parameters
  - Alpha as numerator
  - Theta for decay
- Updates can be made recursively

$$z_t^j = \nabla l(w_t)_j^2$$

$$lr_t^j = \frac{\alpha}{\sqrt{u_t^j}}$$

Adaptive: Adam

- Keep weighted decay moving average of
  - Derivate value
  - Square of derivate value
  - Both corrected by bias
- Correct for bias

w = w – RMSProp weighted_gradient

  - Weigh gradient by sum of squares of derivatives
- Parameters
  - Two weighted decay parameters
  - Numerator alpha

Adaptive: AdaDelta

- Learning rate converts units in variables to derivatives
  - Embed this conversion directly
- Numerator in learning rate
  - Weighted decay of previous weight changes
- Parameters
  - Two weighted decay parameters
- Alpha is no longer parameter

Adaptive Methods

- Not much of a difference among the adaptive methods
- Adam works best often
  - RMSprop extension of Adagrad
  - Adam and RMSprop similar performance initially
    - Towards end of optimization Adam outperforms
- SGD sometimes finds better solutions
  - Very tedious to find the right parameters