NLP 202
Deep Learning Optimizers

Jeffrey Flanigan

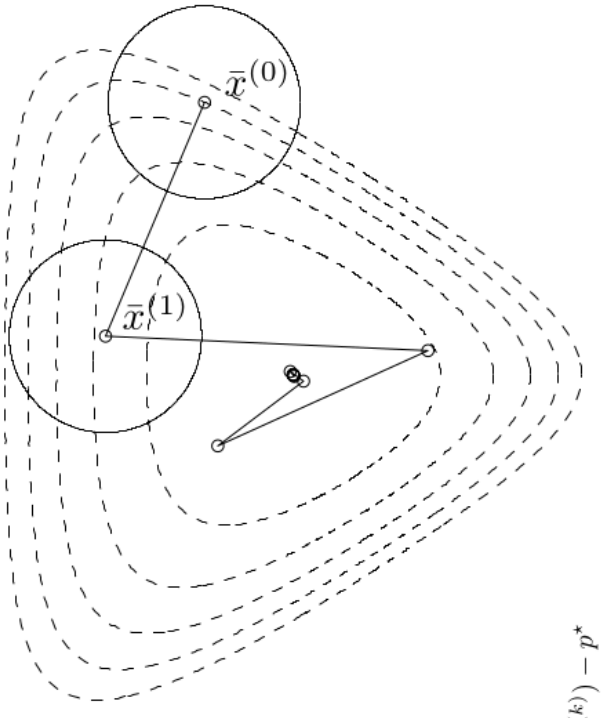
Winter 2023

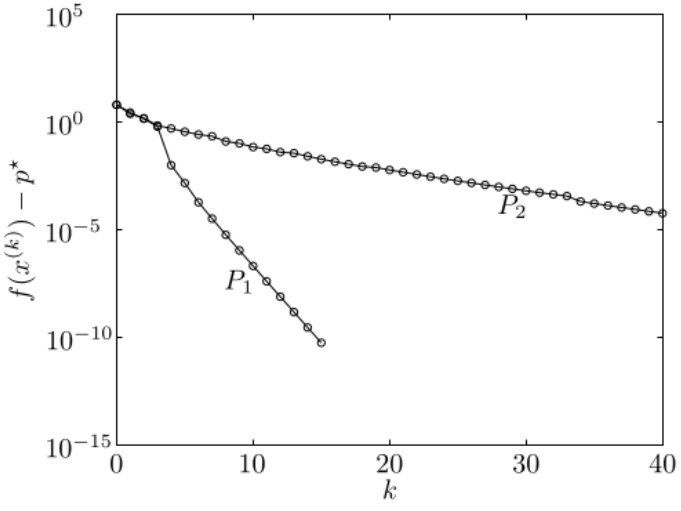
University of California Santa Cruz
jmflanig@ucsc.edu

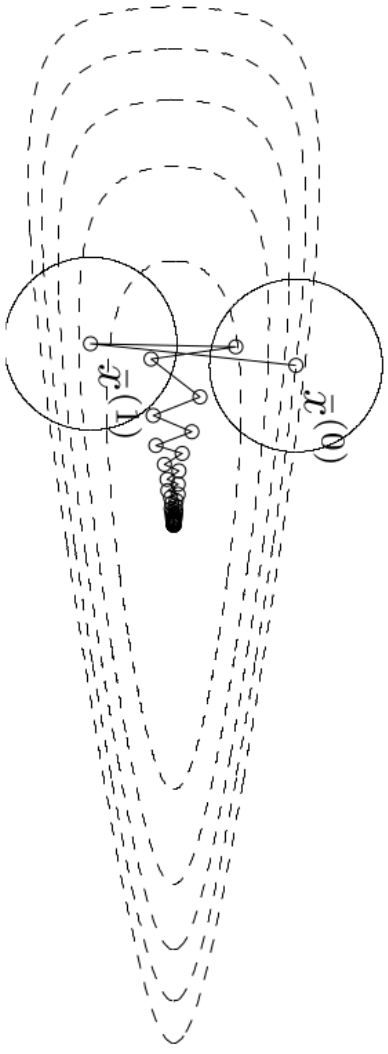
Plan

- Modern deep learning optimizers
- Distributed training

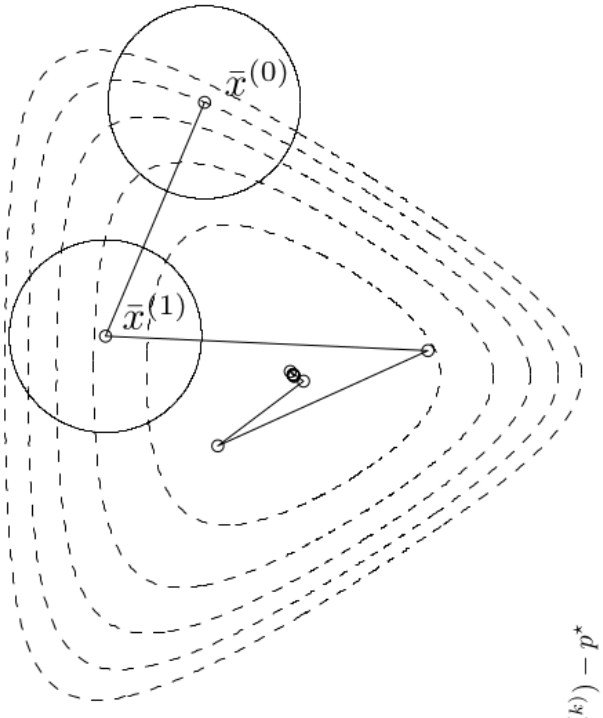
Conditioning

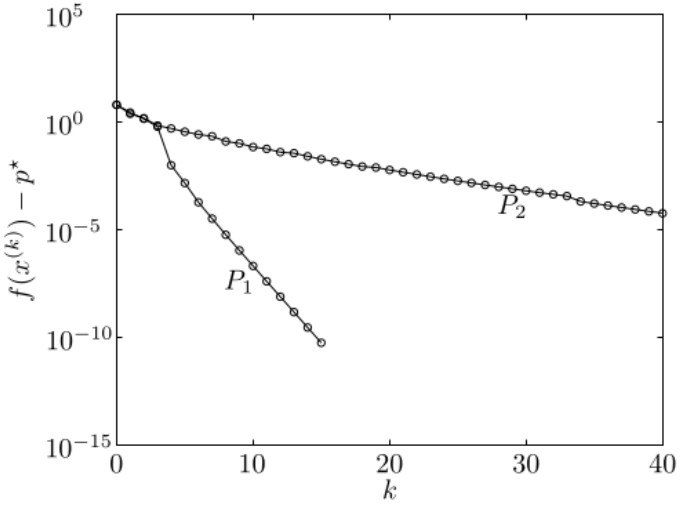


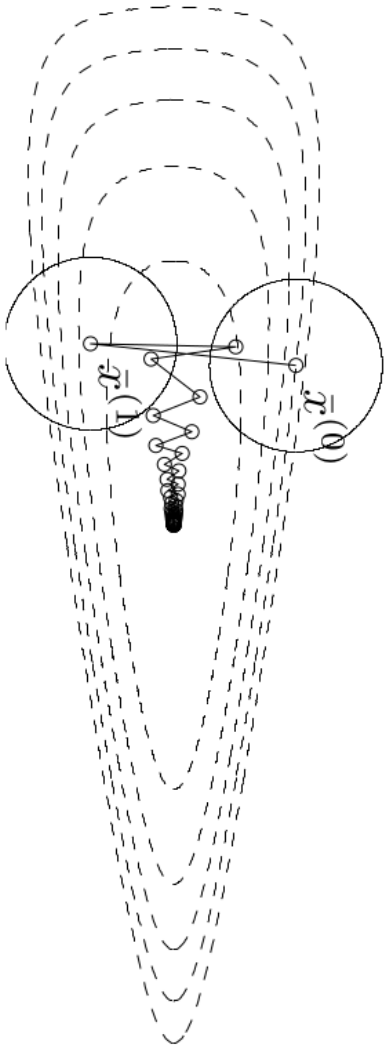




Zig-zagging of gradient descent due to poor conditioning

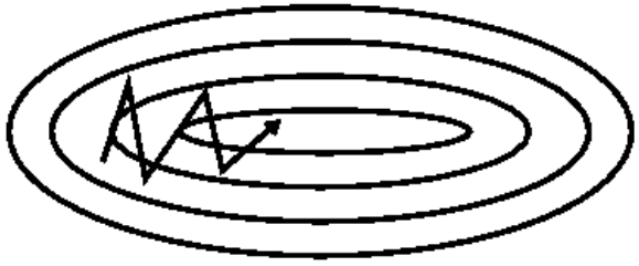






Momentum

(a) SGD without momentum



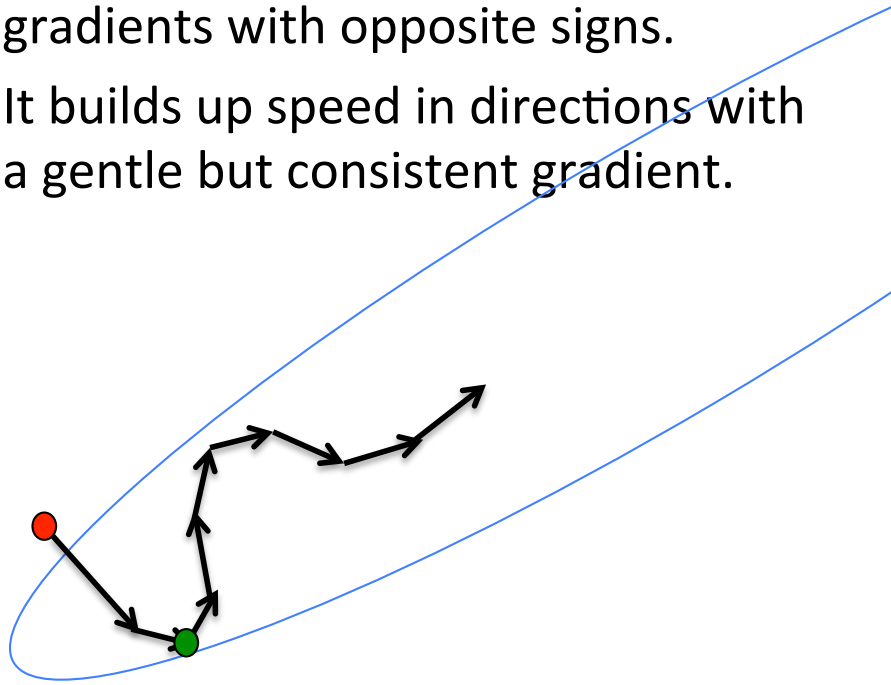
(b) SGD with momentum

The intuition behind the momentum method

Imagine a ball on the error surface. The location of the ball in the horizontal plane represents the weight vector.

- The ball starts off by following the gradient, but once it has velocity, it no longer does steepest descent.
- Its momentum makes it keep going in the previous direction.

- It damps oscillations in directions of high curvature by combining gradients with opposite signs.
- It builds up speed in directions with a gentle but consistent gradient.



Momentum (Polyak, 1964)

$$\mathbf{m} \leftarrow \beta \mathbf{m} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$$

Derivation: $\mathbf{F}$ is force, $\mathbf{m} = m\mathbf{v}$ is momentum.

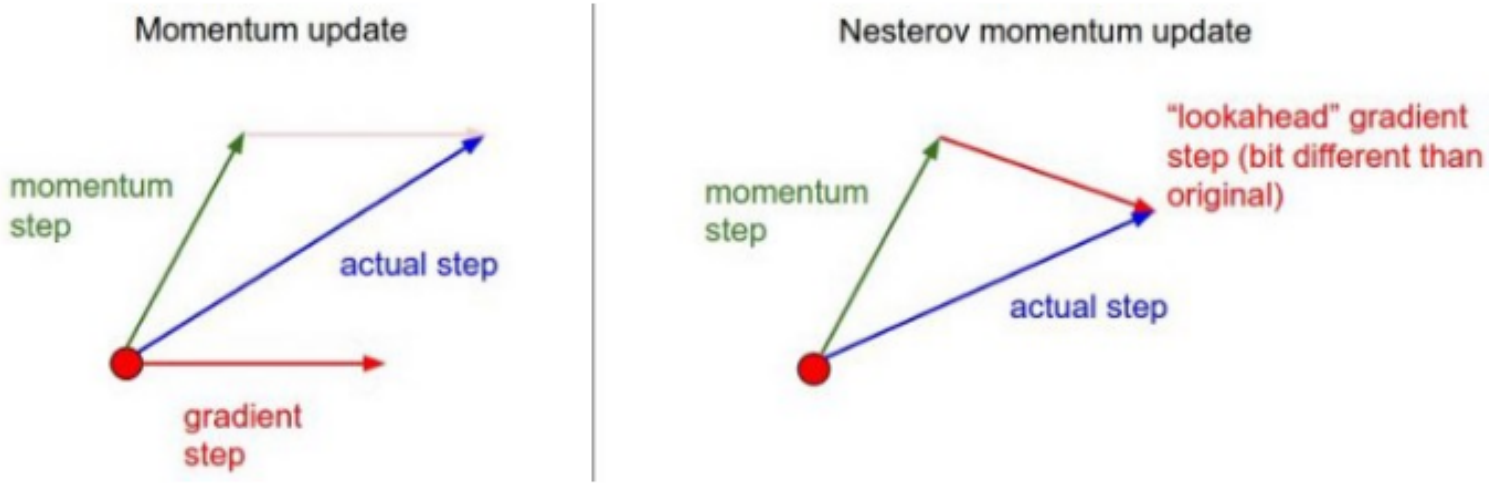
$$\mathbf{F} = m\mathbf{a} \Rightarrow \mathbf{F} = \frac{d\mathbf{m}}{dt} \Rightarrow d\mathbf{m} = \mathbf{F}dt \Rightarrow \mathbf{m}_{t+1} - \mathbf{m}_t = \mathbf{F}\alpha$$

$\mathbf{m}_{t+1} = \mathbf{m}_t + \mathbf{F}\alpha$ α is the finite timestep (dt)

$$\mathbf{F} = -\frac{(1-\beta)}{\alpha}\mathbf{m} - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Viscous drag $\propto \mathbf{m}$ + potential energy $\propto J(\boldsymbol{\theta})$

Nesterov Accelerated Gradient (Nesterov, 1983)



Source: http://cs231n.github.io/neural-networks-3/#sgd

Evaluate the gradient at the position the momentum will take us (look ahead)

$$\mathbf{m} \leftarrow \beta \mathbf{m} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \beta \mathbf{m})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$$

Nesterov Accelerated Gradient (Nesterov, 1983)

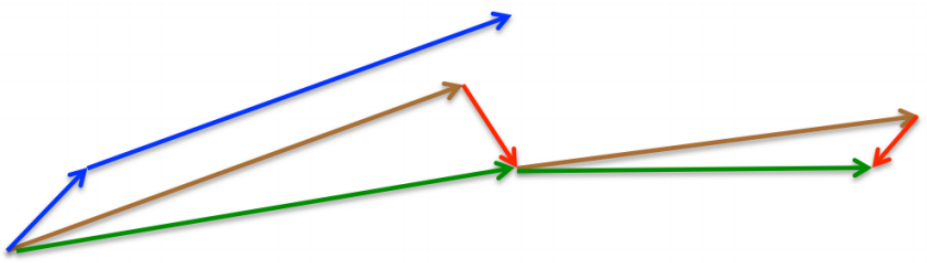


Figure 3: Nesterov update (Source: G. Hinton's lecture 6c)

These differences add up over time, gives a $O(\frac{1}{k^2})$ learning rate for smooth convex problems

AdaGrad (Duchi et al, 2011)

Take smaller steps in steeper directions

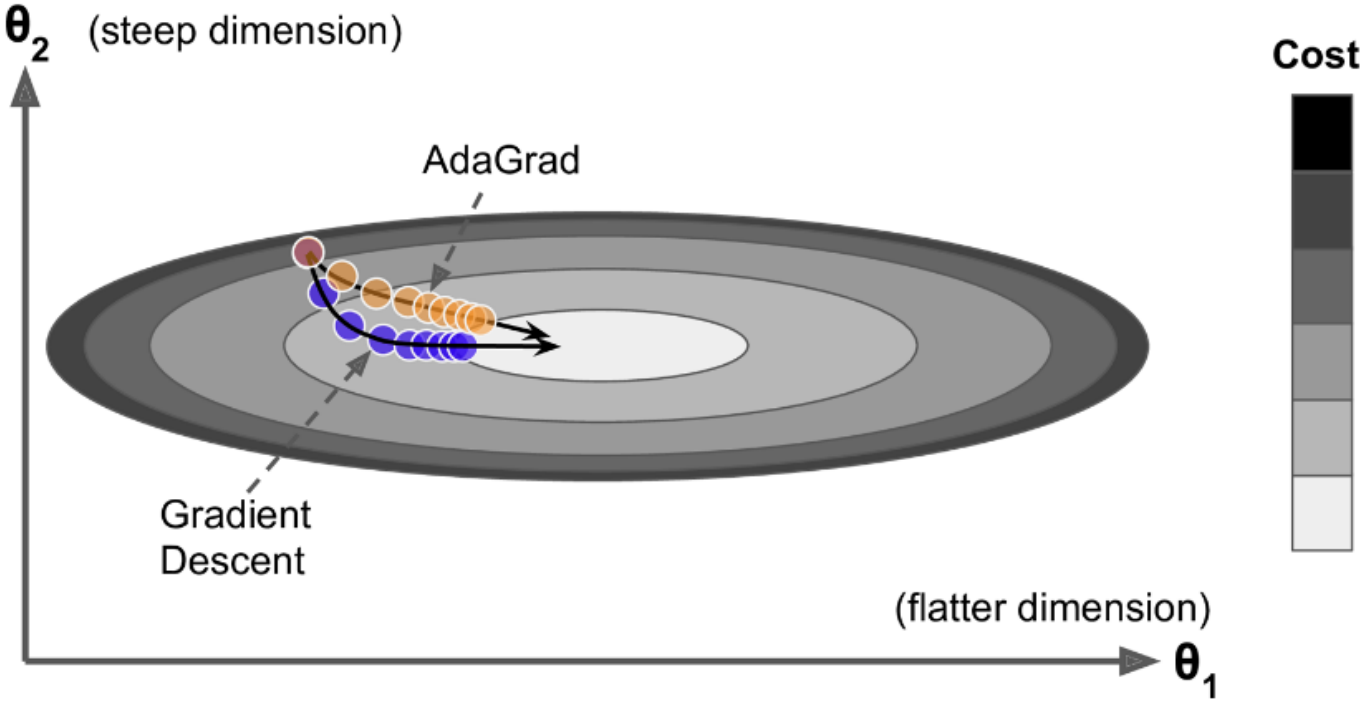


Figure 11-7. AdaGrad versus Gradient Descent: the former can correct its direction earlier to point to the optimum

AdaGrad (Duchi et al, 2011)

Take smaller steps in steeper directions.

Accumulate squared gradients in $\mathbf{s}$

scale-down step in each dimension by $\sqrt{\mathbf{s}}$

$$\mathbf{s} \leftarrow \mathbf{s} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{\sqrt{\mathbf{s}}} \odot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$\odot$ denotes the **Hadamard product** (element-wise multiplication)

RMSProp (Tieleman & Hinton, 2012)

Similar to Adagrad, but exponentially decay the past gradients so we pay more attention to the recent ones.

$$\mathbf{s} \leftarrow \beta \mathbf{s} + (1-\beta) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{\sqrt{\mathbf{s}}} \odot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$\beta \in (0, 1)$ is a hyperparameter.
Gradients k steps ago will be scaled by β^k.

Adam (Kingma & Ba, 2014)

Combine RMSProp and momentum

Equation 11-8. Adam algorithm

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
3. $\widehat{\mathbf{m}} \leftarrow \dfrac{\mathbf{m}}{1 - {\beta_1}^t}$
4. $\hat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1 - {\beta_2}^t}$
5. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \widehat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}} + \varepsilon}$

In this equation, *t* represents the iteration number (starting at 1).

$\hat{\mathbf{m}}$ and $\hat{\mathbf{s}}$ are mean and variance of the gradients

Correction factor to make them unbiased estimates

Nadam (Dozat, 2016)

- Combines Adam with the Nesterov trick
- Often converges slightly faster than Adam

Adaptive Gradient Methods Can Be Harmful

- Adaptive gradient methods like Adagrad and Adam can be harmful for some problems (Wilson 2017 - The Marginal Value of Adaptive Gradient Methods in Machine Learning)
- If performance lower than expected, try SGD with momentum or Nesterov accelerated gradient instead
- Can also restart Adam (Denkowski 2017 - Stronger Baselines for Trustable Results in Neural Machine Translation) so it forgets the gradient history

Apollo (Ma, 2021)

Quasi-Newton optimizer for deep learning. Builds an estimate of the hessian from the gradients. Makes a diagonal approximation to the hessian.

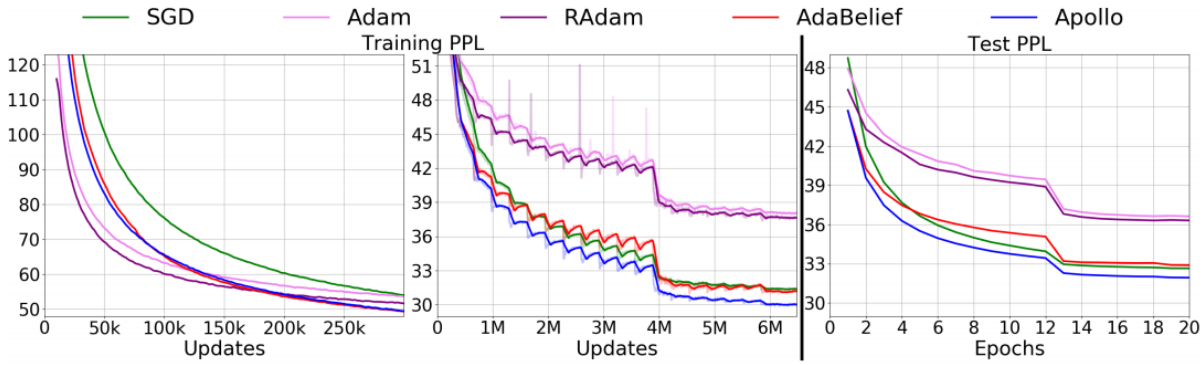


Figure 3: Language modeling (LSTMs) on One Billion Words.

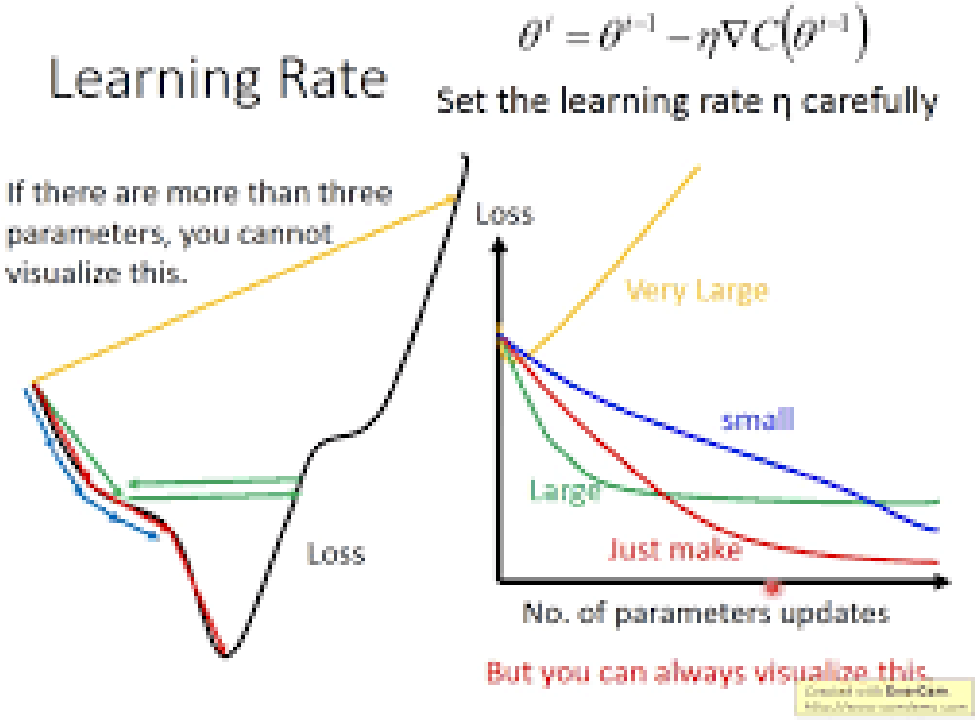
Method	PPL
SGD	32.65
Adam	36.68
RAdam	36.20
AdaBelief	32.83
APOLLO	**31.94**

Table 2: Test PPL.

Convergence Rate

- Usually $O(\frac{1}{\sqrt{T}})$ for convex and non-convex case

Learning rate



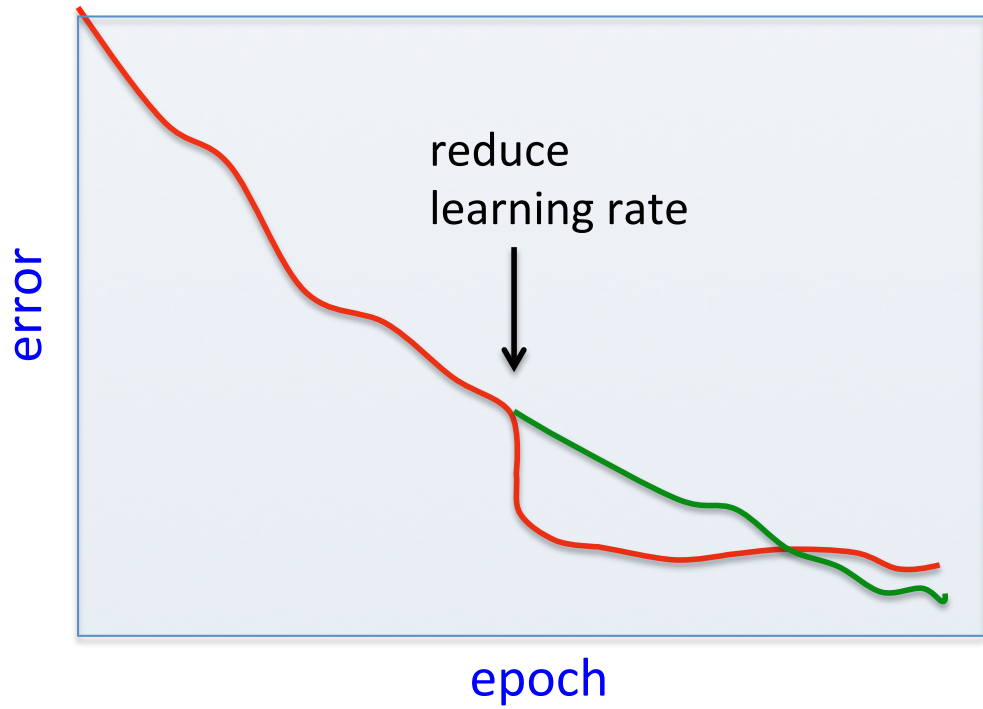
Learning rate schedules

- Decaying step size
- Step reduction
- Warm-up (see also RAdam optimizer)
- Reduce the learning rate (divide by 2 or 10) when performance increase on dev set stops
- Cyclic and one-cycle learning rates

Check NLP wiki - Learning Rate for more.

Be careful about turning down the learning rate

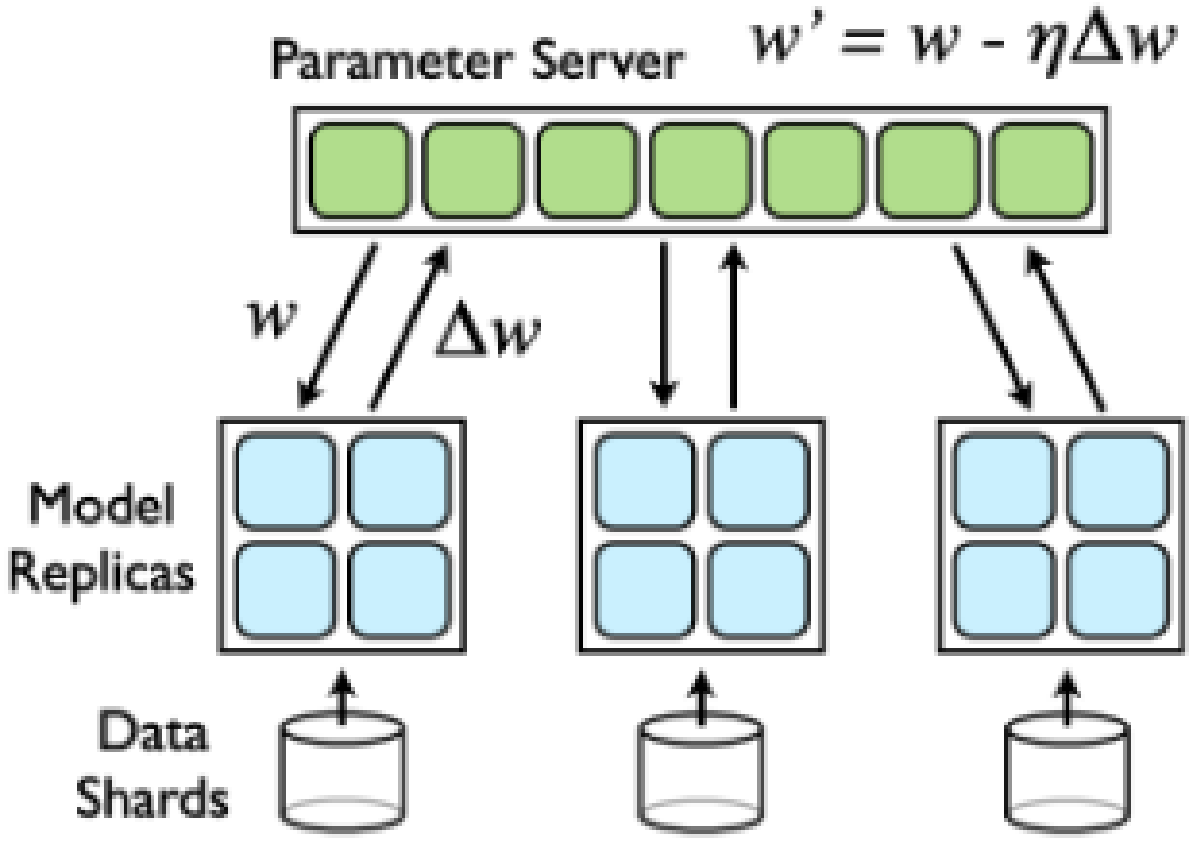
- Turning down the learning rate reduces the random fluctuations in the error due to the different gradients on different mini-batches.
  - So we get a quick win.
  - But then we get slower learning.
- Don't turn down the learning rate too soon!



Stopping Criterion

- Ideally do early-stopping on a development set using the task metric (F1, BLEU, etc)
- With Transformer models, often takes too long to converge
- Common to stop after a fixed number of minibatch steps

Distributed Training: Parameter Server



One or more parameter servers aggegate the model updates, send to the workers.

Major limitation: latency and bandwidth for model and gradients.

Distributed Training: Parameter Server

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

issue LoadData() to all workers
for iteration $t = 0, \ldots, T$ **do**
    issue WORKERITERATE(t) to all workers.
end for

Worker $r = 1, \ldots, m$:

function LOADDATA()
    load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
    pull the working set $w_r^{(0)}$ from servers
end function
function WORKERITERATE(t)
    gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
    push $g_r^{(t)}$ to servers
    pull $w_r^{(t+1)}$ from servers
end function

Servers:

function SERVERITERATE(t)
    aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
    $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)})\right)$
end function

Figure from Li et al 2014 - Scaling Distributed Machine Learning with the Parameter Server.

Distributed Training: Hogwild and sparse updates

- Niu et al 2011 - Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent
  Do the updates asynchronously, no locking. Works well for sparse updates.
- Aji and Heafield 2017 - Sparse Communication for Distributed Gradient Descent
  Zero out small components of the gradient (reduce by 99%)