DeepMind

Optimization for Machine Learning

James Martens

UCL x DeepMind Lectures

Plan for this Lecture

1. Intro and motivation
2. Gradient descent
3. Momentum methods
4. 2nd-order methods
5. Stochastic optimization

1 Intro and motivation

Motivation

- Optimization algorithms are the basic engine behind deep learning methods that enable models to learn from data by adapting their **parameters**

- They solve the problem of the minimization of an **objective function** that measures the mistakes made by the model
    - e.g. prediction error (classification), negative reward (reinforcement learning)

- Work by making a sequence of small incremental changes to model parameters that are each guaranteed to reduce the objective by some small amount

Basic notation

- Parameters:

$$\theta \in \mathbb{R}^n$$ ← dimension

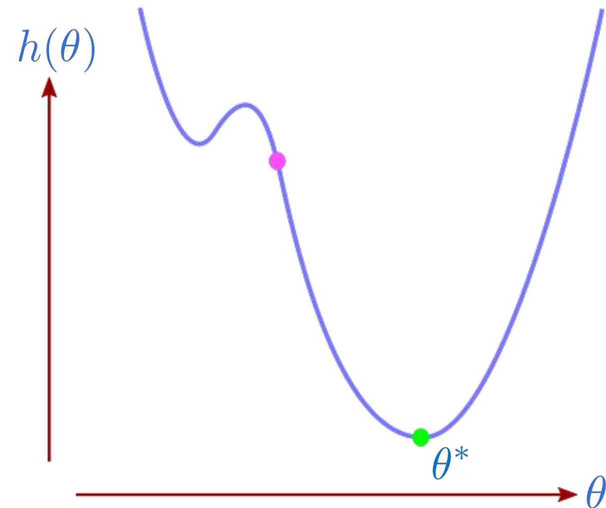
- Real-valued objective function :

$$h(\theta)$$

- Goal of optimization:

$$\theta^* = \arg\min_\theta h(\theta)$$

1D example objective function



Example: neural network training objective

- The standard neural network training objective is given by:

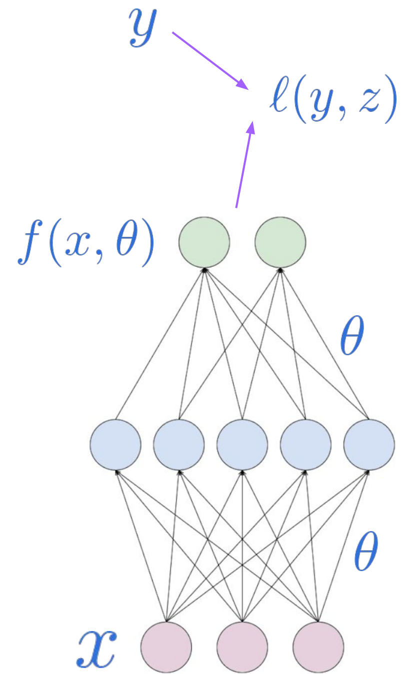
$$h(\theta) = \frac{1}{m}\sum_{i=1}^{m} \ell(y_i, f(x_i, \theta))$$

where:

$\ell(y, z)$ is a loss function measuring disagreement between y and z

and

$f(x, \theta)$ is a neural network function taking input x and outputting some prediction



2 Gradient descent

Gradient descent: definition

- Basic gradient descent iteration:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla h(\theta_k)$$

Learning rate: α_k

(aka "step size")

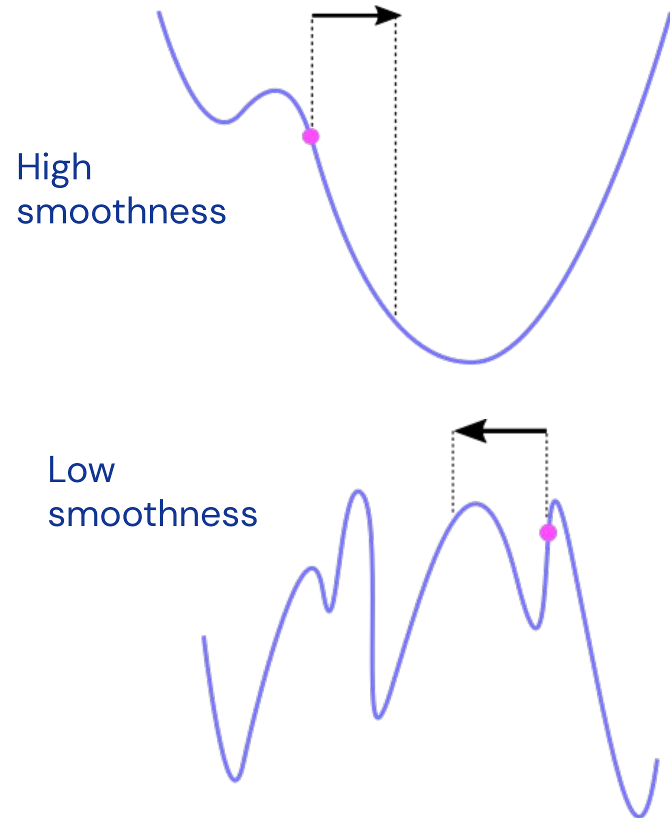
Gradient: $\nabla h(\theta) = \begin{bmatrix} \frac{\partial h(\theta)}{\partial [\theta]_1} \\ \frac{\partial h(\theta)}{\partial [\theta]_2} \\ \vdots \\ \frac{\partial h(\theta)}{\partial [\theta]_n} \end{bmatrix}$

Intuition: gradient descent is "steepest descent"

$$\theta_{k+1} = \theta_k - \alpha_k \nabla h(\theta_k)$$

- Gradient direction $\nabla h(\theta)$ gives *greatest* reduction in $h(\theta)$ per unit of change* in θ.

- If $h(\theta)$ is "sufficiently smooth", and learning rate small, gradient will keep pointing down-hill over the region in which we take our step



Intuition: gradient descent is minimizing a local approximation

- 1st-order Taylor series for $h(\theta)$ around current θ is:

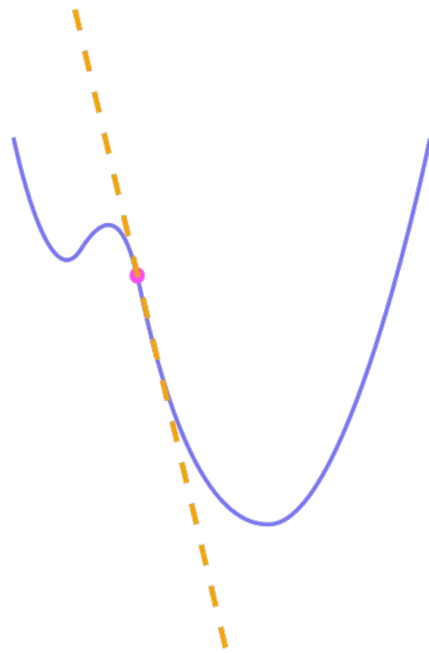
$$h(\theta + d) \approx h(\theta) + \nabla h(\theta)^\top d$$

- For small enough d this will be a reasonable approximation
- Gradient update computed by minimizing this within a sphere of radius r:

$$-\alpha \nabla h(\theta) = \arg\min_{d : \|d\| \leq r} \left(h(\theta) + \nabla h(\theta)^\top d \right)$$

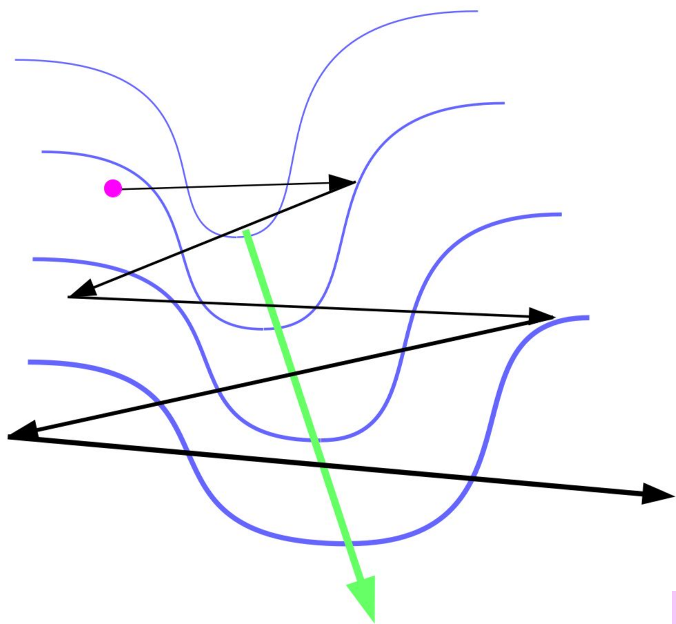
where

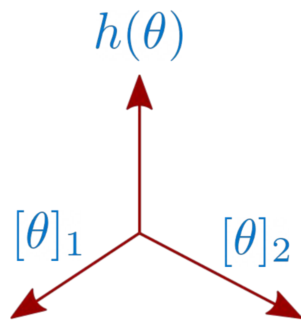
$$r = \alpha \|\nabla h(\theta)\|$$



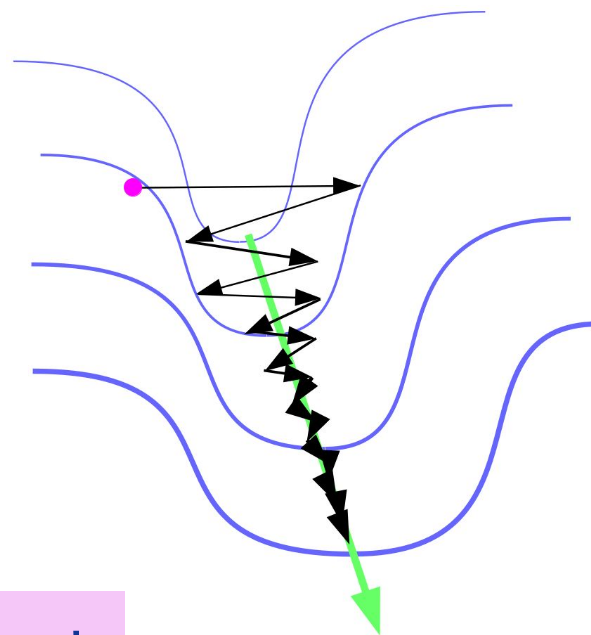
The problem with gradient descent visualized: the 2D "narrow valley" example

Large learning rate (α)





Small learning rate



No good choice !

Convergence theory: technical assumptions

- $h(\theta)$ has Lipschitz continuous derivatives (i.e. is "Lipschitz smooth"):

$$\|\nabla h(\theta) - \nabla h(\theta')\| \leq L\|\theta - \theta'\|$$

(an **upper bound** on the curvature)

- $h(\theta)$ is strongly convex (perhaps only near minimum):

$$h(\theta + d) \geq h(\theta) + \nabla h(\theta)^\top d + \frac{\mu}{2}\|d\|^2$$

(a **lower bound** on the curvature)

- And *for now*: Gradients are computed exactly (i.e. **not** stochastic)

Convergence theory: upper bounds

If previous conditions hold and we take $\alpha_k = \dfrac{2}{L+\mu}$:

$$h(\theta_k) - h(\theta^*) \leq \frac{L}{2}\left(\frac{\kappa - 1}{\kappa + 1}\right)^{2k} \|\theta_0 - \theta^*\|^2$$

minimizer

where $\kappa = L/\mu$.

Number of iterations to achieve $h(\theta_k) - h(\theta^*) \leq \epsilon$ is

$$k \in \mathcal{O}\left(\kappa \log \frac{1}{\epsilon}\right)$$

Convergence theory: useful in practice?

- Issues with bounds such as this one:
  - too pessimistic (they must cover worst-case examples)
  - some assumptions too strong (e.g. convexity)
  - other assumptions too weak (real problems have additional useful structure)
  - rely on crude measures of objective (e.g. condition numbers)
  - usually focused on asymptotic behavior
- The design/choice of an optimizer should always be informed by **practice** more than anything else. But theory can help guide the way and build intuitions.

3 Momentum methods

The momentum method

- Motivation:
  - the gradient has a tendency to flip back and forth as we take steps when the learning rate is large
  - e.g. the narrow valley example
- The key idea:
  - accelerate movement along directions that point consistently down-hill across many consecutive iterations (i.e. have low curvature)
- How?
  - treat current solution for θ like a "ball" rolling along a "surface" whose height is given by $h(\theta)$, subject the force of gravity

Credit: Devinsupertramp via youtube.com

Defining equations for momentum

- Classical Momentum:

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

$$\theta_{k+1} = \theta_k + \alpha_k v_{k+1}$$

Learning rate: α_k

Momentum constant: η_k

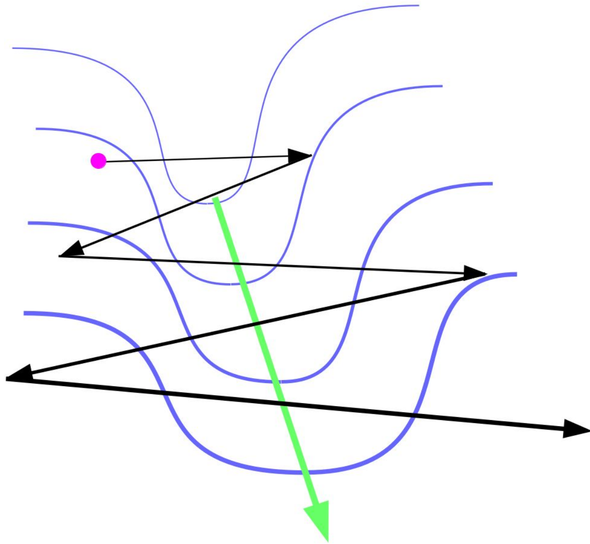
- Nesterov's variant:

$$v_{k+1} = \eta_k v_k - \nabla h(\theta_k + \alpha_k \eta_k v_k) \qquad v_0 = 0$$

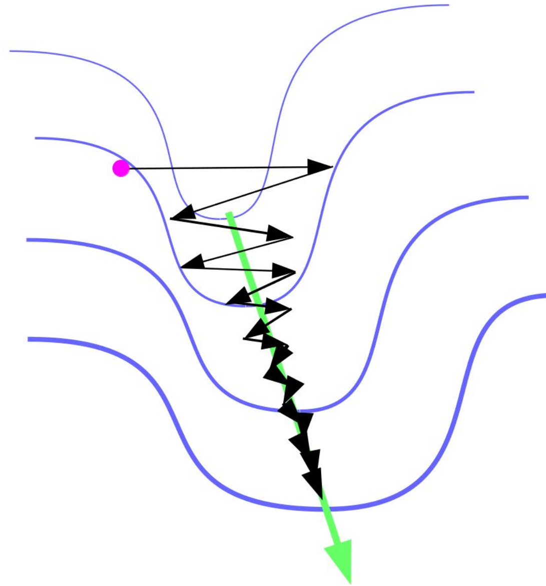
$$\theta_{k+1} = \theta_k + \alpha_k v_{k+1}$$

Narrow 2D valley example revisited

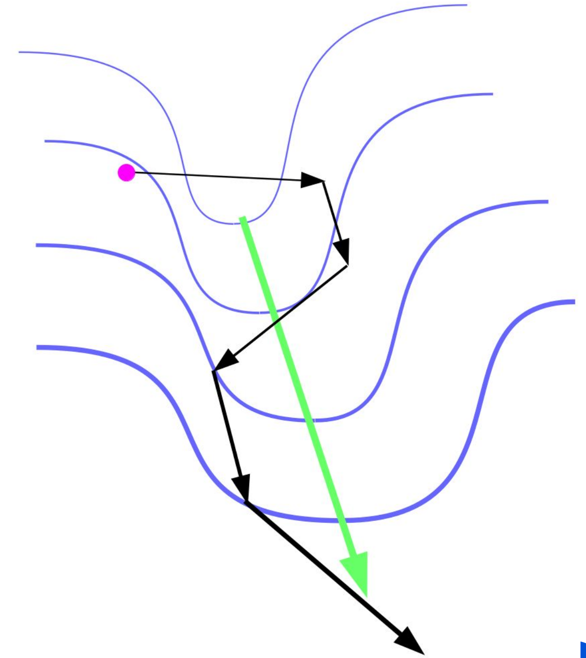
Gradient descent with large learning rate



Gradient descent with small learning rate



Momentum method



Upper bounds for Nesterov's momentum variant

Given objective $h(\theta)$ satisfying same technical conditions as before, and careful choice of α_k and η_k, Nesterov's momentum method satisfies:

$$h(\theta_k) - h(\theta^*) \leq L \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa}} \right)^k \|\theta_0 - \theta^*\|^2 \qquad \kappa = \frac{L}{\mu}$$

Number of iterations to achieve $h(\theta_k) - h(\theta^*) \leq \epsilon$:

$$k \in \mathcal{O}\left(\sqrt{\kappa} \log \frac{1}{\epsilon} \right)$$

Convergence theory: 1st-order methods and *lower bounds*

- A **first-order method** is one where updates are linear combinations of observed gradients. i.e.:

$$\theta_{k+1} - \theta_k = d \in \mathrm{Span}\{\nabla h(\theta_0), \nabla h(\theta_1), \ldots, \nabla h(\theta_k)\}$$

- Included:
  - gradient descent
  - momentum methods
  - conjugate gradients (CG)
- Not included:
  - preconditioned gradient descent / 2nd-order methods

Lower bounds (cont.)

Assume number of steps is greater than the dimension n (it usually is).
Then, there is example objective satisfying previous conditions for which:

$$h(\theta_k) - h(\theta^*) \geq \frac{\mu}{2} \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^{2k} \|\theta_0 - \theta^*\|^2 \qquad \kappa = L/\mu$$

Number of iterations to achieve $h(\theta_k) - h(\theta^*) \leq \epsilon$:

$$k \in \Omega \left(\sqrt{\kappa} \log \frac{1}{\epsilon} \right)$$

Comparison of iteration counts

To achieve $h(\theta_k) - h(\theta^*) \leq \epsilon$ the number of iterations k satisfies:

- (Worst-case) lower bound for 1st-order methods: $k \in \Omega\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$
- Upper bound for gradient descent: $k \in \mathcal{O}\left(\kappa\log\frac{1}{\epsilon}\right)$
- Upper bound for GD w/ Nesterov's momentum: $k \in \mathcal{O}\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$

4 2nd-order methods

The problem with 1st-order methods

- For any 1st-order method, the number of steps needed to converge grows with "condition number":

$$\kappa = \frac{L}{\mu}$$

L ← Max curvature

μ ← Min curvature

- This will be very large for some problems (e.g. certain deep architectures)
- 2nd-order methods can improve (or even eliminate) this dependency

Derivation of Newton's method

- Approximate $h(\theta)$ by its 2nd-order Taylor series around current θ :

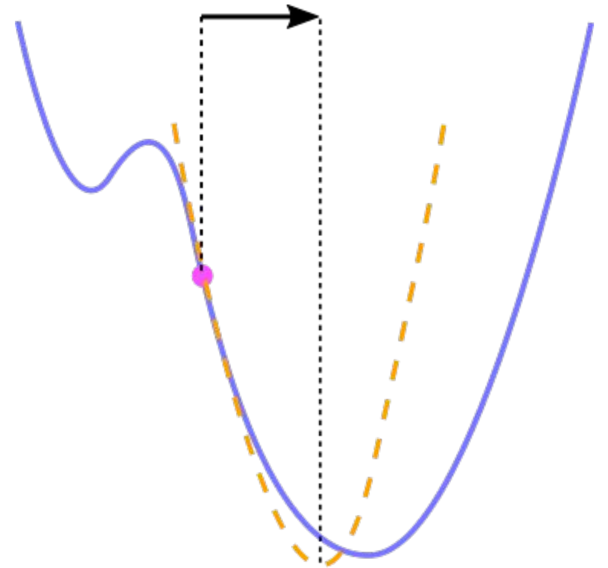
$$h(\theta + d) \approx h(\theta) + \nabla h(\theta)^\top d + \frac{1}{2} d^\top H(\theta) d$$

- Minimize this local approximation to obtain:

$$d = -H(\theta)^{-1} \nabla h(\theta)$$

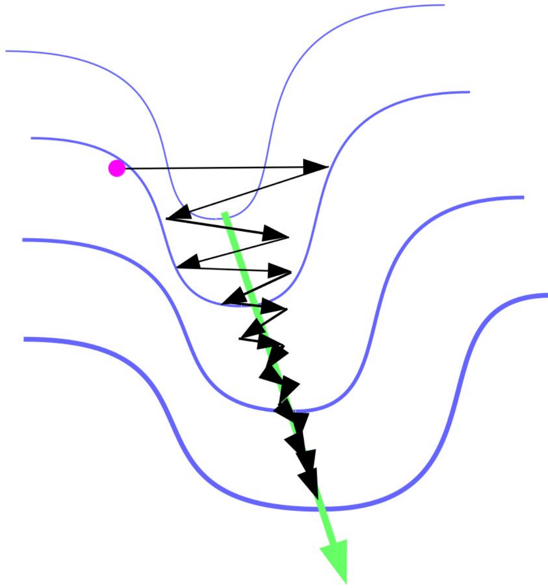
- Update current iterate with this:

$$\theta_{k+1} = \theta_k - H(\theta)^{-1} \nabla h(\theta_k)$$

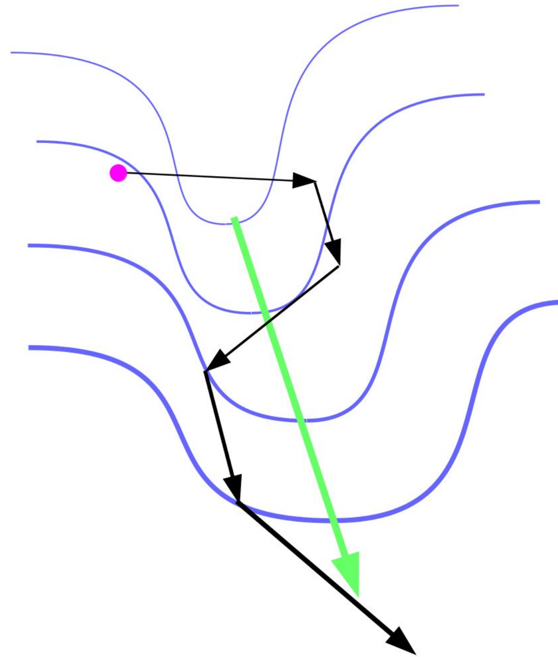


The 2D narrow valley example revisited (again)

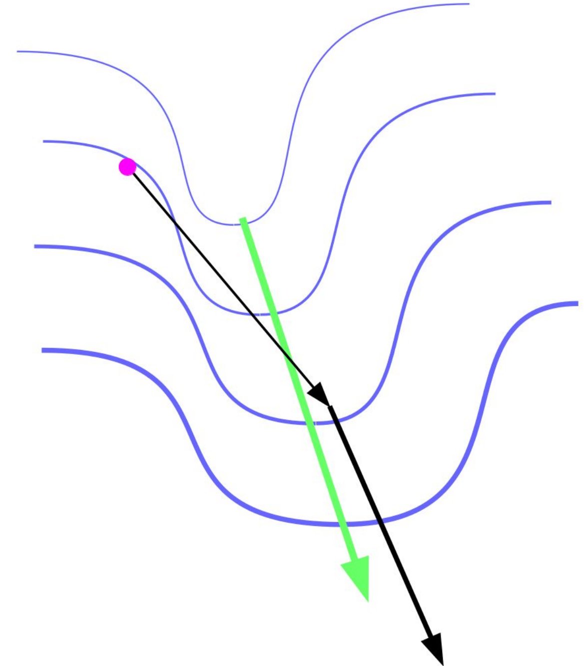
Gradient descent



Momentum method



2nd-order method



Comparison to gradient descent

- Maximum allowable global learning rate for GD to avoid divergence:

$$\alpha = 1/L$$

L is maximum curvature
aka "Lipschitz constant"

- Gradient descent implicitly minimizes a bad approximation of 2nd-order Taylor series:

$$h(\theta + d) \approx h(\theta) + \nabla h(\theta)^\top d + \frac{1}{2} d^\top H(\theta) d$$

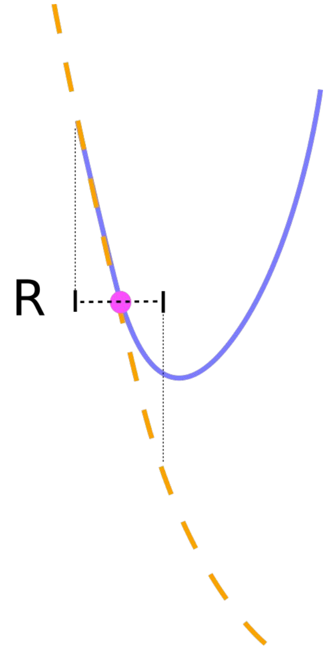
$$\approx h(\theta) + \nabla h(\theta)^\top d + \frac{1}{2} d^\top (LI) d$$

- LI is too pessimistic / conservative an approximation of $H(\theta)$! Treats all directions as having max curvature.

Breakdown of local quadratic approximation and how to deal with it

- Quadratic approximation of objective is only trustworthy in a local region around current θ
- Gradient descent (implicitly) approximates the curvature everywhere by its global max (and so doesn't have this problem)
- Newton's method uses $H(\theta)$, which may become an underestimate in the region we are taking our update step

Solution: Constrain update d to lie in a "trust region" R around, where approximation remains "good enough"



Trust-regions and "damping"

- If we take $R = \{d : \|d\|_2 \leq r\}$ then computing

$$\arg\min_{d \in R} \left(h(\theta) + \nabla h(\theta)^\top d + \frac{1}{2} d^\top H(\theta) d \right)$$

  is often equivalent to

$$-(H(\theta) + \lambda I)^{-1} \nabla h(\theta) = \arg\min_{d} \left(h(\theta) + \nabla h(\theta)^\top d + \frac{1}{2} d^\top (H(\theta) + \lambda I) d \right)$$

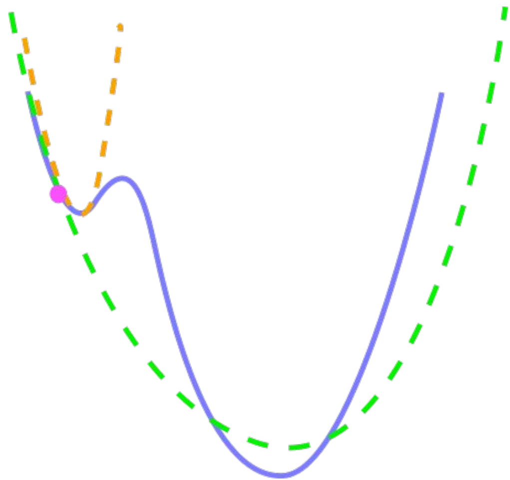
  for some λ.

- λ depends on r in a complicated way, but we can just work with λ directly

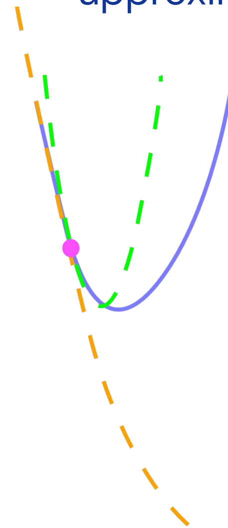
Alternative curvature matrices

$H(\theta)$ does not necessarily give the best quadratic approximation for optimization. Different replacements for $H(\theta)$ could produce:

A more global approximation



A more conservative approximation



Alternative curvature matrices (cont.)

- The most important family of related examples includes:
  - Generalized Gauss-Newton matrix (GGN)
  - Fisher information matrix
  - “Empirical Fisher”
- Nice properties:
  - always positive semi-definite (i.e. no negative curvature)
  - give parameterization invariant updates in small learning rate limit (unlike Newton’s method!)
  - work much better in practice for neural net optimization

Barrier to application of 2nd-order methods for neural networks

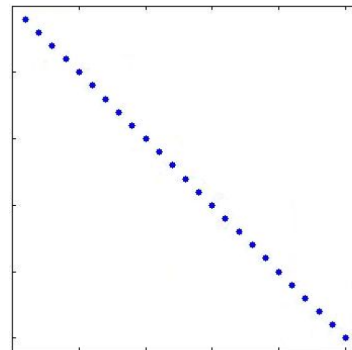
- For neural networks, $\theta \in \mathbb{R}^n$ can have 10s of millions of dimensions

- We simply cannot compute and store an $n \times n$ matrix, let alone invert it!

- To use 2nd-order methods, we must simplify the curvature matrix's
  - computation,
  - storage,
  - and inversion

This is typically done by approximating the matrix with a simpler form.

Diagonal approximations



The simplest approximation: include only the diagonal entries of curvature matrix (setting the rest to zero)

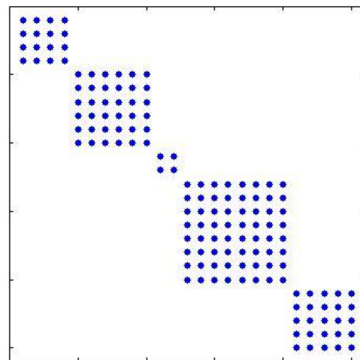
Properties:

- Inversion and storage cost: $\mathcal{O}(n)$
- Computational costs depends on form of original matrix (ranges from easy to hard)
- Unlikely to be accurate, but can compensate for basic scaling differences between parameters

Used (with a square root) in RMS-prop and Adam methods to approximate Empirical Fisher matrix

Block-diagonal approximations

Another option is to take only include certain diagonal blocks.



For neural nets, a block could correspond to:

- weights on connections going into a given unit
- weights on connections going out of a given unit
- all the weights for a given layer

Properties:

- Storage cost: $\mathcal{O}(bn)$ (assuming $b \times b$ block size)
- Inversion cost: $\mathcal{O}(b^2 n)$
- Similar difficulty to computing diagonal
- Can only be realistically applied for small block sizes

Well-known example developed for neural nets: TONGA

Kronecker-product approximations

- Block-diagonal approximation of GGN/Fisher where blocks correspond to network layers
- Approximate each block as Kronecker product of two small matrices:

$$A \otimes C = \begin{bmatrix} [A]_{1,1}C & \cdots & [A]_{1,k}C \\ \vdots & \ddots & \vdots \\ [A]_{k,1}C & \cdots & [A]_{k,k}C \end{bmatrix}$$

- Storage and computation cost: $\mathcal{O}(n)$*
- Cost to apply inverse: $\mathcal{O}(b^{0.5}n)$ (uses $(A \otimes C)^{-1} = A^{-1} \otimes C^{-1}$)
- Used in current most powerful neural net optimizer (K-FAC)

5 Stochastic methods

Motivation for stochastic methods

- Typical objectives in machine learning are an average over training cases of case-specific losses:

$$h(\theta) = \frac{1}{m}\sum_{i=1}^{m} h_i(\theta)$$

- m can be **very** big, and so computing the gradient gets expensive:

$$\nabla h(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla h_i(\theta)$$

Mini-batching

- Fortunately there is often significant statistical overlap between $h_i(\theta)$'s
- Early in learning, when "coarse" features of the data are still being learned, most $\nabla h_i(\theta)$' s will look similar
- **Idea:** randomly subsample a "mini-batch" of training cases $S \subset \{1, 2, ..., m\}$ of size $b \ll m$, and estimate gradient as:

$$\widetilde{\nabla} h(\theta) = \frac{1}{b} \sum_{i \in S} \nabla h_i(\theta)$$

Stochastic gradient descent (SGD)

- Stochastic gradient descent (SGD) replaces $\nabla h(\theta)$ with its mini-batch estimate $\widetilde{\nabla} h(\theta)$, giving:

$$\theta_{k+1} = \theta_k - \alpha_k \widetilde{\nabla} h(\theta_k)$$

- To ensure convergence, need to do one of the following:
  - Decay learning rate: $\alpha_k = 1/k$
  - Use "Polyak averaging": $\bar{\theta}_k = \frac{1}{k+1} \sum_{i=0}^{k} \theta_i$ or $\bar{\theta}_k = (1-\beta)\theta_k + \beta \bar{\theta}_{k-1}$
  - Slowly increase the mini-batch size during optimization

Convergence of stochastic methods

- Stochastic methods converge slower than corresponding non-stochastic versions
- Asymptotic rate for SGD with Polyak averaging:

Gradient estimate covariance matrix

$$E[h(\theta_k)] - h(\theta^*) \in \frac{1}{2k} \operatorname{tr}\left(H(\theta^*)^{-1}\Sigma\right) + \mathcal{O}\left(\frac{1}{k^2}\right)$$

- Iterations to converge:

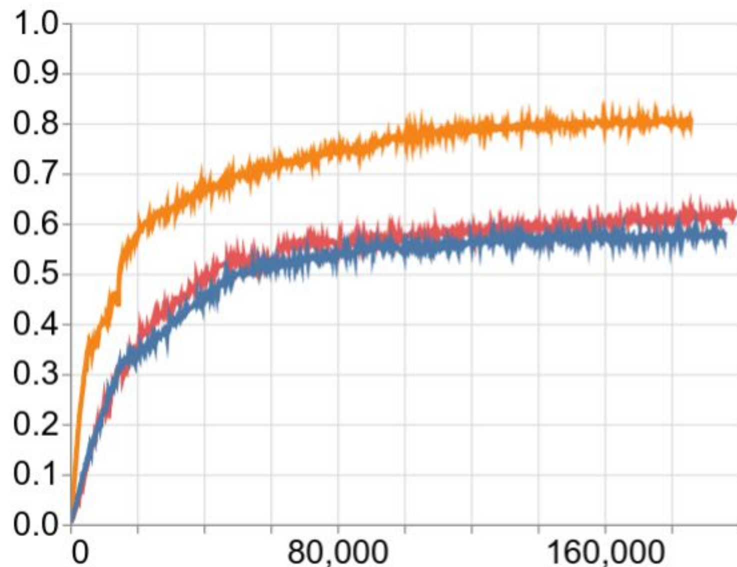
$$k \in \mathcal{O}\left(\operatorname{tr}\left(H(\theta^*)^{-1}\Sigma\right)\frac{1}{\epsilon}\right) \quad \text{vs} \quad k \in \mathcal{O}\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$$

no log!

Stochastic 2nd-order and momentum methods

- Mini-batch gradients estimates can be used with 2nd-order and momentums methods too
- Curvature matrices estimated stochastically using decayed averaging over multiple steps
- No stochastic optimization method that sees the same amount of data can have better **asymptotic** convergence speed than SGD with Polyak averaging
- *But…* **pre-asymptotic** performance usually matters more in practice. So stochastic 2nd-order and momentum methods can still be useful if:
  - the loss surface curvature is bad enough and/or
  - the mini-batch size is large enough

Experiments on deep convnets



Details

- Mini-batch size of 512
- Imagenet dataset
- 100 layer deep convolutional net without skips or batch norm
- Carefully initialized parameters

Conclusions / Summary

- Optimization methods:
  - enable learning in models by adapting parameters to minimize some objective
  - main engine behind neural networks
- 1st-order methods (gradient descent):
  - take steps in direction of “steepest descent”
  - run into issues when curvature varies strongly in different directions
- Momentum methods:
  - use principle of momentum to accelerate along directions of lower curvature
  - obtain “optimal” convergence rates for 1st-order methods

Conclusions / Summary

- 2nd-order methods:
  - improve convergence in problems with bad curvature, even more so than momentum methods
  - require use of trust-regions/damping to work well
  - also require the use of curvature matrix approximations to be practical in high dimensions (e.g. for neural networks)
- Stochastic methods:
  - use “mini-batches” of data to estimate gradients
  - asymptotic convergence is slower
  - pre-asymptotic convergence can be sped up using 2nd-order methods and/or momentum

Thank you

Questions

References and further reading

Solid introductory texts on optimization:

- *Numerical Optimization* (Nocedal & Wright)
- *Introductory Lectures on Convex Optimization: A Basic Course* (Nesterov)

Further reading for those interested in neural network optimization:

- *Optimization Methods for Large-Scale Machine Learning* (Bottou et al)
- *The Importance of Initialization and Momentum in Deep Learning* (Sutskever et al.)
- *New insights and perspectives on the natural gradient method* (Martens)
- *Optimizing Neural Networks with Kronecker-factored Approximate Curvature* (Martens & Grosse)
- *Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model* (Zhang et al.)