# Optimization Techniques for Neural Networks

Kapil Thadani
kapil@cs.columbia.edu

YAHOO!
RESEARCH

# Outline

- Learning as optimization

- Gradient descent
  - Stochastic gradient descent
  - Momentum
  - Nesterov accelerated gradient

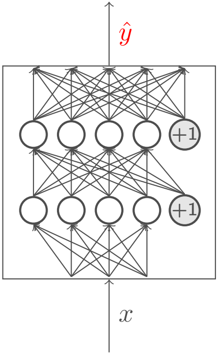
- Adaptive learning rates
  - Adagrad
  - RMSprop
  - Adadelta
  - Adam

- Second-order methods
  - Newton's method
  - L-BFGS
  - Hessian-free optimization

- Improving further
  - Unit tests for optimization
  - Learning to learn

# Prediction

Given network weights $\theta$ and new datapoint $x$, predict label $\hat{y}$



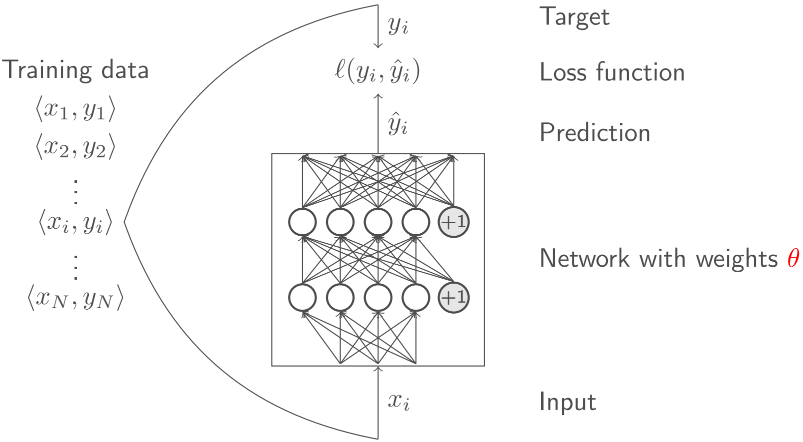

Prediction

Network with weights $\theta$

Input

# Learning

Given $N$ training pairs $\langle x_i, y_i \rangle$, learn network weights $\theta$



# Learning as optimization

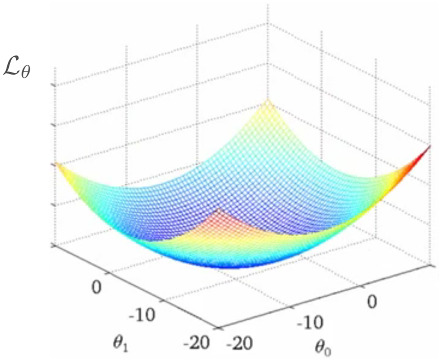
Minimize expected loss over training dataset (a.k.a. empirical risk)

$$\theta^* \;=\; \arg\min_\theta \mathbb{E}\,\ell_\theta \;=\; \arg\min_\theta \sum_{i=1}^{N} \ell_\theta(y_i, \hat{y}_i) \;=\; \arg\min_\theta \mathcal{L}_\theta$$

# Learning as optimization

Minimize expected loss over training dataset (a.k.a. empirical risk)

$$\theta^* \;=\; \arg\min_\theta \mathbb{E}\,\ell_\theta \;=\; \arg\min_\theta \sum_{i=1}^{N} \ell_\theta(y_i, \hat{y}_i) \;=\; \arg\min_\theta \mathcal{L}_\theta$$



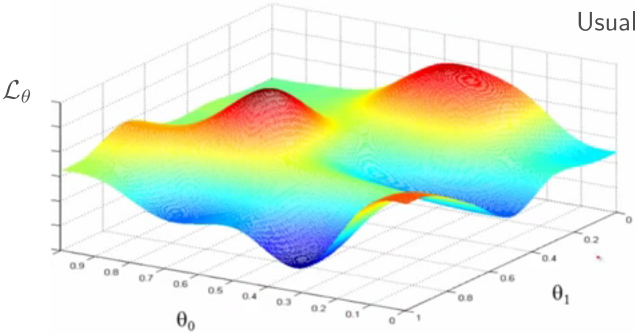

Ideally convex loss surface

# Learning as optimization

Minimize expected loss over training dataset (a.k.a. empirical risk)

$$\theta^* = \arg\min_\theta \mathbb{E}\,\ell_\theta = \arg\min_\theta \sum_{i=1}^{N} \ell_\theta(y_i, \hat{y}_i) = \arg\min_\theta \mathcal{L}_\theta$$

Usually non-convex



# Gradient descent

Given weights $\theta = \langle w_{11}, w_{12} \cdots w_{ij} \cdots \rangle^\top$, the gradient of $\mathcal{L}$ w.r.t. $\theta$

$$\nabla \mathcal{L} = \left\langle \frac{\partial \mathcal{L}}{\partial w_{11}}, \frac{\partial \mathcal{L}}{\partial w_{12}} \cdots \frac{\partial \mathcal{L}}{\partial w_{ij}} \cdots \right\rangle^\top$$

always points in the direction of steepest increase

Algorithm:

1. Initialize some $\theta_0$
2. Compute $\nabla \mathcal{L}$ w.r.t. $\theta_t$
3. Update in direction of negative gradient with some step size $\eta$

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}$$

4. Iterate until convergence

# Stochastic gradient descent (SGD)



$\nabla\mathcal{L}$ was computed over the full dataset for each update!

Instead update $\theta$ with every training example (i.e., online learning)

$$\theta_{t+1} = \theta_t - \eta \nabla \ell(y_i, \hat{y}_i)$$

or in mini-batches

$$\theta_{t+1} = \theta_t - \eta \sum_{j=i}^{i+k} \nabla \ell(y_j, \hat{y}_j)$$

Advantages:

- + Fewer redundant gradient computations, i.e., faster
- + Parallelizable, optional asynchronous updates
- + High-variance updates can hop out of local minima
- + Can encourage convergence by annealing the learning rate

# Momentum



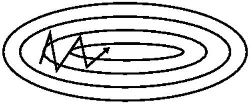
Gradient descent can be stopped by small bumps (though SGD helps) and can oscillate continuously in long, narrow valleys

Can simply combine current weight update with previous update

$$m_{t+1} = \mu\, m_t - \eta \nabla \ell \qquad \text{"velocity"}$$
$$\theta_{t+1} = \theta_t + m_{t+1} \qquad \text{"position"}$$

where $\mu$ is a hyperparameter (typically 0.9, sometimes annealed)

Without momentum



With momentum

Advantages:

+ Dampened oscillations and faster convergence

# Nesterov accelerated gradient (NAG)

Nesterov (1983)

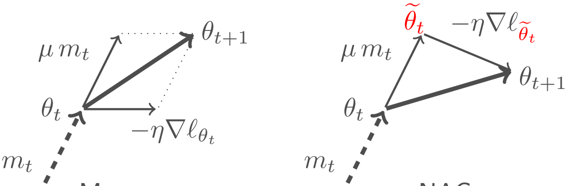
Now we can somewhat anticipate the update direction with momentum, but we still compute gradient w.r.t. $\theta_t$

Instead consider gradient at $\theta_t + \mu\, m_t$ accounting for future momentum

$$\widetilde{\theta}_t = \theta_t + \mu\, m_t$$

$$m_{t+1} = \mu\, m_t - \eta \nabla \ell_{\widetilde{\theta}_t}$$

$$\theta_{t+1} = \theta_t + m_{t+1}$$




Momentum NAG

Advantages:

+ Stronger theoretical guarantees for convex loss
+ Slightly better in practice than standard momentum

# Adagrad

Duchi et al. (2011)

Inputs and activations can vary widely in scale and frequency, but they are always updated with the same learning rate $\eta$ (or $\eta_t$)

Here, each parameter's learning rate is normalized by the RMS of accumulated gradients

$$\begin{aligned} v_{t+1} &= v_t + (\nabla \ell_{\theta_t})^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_{t+1} + \epsilon}} \nabla \ell_{\theta_t} \end{aligned}$$

where $\epsilon$ avoids division by zero

Advantages:

+ Lower learning rate for parameters with large/frequent gradients
+ Higher learning rate for parameters with small/rare gradients
+ $\eta$ doesn't need much tuning (typically 0.01)

# RMSprop

Tieleman & Hinton (2012)

Learning rates in Adagrad accumulate monotonically in the denominator, eventually halting progress

Normalize each gradient by a moving average of squared gradients (originally developed to improve adaptative rates across mini-batches)

$$v_{t+1} = \rho\, v_t + (1-\rho)\left(\nabla \ell_{\theta_t}\right)^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_{t+1}+\epsilon}} \nabla \ell_{\theta_t}$$

where $\rho$ is a decay rate (typically 0.9)

Advantages:

+ Exponentially decaying average prevents learning from halting prematurely

# Adadelta

Zeiler (2012)

Learning rates in Adagrad accumulate monotonically (observed again), and updates to $\theta$ seem to have the wrong "units", i.e., $\propto \frac{1}{\theta}$

Exponentially decaying average of squared gradients (again), and correcting units with Hessian ($\nabla^2 \ell$) approximation

$$v_{t+1} = \rho\, v_t + (1-\rho)\left(\nabla \ell_{\theta_t}\right)^2$$

$$\Delta\theta_{t+1} = -\frac{\sqrt{(\Delta\theta_t)^2 + \epsilon}}{\sqrt{v_{t+1} + \epsilon}} \nabla \ell_{\theta_t}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_{t+1}$$

Advantages:

+ No learning rate hyperparameter!
+ Numerator acts as an acceleration term like momentum
+ Robust to large, sudden gradients by reducing learning rate
+ Hessian approximation is efficient and always positive

# Adaptive Moment Estimation (Adam)

Kingma & Ba (2015)

Momentum and adaptive learning rates are estimates of moments of $\nabla \ell$

$$m_{t+1} = \beta_1 \, m_t + (1 - \beta_1) \nabla \ell_{\theta_t} \qquad 1^{\text{st}} \text{ moment estimate}$$

$$v_{t+1} = \beta_2 \, v_t + (1 - \beta_2) \left( \nabla \ell_{\theta_t} \right)^2 \qquad 2^{\text{nd}} \text{ moment estimate}$$

Correct for biases at initialization when moment estimates are 0

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - (\beta_1)^{t+1}}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - (\beta_2)^{t+1}}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{t+1} + \epsilon}} \hat{m}_{t+1}$$

with hyperparameters $\beta_1$ (typically 0.9) and $\beta_2$ (typically 0.999)

Advantages:

+ Regret bound comparable to best known
+ Works well in practice

# Visualizations



http://imgur.com/a/Hqolp

# Newton's method

Second-order Taylor approximation of $\mathcal{L}(\theta)$ around $\theta_t$:

$$\mathcal{L}(\theta_t + \Delta\theta) \;\approx\; \mathcal{L}(\theta_t) + \nabla\mathcal{L}(\theta_t)^\top \Delta\theta + \frac{1}{2}\Delta\theta^\top H_t \Delta\theta$$

where the Hessian $H_t = \nabla^2\mathcal{L}(\theta_t)$ is an $n \times n$ matrix

To minimize this, compute the gradient w.r.t. $\Delta\theta$ and set it to 0

$$\begin{aligned} \nabla\mathcal{L}(\theta_t + \Delta\theta) \;\approx\; \nabla\mathcal{L}(\theta_t) + H_t\Delta\theta &= 0 \\ \Delta\theta &= -H_t^{-1}\nabla\mathcal{L}(\theta_t) \end{aligned}$$

Algorithm:

1. Initialize some $\theta_0$
2. Compute $\nabla\mathcal{L}_{\theta_t}$ and $H_t$ w.r.t. current $\theta_t$
3. Determine $\eta$, e.g., with backtracking line search
4. Update towards minimum of local quadratic approximation around $\theta_t$

$$\theta_{t+1} = \theta_t - \eta H_t^{-1}\nabla\mathcal{L}_{\theta_t}$$

5. Iterate until convergence

# Quasi-Newton methods: L-BFGS



Expensive to compute and store $H_t$, so we approximate $H_t \succ 0$ (or $H_t^{-1}$)

e.g., BFGS update

$$s = \theta_t - \theta_{t-1} \qquad z = \nabla \mathcal{L}_{\theta_t} - \nabla \mathcal{L}_{\theta_{t-1}}$$

$$H_t = H_{t-1} - \frac{zz^\top}{z^\top s} - \frac{H_{t-1} s s^\top H_{t-1}}{s^\top H_{t-1} s}$$

$$\text{or} \quad H_t^{-1} = \left( I - \frac{sz^\top}{z^\top s} \right) H_{t-1}^{-1} \left( I - \frac{zs^\top}{z^\top s} \right) + \frac{ss^\top}{z^\top s}$$

Limited-memory BFGS (L-BFGS): store only the $m$ most recent values of $s$ and $z$ instead of $H_t^{-1}$

Advantages:

+ Good global and local convergence bounds
+ Cost per iteration $\mathcal{O}(mn)$ while Newton's method is $\mathcal{O}(n^3)$
+ Storage is $\mathcal{O}(mn)$ instead of $\mathcal{O}(n^2)$ for storing $H_t$

# Hessian-free optimization



Martens (2010)

Minimize second-order Taylor expansion of $\mathcal{L}(\theta)$ with conjugate gradient

1. Set initial direction $d_0 = \nabla \mathcal{L}_{\theta_0}$
2. Update $\theta_{t+1} = \theta_t + \alpha d_t$ with $\alpha = d_t^\top (H_t \theta_t + \nabla \mathcal{L}_{\theta_t}) / d_t^\top H_t d_t$
3. Update $d_{t+1} = -\nabla \mathcal{L}_{\theta_{t+1}} + \beta d_t$ where $\beta = \nabla \mathcal{L}_{\theta_{t+1}}^\top H_t d_t / d_t^\top H_t d_t$
4. Iterate up to $n$ times

Requires only Hessian-vector products $H_t v$

- Equivalent to directional derivative of $\nabla \mathcal{L}_{\theta_t}$ in the direction $v$
- Approximate with finite differences, etc

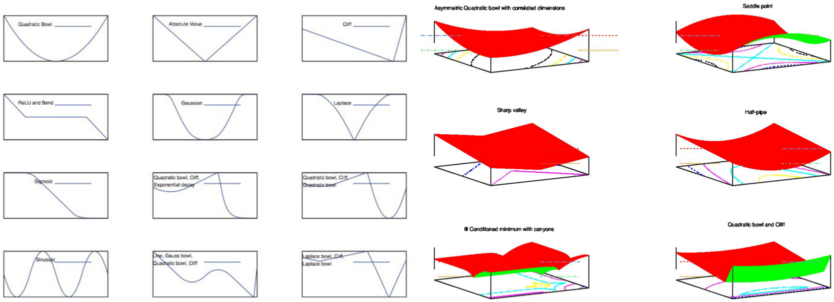
Advantages:

+ Scales to very large datasets
+ Empirically leads to lower training error than first-order methods
+ Can be made faster by pre-training conjugate gradient, etc

# Improving further

Unit tests for stochastic optimization (Schaul et al., 2014)

- Construct synthetic optimization landscapes with known difficulties
- Use these to benchmark and analyze optimization algorithms




1-D prototypes 2-D combinations

# Improving further

Learning to learn by gradient descent by gradient descent (Andrychowicz et al., 2016)

- Replace hand-designed update rules with a learned update rule

$$\theta_{t+1} = \theta_t + g_\phi(\nabla \ell_{\theta_t})$$

- Model $g$ as outputs of a recurrent neural network (RNN) with parameters $\phi$