Optimization Techniques for Neural Networks

Kapil Thadani
kapil@cs.columbia.edu

YAHOO! RESEARCH

Outline

- Learning as optimization

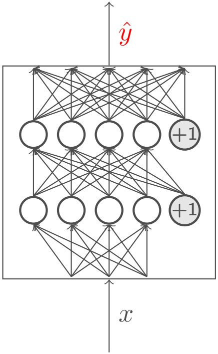
- First-order methods
  - Stochastic gradient descent
  - Momentum
  - Nesterov accelerated gradient
  - Adagrad
  - RMSprop
  - Adadelta
  - Adam
  - Adamax
  - Nadam
  - AMSgrad

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

- Improving further

Prediction

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



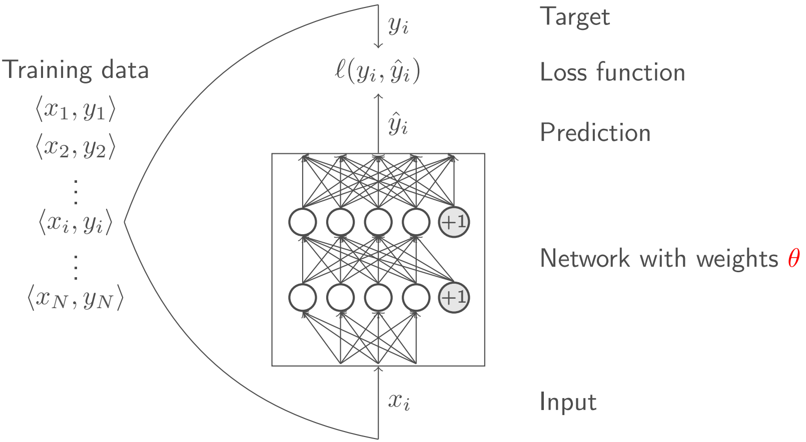
Prediction

Network with weights θ

Input

Learning

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



Learning as optimization

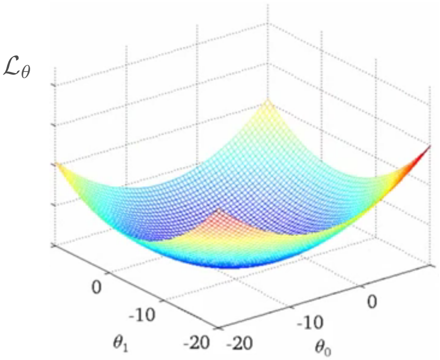
Minimize expected loss over training dataset (a.k.a. empirical risk)

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

Learning as optimization

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

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



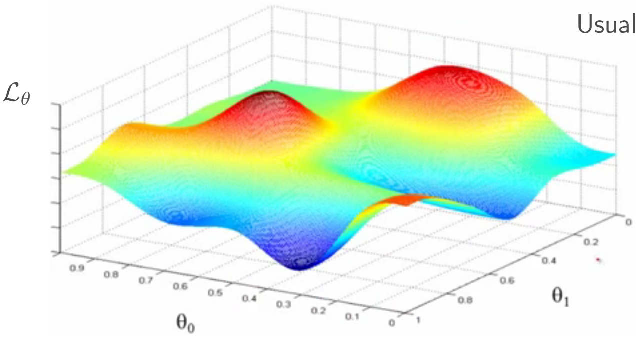
Ideally convex loss surface

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. θ

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

always points in the direction of steepest increase

Algorithm:

1. Initialize some θ_0
2. Compute $\nabla \mathcal{L}$ w.r.t. θ_t
3. Update in direction of negative gradient with some step size η

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

4. Iterate until convergence

Stochastic gradient descent (SGD)

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

Instead update θ 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 μ is a hyperparameter (typically 0.9, sometimes annealed)

Without momentum With momentum

Advantages:

+ Dampened oscillations and faster convergence

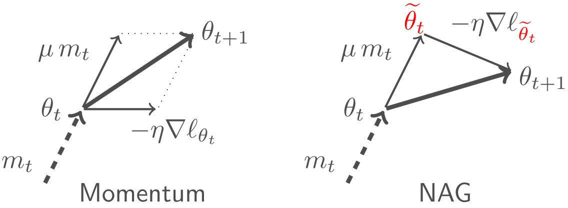
Nesterov accelerated gradient (NAG)

Nesterov (1983)

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

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

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



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 η (or η_t)

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

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

where ϵ avoids division by zero

Advantages:

+ Lower learning rate for parameters with large/frequent gradients
+ Higher learning rate for parameters with small/rare gradients
+ η 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 ρ 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 θ 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

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

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

Intermission: Visualizations

http://imgur.com/a/Hqolp

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

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-(\beta_1)^{t+1}} \qquad \hat{v}_{t+1} = \frac{v_{t+1}}{1-(\beta_2)^{t+1}}$$

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

with hyperparameters β_1 (typically 0.9) and β_2 (typically 0.999)

Advantages:

+ Update steps bounded by *trust region*: $\left|\frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}}}\right| < \max\left(\frac{1-\beta_1}{\sqrt{1-\beta_2}}, 1\right)$
+ Works well in practice

Adamax

Kingma & Ba (2015)

Scale gradients proportional to L_∞ norm of past gradients instead of L_2

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

$$u_{t+1} = \max\left(\beta_2 \cdot u_t, |\nabla \ell_{\theta_t}|\right) \qquad \text{exp-weighted } L_\infty \text{ norm}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{1-(\beta_1)^{t+1}} \frac{m_{t+1}}{u_{t+1}}$$

Advantages:

+ L_p norms with $p > 2$ are not stable, but this is
+ No need for bias correction for u_t

Nesterov-accelerated Adam (Nadam)

Dozat (2016)

Nesterov-accelerated momentum for Adam

$$m_{t+1} = \beta_1 \, m_t + (1 - \beta_1) \nabla \ell_{\theta_t} \qquad v_{t+1} = \beta_2 \, v_t + (1 - \beta_2) \left(\nabla \ell_{\theta_t}\right)^2$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - (\beta_1)^{t+1}} \qquad \hat{v}_{t+1} = \frac{v_{t+1}}{1 - (\beta_2)^{t+1}}$$

Anticipate future momentum from current gradient

$$\widetilde{m}_{t+1} = \beta_1 \hat{m}_{t+1} + \frac{1 - \beta_1}{1 - \beta_1^t} \nabla \ell_{\theta_t}$$

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

Advantages:

+ Significant improvements over Adam on some tasks

AMSgrad

Reddi et al. (2018)

Exponentially-moving averages do not guarantee a non-increasing learning rate over minibatches, leading to convergence issues for RMSprop, Adam, etc

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla \ell_{\theta_t} \qquad v_{t+1} = \beta_2 v_t + (1 - \beta_2)\left(\nabla \ell_{\theta_t}\right)^2$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - (\beta_1)^{t+1}} \qquad \hat{v}_{t+1} = \frac{v_{t+1}}{1 - (\beta_2)^{t+1}}$$

Scale gradients with the maximum over current and past gradients

$$\widetilde{v}_{t+1} = \max(\widetilde{v}_t, \hat{v}_{t+1})$$

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

Advantages:

+ Regret bound comparable to best known
+ Initial results look promising
+ May explain problems with adaptive methods (Wilson et al. 2018)

Newton's method

Second-order Taylor approximation of $\mathcal{L}(\theta)$ around θ_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 θ_0
2. Compute $\nabla\mathcal{L}_{\theta_t}$ and H_t w.r.t. current θ_t
3. Determine η, e.g., with backtracking line search
4. Update towards minimum of local quadratic approximation around θ_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

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

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

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

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

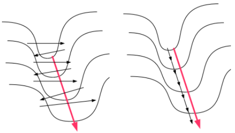
Martens (2010), Martens & Sutskever (2011)

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
- Can approximate with finite differences, etc
- Gauss-Newton matrix $G \succ 0$ instead of H
- Tricks: damping, termination conditions, etc



Advantages:

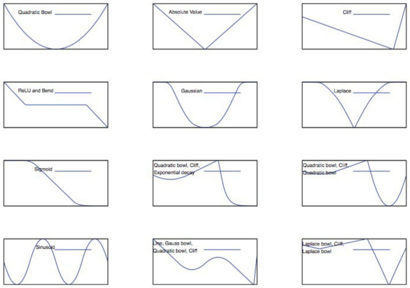
+ Scales to very large datasets
+ Empirically leads to lower training error than first-order methods
+ Can be made faster by pre-training conjugate gradient, etc

Improving further

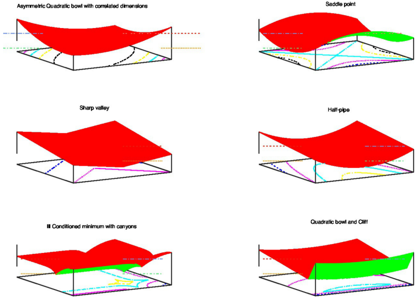
Schaul et al. (2014)

Unit tests for stochastic optimization

Synthetic optimization landscapes with known difficulties used to benchmark and analyze optimization algorithms



1-D prototypes



2-D combinations

Improving further

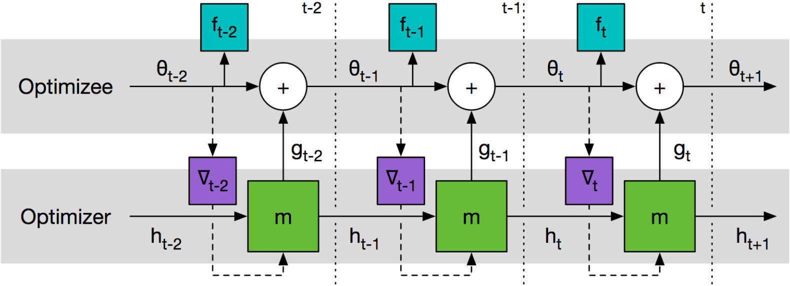
Andrychowicz et al. (2016)

Learning to learn by gradient descent by gradient descent

Learned update rule instead of hand-designed algorithms

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

where g is modeled as outputs of a recurrent neural network (RNN) with parameters ϕ



Improving further

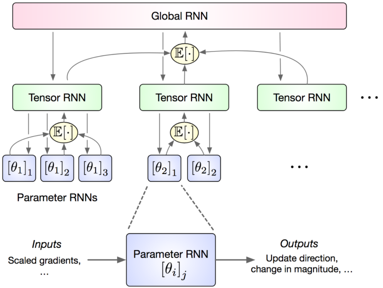
Wichrowska et al. (2017)

Learned optimizers that scale and generalize

Hierarchical RNN structure to track state for individual parameters, parameter tensors (e.g., layers) and globally

Input:

- Momentum on multiple timescales scaled by L_2 norm of avg gradients
- Average gradient magnitudes
- Relative learning rate



Output:

- Direction updates
- Learning rate update
- Momentum hyperparameters

+ Improvements on MNIST compared to Adam, RMSprop
+ Competitive with non-learned optimizers on new problems