Optimization for Training Deep Models

Jie Tang

Tsinghua University

November 5, 2019

Overview

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Outline

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

How Learning Differs from Pure Optimization?

- Typically, the cost function can be written as an average over the training set, such as

$$J(\theta) = \mathbb{E}_{(x,y)\sim\hat{p}_{data}} L(f(x;\theta), y), \tag{1}$$

- Minimize the corresponding objective function where the expectation is taken across the data generating distribution p_{data}:

$$J^*(\theta) = \mathbb{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y), \tag{2}$$

Empirical Risk Minimization

- The goal of a machine learning algorithm is to reduce the expected generalization error given by Eq. 2.
- When we do not know $p_{data}(x, y)$, we minimize the *empirical risk*

$$\mathbb{E}_{x,y\sim\hat{p}_{data}(x,y)}[L(f(x;\theta), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)}) \quad (3)$$

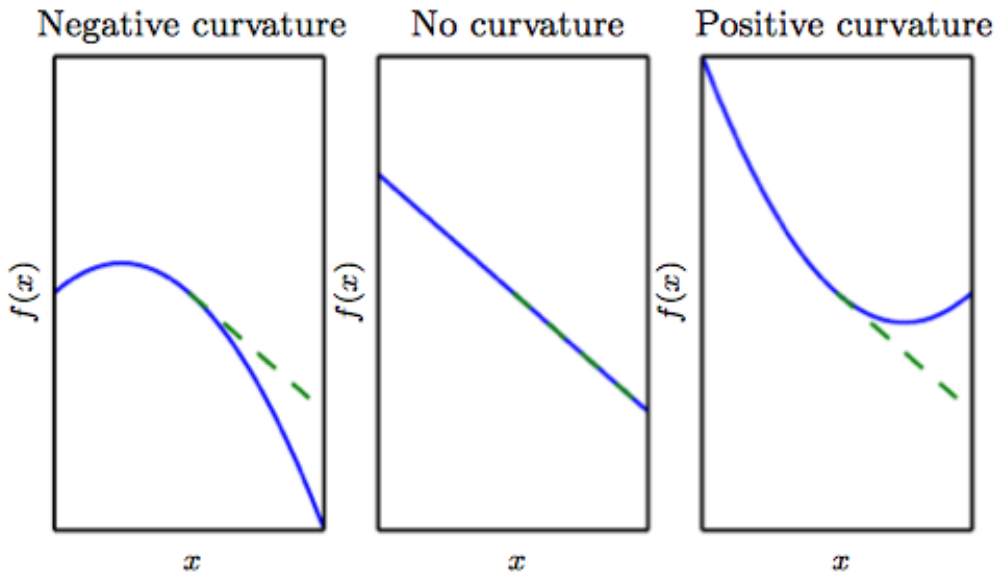
  - Empirical risk minimization is prone to overfitting – simply memorize the training set.
  - Empirical risk minimization is not really feasible. Many useful loss functions, such as 0-1 loss, have no useful derivatives to perform gradient descent.

Surrogate Loss Functions and Early Stopping

- *surrogate loss function*.
  - For example, the negative log-likelihood of the correct class is typically used as a surrogate for the 0-1 loss.
  - A surrogate loss function results in being able to learn more. When the expected 0-1 loss is zero, one can improve the robustness of the classifier by further pushing the classes apart from each other.
- A machine learning algorithm usually minimizes a surrogate loss function but halts when a convergence criterion based on early stopping is satisfied.

Preliminaries: First-order and Second-order Derivatives

Hessian matrix



- **Gradient**

$$g = \nabla_\theta J(\theta) \qquad (4)$$

- **Hessian matrix**

$$H_{i,j} = \frac{\partial}{\partial \theta_j} \frac{\partial J(\theta)}{\partial \theta_i} \qquad (5)$$

Directional Curvature

Hessian matrix:

- H is symmetric, so there exists

$$Q = [v_1, ..., v_n] \tag{6}$$

$$H = Q \Lambda Q^\top \tag{7}$$

- A second derivative in direction d is given by:

$$d^\top H d \tag{8}$$

Taylor series approximation

- Taylor expansion: if $f(x)$ and $J(\theta)$ has second-order derivatives, then we have

$$
\begin{aligned}
f(x) &\approx f(x_0) + (x - x_0) f'(x) + \frac{1}{2}(x - x_0)^2 f''(x) \\
J(\theta) &\approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)
\end{aligned}
\tag{9}
$$

How much does a gradient step improve?

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0) \quad (10)$$

- Let $g = \nabla_\theta J(\theta_0)$, ϵ is step size, $\Lambda = \{\lambda_i\}$ are eigenvalues of Hessian matrix H, then we have

$$J(\theta_0 - \epsilon g) = J(\theta_0) - \epsilon g^T g + \frac{1}{2}\epsilon^2 g^T H g$$

- essentially, we have two new terms in the gradient descent

$$\frac{1}{2}\epsilon^2 g^\top H g - \epsilon g^\top g \quad (11)$$

- Ill-conditioning of the Hessian matrix becomes a problem when $\frac{1}{2}\epsilon^2 g^\top H g$ exceeds $\epsilon g^\top g$.

Ill-Conditioning

- Optimal step size is:[1]

$$\epsilon^* = \frac{g^\top g}{g^\top H g} \tag{12}$$

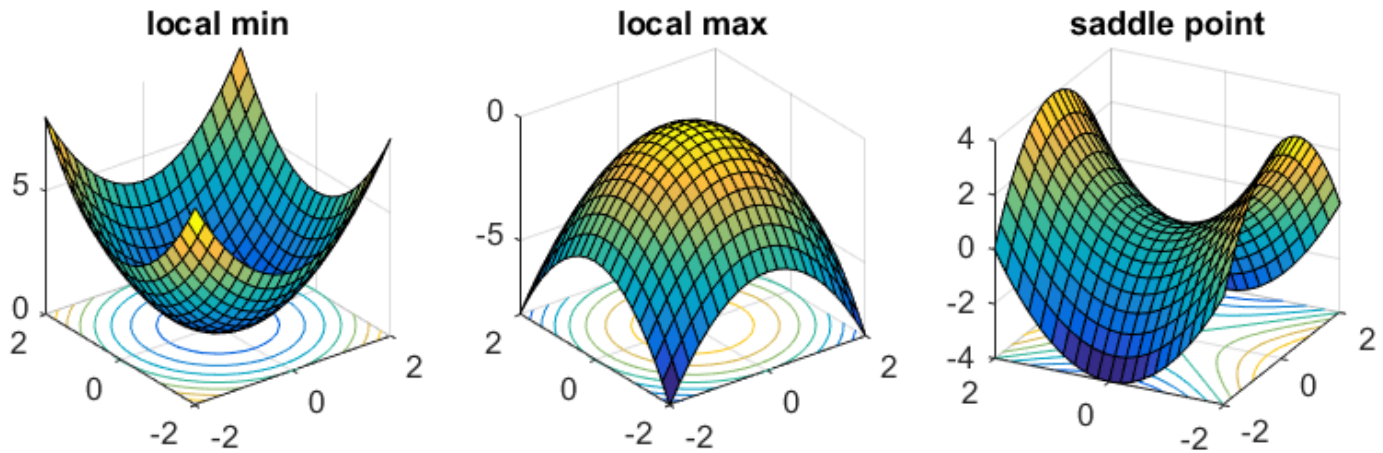
- if g aligns with a large eigenvalue, ϵ^* is very small; when g aligns with $\lambda_{\max}$, the improvement is the worst.
- if ϵ is not chosen to optimal step, the cost may go uphill.
- Usually choose ϵ fixed in advance, and $g^\top H g$ can fluctuate rapidly if H has many distinct eigenvalues.

[1] when $g^\top H g \leq 0$, all step size ϵ will reduce the cost.

Critical points

Zero gradient, with Hessian matrix

- all positive eigenvalues – local minimum
- all negative eigenvalues – local maximum
- mixture – saddle point – local minimum at some axes and local maximum at some axes



Outline

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Local Minima

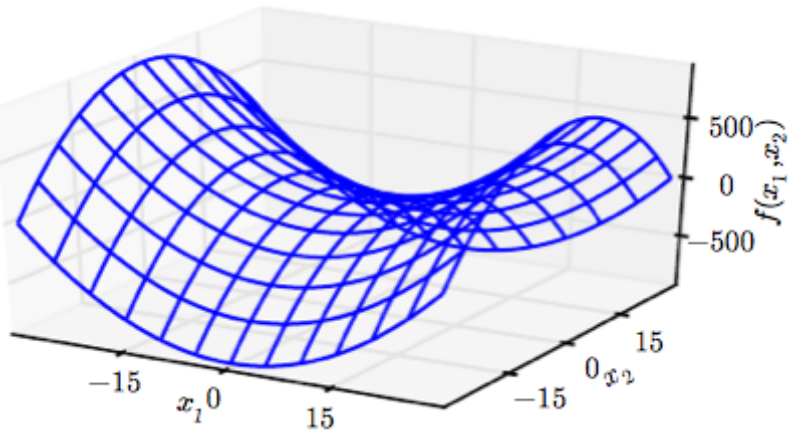
- One of the most prominent features of a convex optimization problem is that it can be reduced to the problem of finding a local minimum. Any local minimum is guaranteed to be a global minimum.
- Some convex functions have a flat region at the bottom rather than a single global minimum point, but any point within such a flat region is an acceptable solution.
- When optimizing a convex function, we know that we have reached a good solution if we find a critical point of any kind.

Local Minima

- DNN usually has non-convex functions, thus it is possible to have many local minima.
- The problem of finding a local minimum of deep model is not necessarily a major problem.
- Causes of local minimal:
  - Model identifiability problem. e.g., weight space symmetry.

Saddle Points and Other Flat Regions

- At a saddle point, the Hessian matrix has both positive and negative eigenvalues.



- Gradient descent empirically seems to be able to escape saddle points in many cases.
- Only Newton's methods and related techniques are attracted to saddle points.

Gradient descent escapes saddle points

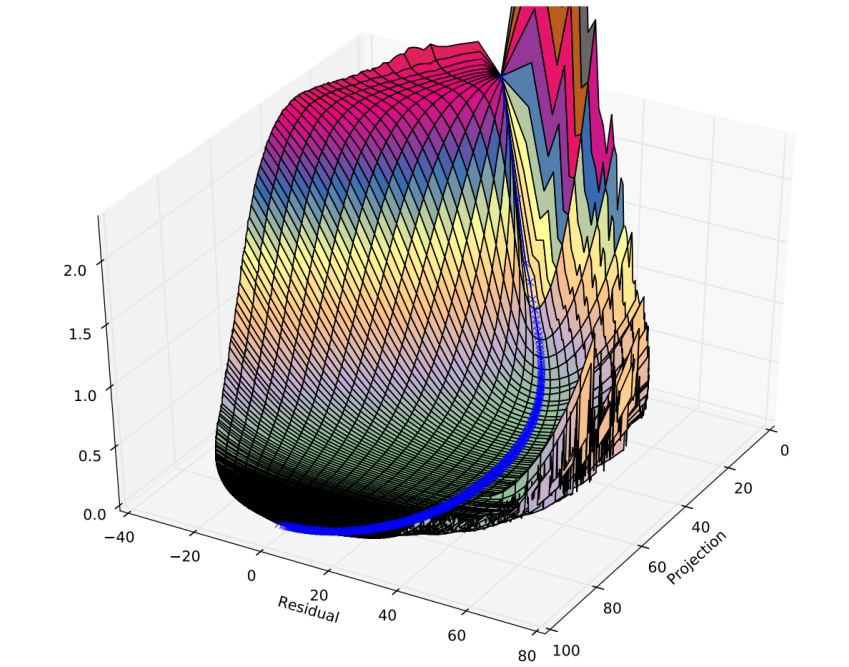


Figure 1: (a) Parameters are initialized saddle point of high cost, but, as indicated by the blue path, the SGD training trajectory escapes this saddle point readily. (b) Most of training time is spent traversing the relatively flat valley of the cost function, which may be due to high noise in the gradient, poor conditioning of the Hessian matrix in this region, or simply the need to circumnavigate the tall "mountain".

Saddle Points and Local Minima

- In higher dimensional spaces, local minima are rare and saddle points are more common.
- For a function: $f : \mathbb{R}^n \to \mathbb{R}$, the expected ratio of the number of saddle points to local minima grows exponentially with n.
  - Local minimum $\to$ positive eigenvalues.
  - Saddle points $\to$ mixture of positive and negative eigenvalues.
  - Imagine flipping a coin, if heads, the eigenvalue is positive, if tails, negative.
  - Then ...

Saddle points

- An amazing property of many random functions is that the eigenvalues of the Hessian become more likely to be positive as we reach regions of lower cost.
- This means that local minima are much more likely to have low cost than high cost.
- **Implication:** most minima are good, and it's more true for big models.

Local Minima and Global Minima

- In recent years, theoretical anaylsis shows that if some constraints being added, then all local minimas become global minimas.
- For Deep Linear Networks (deep neural networks without activation functions), all local minimas become global minimas, if it satisfies the following conditions(Laurent & Brecht 2018):
- 1. The thinnest layer is either the input or the output layer.

$$min(W_0, W_l) = min(W_i | 0 \leq i \leq l) \tag{13}$$

- 2. The loss function $L(\hat{y}, y)$ is convex and differentiable.

Local Minima and Global Minima

- For Deep Linear Networks, if another series of condition holds, all local minimas are also global minimas. (Lu & Kawaguchi 2017)
- 1. The loss function is the mean squared loss.

$$L(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{2} \sum_{i=1}^{n} ||\hat{y}^{(i)} - y^{(i)}||^2 \tag{14}$$

- 2. The data matrix $\mathbf{X} = [\mathbf{x}_1, ...\mathbf{x}_n]$ and the label matrix $\mathbf{Y} = [\mathbf{x}_1, ...\mathbf{x}_n]$ is of full row rank.

$$Rank(\mathbf{X}) = Rank(\mathbf{Y}) = N \tag{15}$$

Local Minima and Global Minima

- For very wide Deep Neural Networks, Gradient Descent also guarantees the global minima. (Du et al. 2018) The loss is the mean squared loss.
- For fully-connected networks, if

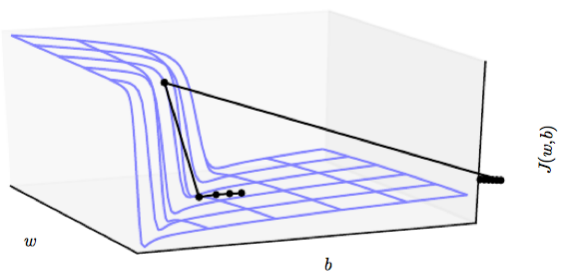
$$W \gtrsim n^4 2^{O(H)}, T \gtrsim n^2 2^{O(H)} log(\frac{1}{\epsilon}), \tag{16}$$

(W represents width, H represents depth, n is the number of samples, T is the number of time steps) we can guarantee a point with $L < \epsilon$ on the training set.

- For Resnet, the conditions become

$$W \gtrsim n^4 H^2, T \gtrsim n^2 log(\frac{1}{\epsilon}). \tag{17}$$

Cliffs and Exploding Gradients



- Neural networks with many layers often have extremely steep regions resembling cliffs (multiplication of large weights).
- When the parameters get close to such a cliff region, a gradient descent update can move the parameters very far, possibly losing most of the optimization work that had been done.
- The cliff is dangerous but can be avoided by gradient clipping (discussed later).
- Cliff structures are most common in the loss functions for recurrent neural networks, since they involve multiplication of many factors with one factor for each time step.

Long-Term Dependencies

Another difficulty that neural network optimization algorithms must overcome is long-term dependencies.

- For recurrent network, after multiplying a weight matrix W for t steps, this is equivalent to multiplying by W^t.
- Suppose that W has an eigen-decomposition $W = V\text{diag}(\lambda)V^{-1}$, i.e.,

$$W^t = V\text{diag}(\lambda)^t V^{-1} \tag{18}$$

- Vanishing and exploding gradient problem: gradients through such a graph are scaled according to $diag(\lambda)^t$.
- Recurrent networks use the same matrix W at each time step, but feedforward networks do not, so it can avoid the vanishing and exploding gradient problem.

Inexact Gradients

- **Noise.** A noisy or even biased estimate of gradients and Hessian because of sampling-based estimates, e.g., minibatch gradient/estimation.
- **Intractable objective.** Intractable objective function, thus intractable gradients.
  - The intractable log-likelihood of a Boltzmann machine: contrastive divergence gives a technique to approximate the gradient.

Some functions lack critical points

For example, the loss function $\log p(y|x;\theta)$ can lack a global minimum point and asymptotically approach some value as the model becomes more confident.

- For a softmax classifier, if the training set is separable, we can always marginally increase the confidence and reduce the negative log-likelihood, but it is impossible to actually reach the value of zero.

Some functions lack critical points

- Gradient descent often does not arrive at a critical point of any kind.

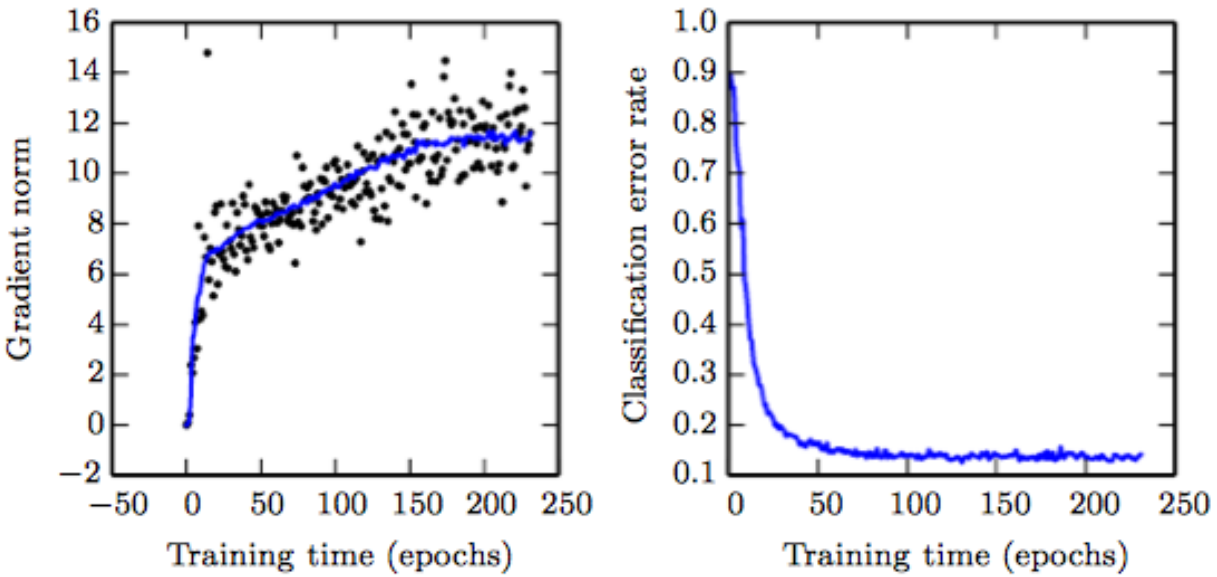
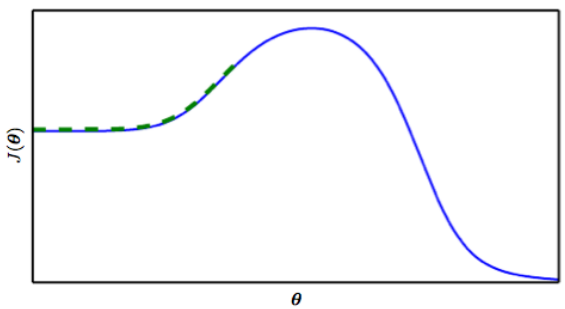


Figure 2: (*Left*) One gradient norm is plotted per epoch. The running average of all gradient norms is plotted as a solid curve. The gradient norm clearly increases over time, rather than decreasing if the training process converged to a critical point. (*Right*) Despite the increasing gradient, the training process is reasonably successful. The validation set classification error decreases to a low level.

Some functions lack critical points



Even hard

- There are no saddle points and no local minima.
- Initialized on the wrong side of the "mountain".
- Find good initial points.

Evidence

When getting in trouble with optimization, consider all possible obstacles and look for evidence,

- Local minima $\rightarrow$ gradient norm
- Poor condition $\rightarrow$ uphill steps and fluctuable $g^\top Hg$
- Inexact gradient $\rightarrow$ uphill steps and fluctuable g
- Saddle points $\rightarrow$ gradient norm and negative eigenvalue of H.

Outline

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) and its variants are the most used optimization algorithms for deep learning.

$$\theta \leftarrow \theta - \epsilon g \tag{19}$$

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

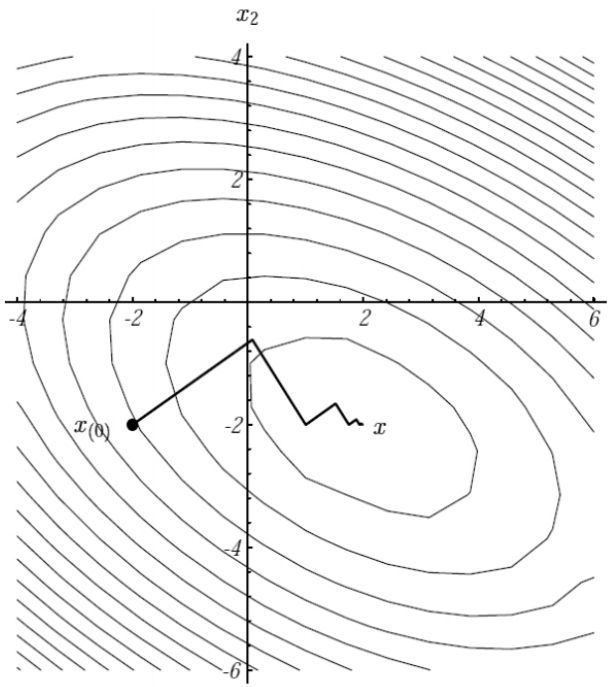
Stochastic Gradient Descent

- Learning rate ϵ is a crucial parameter.
- In practice, it is necessary to gradually decrease the learning rate over time.
  - Because the SGD gradient estimator introduces a source of noise (the random sampling of m training examples) that does not vanish even when we arrive at a minimum.
  - Denote the learning rate at iteration k as ϵ_k. It is common to decay the learning rate linearly until iteration τ:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{20}$$

where $\alpha = \frac{k}{\tau}$.

Stochastic Gradient Descent



The most important property of SGD

- Computation time per update does not grow with the number of training examples.
- The error surface may be curved differently in different directions. This means that the gradient does not necessarily point directly to the nearest local minimum.

Momentum

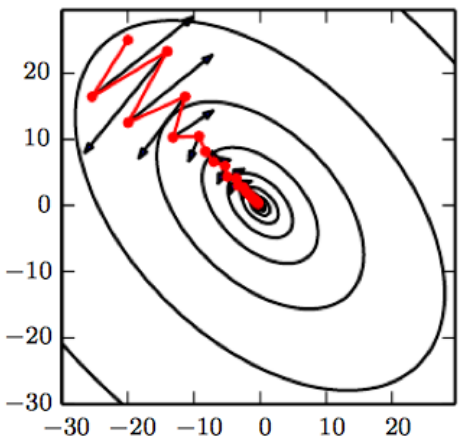
- The learning with SGD can be slow. If the error surface is a long and narrow valley, gradient descent goes quickly down the valley walls, but very slowly along the valley floor.
- Momentum is designed to accelerate learning.
- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

$$v = \alpha v - \epsilon \nabla_\theta [\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \tag{21}$$

$$\theta = \theta + v \tag{22}$$

- Usually α is set quite high, about .9, .99.
- This is like giving momentum to the weights.

Momentum



- Momentum learning in red, gradient descent represented by arrow.
- Solve two problems
  - solve poor conditioning of Hessian matrix.
  - solve the variance in the stochastic gradient.

Nesterov Momentum

$$
\begin{aligned}
v &= \alpha v - \epsilon \nabla_\theta [\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta + \alpha v), y^{(i)})], \\
\theta &= \theta + v
\end{aligned}
\tag{23}
$$

- The difference between Nesterov momentum and standard momentum is where the gradient is evaluated.
- Nesterov momentum adds a *correction factor* to the standard method of momentum.
- In batch gradient, Nesterov momentum brings the rate of convergence of the excess error from $O(1/k)$ (after k steps) to $O(1/k^2)$, but no improvement for SGD.

Outline

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Parameter Initialization Strategies

The initial point can

- determine whether the algorithm converges at all
- determine how quickly learning converges and whether it converges to a point with high or low cost.
- affect the generalization error.

Heuristics: to initialize the weights of a fully connected layer with m inputs and n outputs by sampling each weight from $U(\sqrt{\frac{1}{m}}, \sqrt{\frac{1}{m}})$, while Glorot and Bengio (2010) suggest using the *normalized initialization*

$$W_{i,j} \sim U(\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}}) \tag{24}$$

Outline

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Algorithms with Adaptive Learning Rates

Learning rate was reliably one of the hyperparameters that is the most difficult to set because it has a significant impact on model performance.

- The momentum algorithm can mitigate these issues somewhat, but introducing another hyperparameter.
- Is there another way?
- Use a separate learning rate for each parameter, and automatically adapt these learning rates.

AdaGrad

- Individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values.

$$\begin{aligned} &\text{Accumulated squared gradient: } r \leftarrow r + g \odot g \\ &\text{Compute update: } \Delta\theta \leftarrow -\frac{\epsilon}{\sigma + \sqrt{r}} \odot g \\ &\text{(Division and square root applied element-wise)} \\ &\text{Apply update: } \theta \leftarrow \theta + \Delta\theta \end{aligned} \tag{25}$$

  where ϵ is a global learning rate; and σ is a small constant (e.g., 10^{-7}).
- The parameters with the largest partial derivative have a rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate.

RMSProp

The RMSProp algorithm modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average.

$$\text{Accumulated squared gradient: } r \leftarrow \rho r + (1-\rho) g \odot g \tag{26}$$

where ρ is a hyperparameter that controls the length of the moving average.

RMSProp with Momentum

RMSProp combined with Nesterov momentum.

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Adam

- "Adam" stands for "adaptive moments".

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Choosing the Right Optimization Algorithm

Currently, the most popular optimization algorithms actively in use include

- SGD
- SGD with momentum
- RMSProp
- RMSProp with momentum
- Adam

Outline

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Approximate Second-Order Methods

In contrast to first-order methods, second-order methods make use of second derivatives to improve optimization.

Newton's Method

- A second-order Taylor series expansion to approximate $J(\theta)$ near some point θ_0

  $J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)$

  where H is the Hessian of J.
- If we then solve the critical point of this function, we obtain Newton parameter update rule:

$$\theta^* = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$$

  when H is positive definite.
  - for quadratic function, Newton's method jumps directly to the minimum.
  - this update can be iterated for a convex objective function.

Newton's Method

- This update can be iterated for a convex objective function.

$$\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)$$

Algorithm 8.8 Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$.

Require: Initial parameter $\boldsymbol{\theta}_0$
Require: Training set of m examples
  while stopping criterion not met **do**
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian inverse: $\boldsymbol{H}^{-1}$
    Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

- complexity of inverse H^{-1} is $\mathcal{O}(k^3)$.
- Newton's method works poorly for neural nets for its attraction to the ubiquitous saddle points.

Newton's Method

- If the eigenvalues of the Hessian are not all positive (e.g., near a saddle point), then Newtons method can cause updates to move in the wrong direction.
- Regularizing the Hessian.
  - Adding a constant, α, along the diagonal of the Hessian.

$$\theta^* = \theta_0 - [H + \alpha I]^{-1} \nabla_\theta J(\theta_0) \tag{27}$$

  - as α increases in size, the Hessian becomes dominated by the αI diagonal and it converges to the standard gradient divided by α.

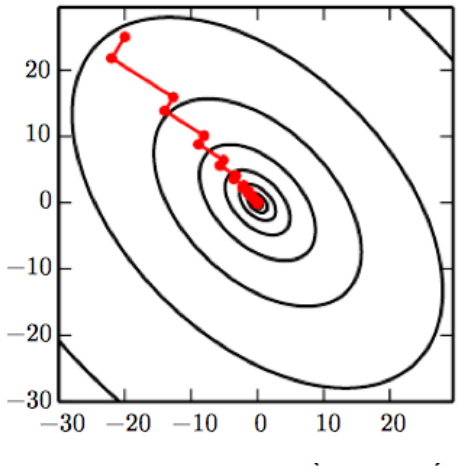
Second Order Methods

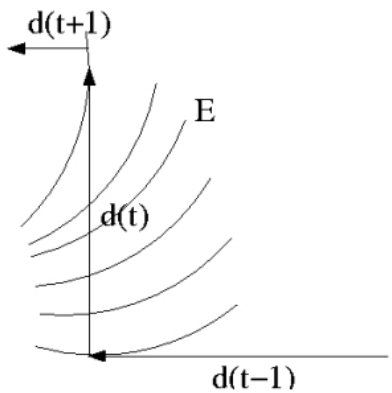
- Newton's method is an example of a *second order optimization*.
- Second order methods often converge much more quickly, but it can be very expensive to calculate and store the Hessian matrix.
- Often the sequence of gradients (first order derivatives) can be used to approximate the second order curvature. This can even be better than the true Hessian, because we can constrain the approximation to always be positive definite.

Newton and Quasi-Newton Methods

- Broyden-Fletcher-Goldfarb-Shanno (BFGS); ConjugateGradients (CG); Davidon-Fletcher-Powell (DVP); LevenbergMarquardt (LM)
- All approximate the Hessian using recent function and gradient evaluations (e.g., by averaging outer products of gradient vectors – BFGS, by tracking the "twist" in the gradient – CG; by projecting out previous gradient directions...).
- Then they use this approximate gradient to come up with a new search direction in which they do a combination of fixed-step, analytic-step and line-search minimizations.
- Very complex area – we will go through in detail only the CG method, and a bit of the limited-memory BFGS.

Conjugate Gradients





- Figure: The method of steepest descent applied to a quadratic cost surface. It jumps to the point of lowest cost along gradient line, but even with the optimal step size the algorithm still makes back-and-forth progress toward the optimum.
- Observation: d_t is (almost) orthogonal to d_{t-1}. The choice of orthogonal directions of descent do not preserve the minimum along the previous search directions.
- **Conjugate directions:** to first order, as we move in the new direction, the gradient parallel to the old direction stays zero.

Conjugate Gradients

Avoid the calculation of the inverse Hessian by iteratively descending *conjugate directions*.

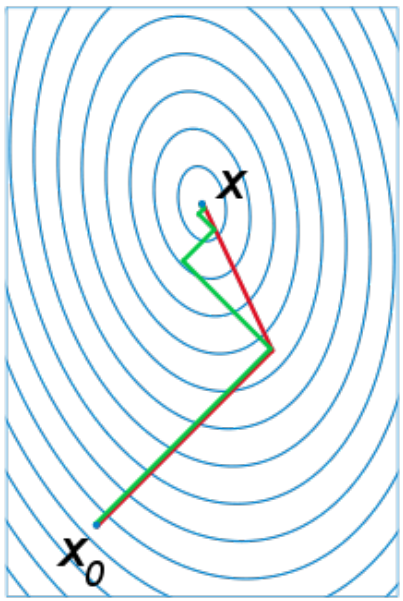


Figure 3:

- At training iteration t, the next search direction d_t takes the form:

$$d_t = \nabla_\theta J(\theta) + \beta_t d_{t-1}$$

- d_t and d_{t-1} are defined as conjugate if
  $d_t^\top H d_{t-1} = 0$
- Gradient descent with optimal step size (in green) and conjugate vector (in red) for minimizing a quadratic function. Conjugate gradient, converges in at most n steps where n is the size of the matrix of the system (here n=2).

Conjugate Gradients

Two popular methods for computing the β_t are:

- Fletcher-Reeves:

$$\beta_t = \frac{\nabla_\theta J(\theta_t)^\top \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^\top \nabla_\theta J(\theta_{t-1})} \tag{28}$$

- Polak-Ribire:

$$\beta_t = \frac{\nabla_\theta J(\theta_t) - \nabla_\theta J(\theta_{t-1})^\top \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^\top \nabla_\theta J(\theta_{t-1})} \tag{29}$$

BFGS

It is one of the most prominent quasi-Newton methods.

- BroydenFletcherGoldfarbShanno (BFGS) algorithm approximate the Newton algorithm.
- Approximate the inverse H^{-1} with a matrix M_t. M_t is iteratively refined by low-rank updates.
- Every N steps, where N is number of parameters, search is restarted in the direction of the negative gradient.
- Memory complexity: $O(n^2)$. $L - BFGS$ further reduces to $O(n)$.

Outline

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Batch Normalization

Batch normalization is one of the most interesting recent innovations in optimizing DNN by reparametrization.

- For a network with l layers, after an update $w \leftarrow w - \epsilon g$, The first-order Taylor series approximation of $\hat{y}$ predicts that the value of $\hat{y}$ will decrease by $\epsilon g^\top g$.
- But the actual update will include up to order l, the new value of $\hat{y}$ is given by

$$x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)...(w_l - \epsilon g_l)$$

  - If all weights are small, this exponentially vanishes.
  - If all weights are large, this exponentially explodes.
- Second-order optimization algorithms only mitigate this issue by computing an update that takes these second-order interactions into account.
- It requires an n-th order method to see the real effect of an update.

Batch Normalization

- Let H be a minibatch of activations for each example to normalize, with the activations for each example appearing in a row of the matrix. To normalize H, we replace it with

$$H' = \frac{H - \mu}{\sigma} \tag{30}$$

  where μ is a vector containing the mean of each unit and σ is a vector containing the standard deviation of each unit.
- This means that the gradient will never increases the standard deviation or mean of h_i.
- At training time,

$$\mu = \frac{1}{m} \sum_i H_{i,:}$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum_i (H - \mu)_i^2}$$

  where δ is a small positive value.

Coordinate Descent

If we minimize $f(x)$ with respect to a single variable x_i , then minimize it with respect to another variable x_j and so on, repeatedly cycling through all variables, we are guaranteed to arrive at a (local) minimum.

- Block coordinate descent
- Individual coordinate descent

Coordinate Descent

Coordinate descent is not a very good strategy when the value of one variable strongly influences the optimal value of another variable,

- $f(x) = (x_1 - x_2) + (x_1^2 + x_2^2)$
- The solution is to set both to zero.
- When use the coordinate descent, it's slow. the first term doesn't allow a single variable to be changed to a value that differs significantly from the current value of another value.

Polyak Averaging

- If t iterations of gradient descent visit points $\theta^{(1)}, ..., \theta^{(t)}$, the output of the Polyak Averaging is

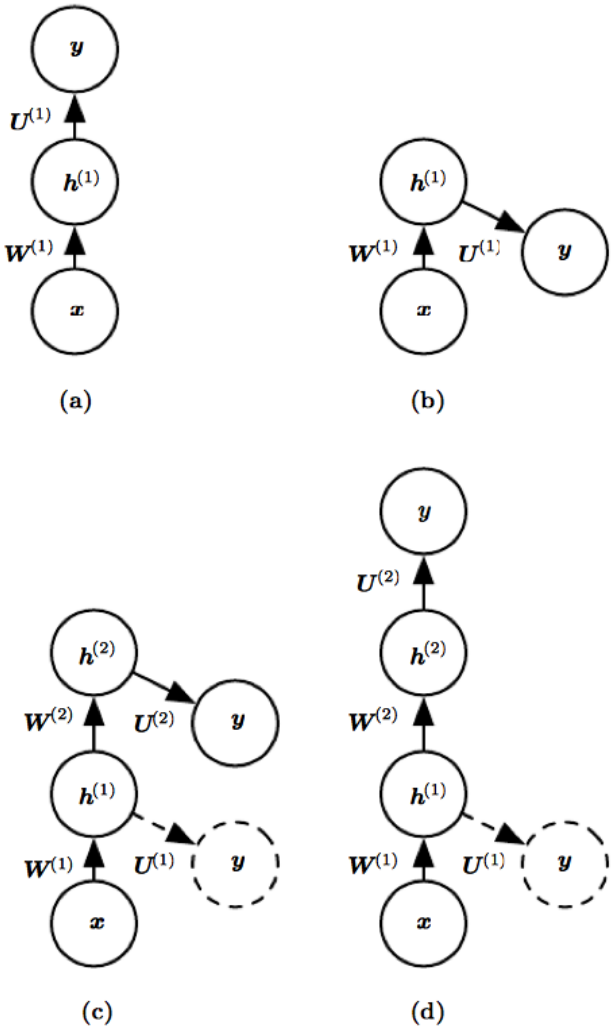
$$\hat{\theta}^{(t)} = \frac{1}{t} \sum_i \theta^{(t)} \tag{31}$$

- For non-convex problems, it is typical to use an exponentially decaying running average:

$$\hat{\theta}^{(t)} = \alpha \hat{\theta}^{(t-1)} + (1 - \alpha)\theta^{(t)} \tag{32}$$

- Basic idea: optimization algorithm may leap back and forth across a valley without visiting a point near the bottom of the valley. The average should be close to the bottom of the valley though.

Supervised Pretraining



DNNs are usually too complex to directly optimize. The idea of Pretraining:

(a) Train a shallow architecture.

(b) Another drawing of (a)

(c) Keep only the input-to-hidden layer of the original network and send the output of the first hidden layer as input to another supervised single hidden layer MLP that is trained with the same objective, thus adding a second hidden layer.

(d) A feedforward network, and jointly fine-tune all the layers.

Designing Models to Aid Optimization

It is more important to choose a model family that is easy to optimize than to use a powerful optimization algorithm.

- use linear transformations between layers.
- use activation functions that are differentiable almost everywhere, and have significant slope in large portions of domain.
  - rectified linear units and maxout units.
- Other model design strategies.
  - skip connections to reduce the length of the shortest path from lower layers' parameters to the output.

Overview

1. Preliminaries
2. Challenges in Neural Network Optimization
3. Basic Algorithms
4. Parameter Initialization Strategies
5. Algorithms with Adaptive Learning Rates
6. Approximate Second-Order Methods
7. Optimization Strategies and Meta-Algorithms

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Email: jietang@tsinghua.edu.cn