Optimization for Training Deep Models

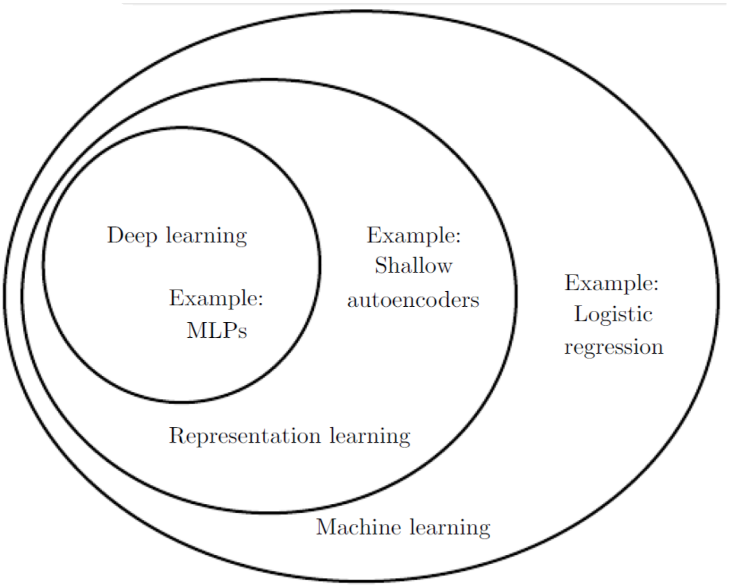
Yong Li

liyongforevercas@163.com
www.foreverlee.net

2017-03-12

Overview

Optimization in ML



Optimization in ML

Model

Hypothesis space $F = \{f | Y = f(X)\}$

Strategy

To determine the loss function $L(Y, f(x))$, like 0-1 loss function and quadratic loss function.

Algorithm

The algorithm is to select the best model in the hypothesis space by optimizing the loss function.

Empirical Risk Minimization

Cost function over the training set.

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

Empirical Risk Minimization

$$E_{x,y\sim\hat{p}_{\text{data}}}[L(f(x;\theta), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)}), \tag{2}$$

Surrogate Loss Functions and Early Stopping

Gradient-Based Optimization

Optimization

Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x

objective function

The function we want to minimize or maximize is called the objective function or criterion. We may also call it the cost function, loss function or error function in the minimization case.

Batch and Minibatch Algorithms

- General optimization algorithms: the objective function usually decomposes as a sum over the training examples.
- Batch Algorithms: compute each update based on total training examples.
- Minibatch Algorithms: compute each update based on a subset of the terms of the full cost function.

Minibatch SGD

- Larger batches provide a more accurate estimate of the gradient.
- Multicore architectures are usually underutilized by extremely small batches.
- The amount of memory scales with the batch size.
- Small batches can offer a regularizing effect. Generalization error is often best for a batch size of 1.

Minibatch SGD

- Different kinds of algorithms use different kinds of information from the minibatch in different ways.
- Some algorithms are more sensitive to sampling error than others.
- Methods based only on the gradient can handle smaller batch sizes like 100. Second-order method typically require much larger batch sizes like 10,000.

Key points of Minibatch SGD

- Samples in the minibatch should be independent.
- Two subsequent minibatches of examples should also be independent.
- Minibatch follows the gradient of the true generalization error.

Key points of Minibatch SGD

- Samples in the minibatch should be independent.
- Two subsequent minibatches of examples should also be independent.
- Minibatch follows the gradient of the true generalization error.

The generalization error can be written as a sum.

$$J^*(\theta) = \sum_x \sum_y p_{\text{data}}(x, y) L(f(x; \theta), y), \tag{3}$$

with the exact gradient.

$$g = \nabla_\theta J^*(\theta) = \sum_x \sum_y p_{\text{data}}(x, y) \nabla_\theta L(f(x; \theta), y). \tag{4}$$

We can obtain an unbiased estimator by sampling a minibatch of examples.

Challenges in Neural Network Optimization

- Ill-Conditioning.
- Local Minima.
- Plateaus, Saddle Points and Other Flat Regions.
- Cliffs and Exploding Gradients.
- Long-Term Dependencies.
- Inexact Gradients.
- Poor Correspondence between Local and Global Structure.
- Theoretical Limits of Optimization.

Ill-Conditioning

$$f(x^{(0)} - \epsilon \mathbf{g}) \approx f(\mathbf{x}^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2}\epsilon^2 \mathbf{g}^T \mathbf{H}\mathbf{g}$$

$$\frac{1}{2}\epsilon^2 \mathbf{g}^\top \mathbf{H}\mathbf{g} - \epsilon \mathbf{g}^\top \mathbf{g} \tag{5}$$

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

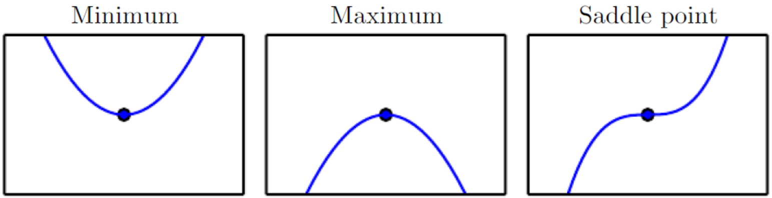
Local Minima

- Neural networks have multiple local minima because of the model identifiability problem.
- A model is said to be identifiable if a sufficiently large training set can rule out all but one setting of the models parameters.
- Weight space symmetry, m layers with n units each, then there are $(n!)^m$ ways of arranging the hidden units.
- Scaling: we can scale all of the incoming weights and biases of a unit by α if we also scale all of its outgoing weights by $\frac{1}{\alpha}$

Plateaus, Saddle Points and Other Flat Regions

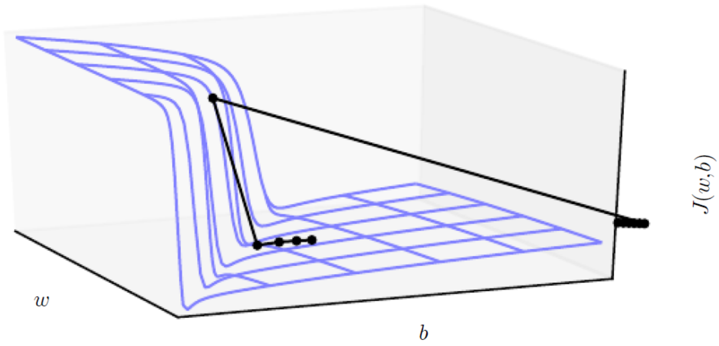
Many classes of random functions exhibit the following behavior:

- In lowdimensional spaces, local minima are common.
- In higher dimensional spaces, local minima are rare and saddle points are more common.

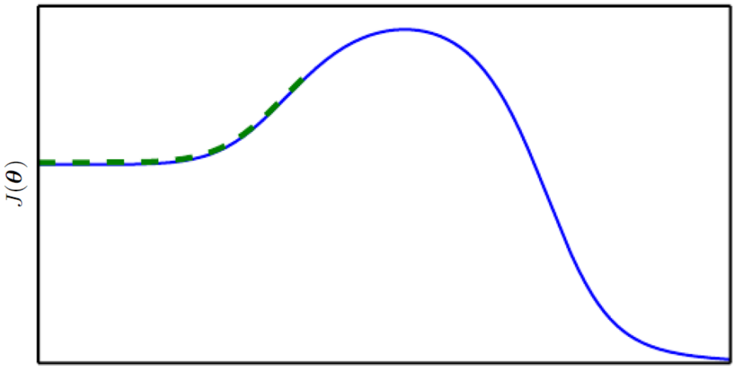


Cliffs and Exploding Gradients and Long-Term Dependencies

$$W^t = (V \text{diag}(\lambda) V^{-1})^t = V \text{diag}(\lambda)^t V^{-1}. \tag{6}$$



Poor Correspondence between Local and Global Structure



Many existing research directions are aimed at finding good initial points for problems that have difficult global structure, rather than developing algorithms that use non-local moves.

Theoretical Limits of Optimization

- Some results show that finding a solution for a network of a given size is intractable
- In practice we can find a solution easily by using a larger network for which many more parameter settings correspond to an acceptable solution.
- We usually do not care about finding the exact minimum of a function, but only in reducing its value sufficiently to obtain good generalization error.

Optimization Algorithms

- SGD, Moment SGD
- AdaGrad, RMSProp, Adam

SGD and Mini-batch SGD

Stochastic gradient descent proceeds one example at a time instead of the entire training set,

$$\theta^{k+1} = \theta^k - \eta \frac{\partial L(\theta, x^{(i)}, y^{(i)})}{\partial \theta}$$

Mini-batch SGD use more than one training example to make each estimate of the gradient,

$$\theta^{k+1} = \theta^k - \eta \frac{\partial L(\theta, x^{(i:i+n)}, y^{(i:i+n)})}{\partial \theta}$$

Convergence of SGD

A sufficient condition to guarantee convergence of SGD is that:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \tag{7}$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty. \tag{8}$$

Problem: manually adjust learning rate. When SGD is applied to a convex problem, the excess error is $O(\frac{1}{\sqrt{k}})$, while in the strongly convex case it is $O(\frac{1}{k})$.

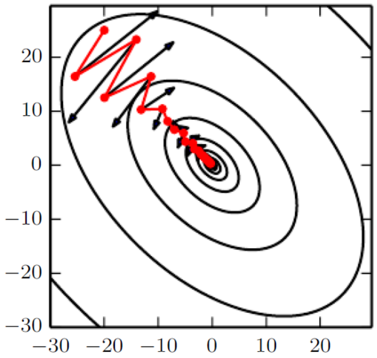
Momentum SGD

Momentum SGD adds a fraction of the update vector of the past time step to the current update vector

$$v^k = \gamma v^{k-1} + \eta \frac{\partial L(\theta, x^{(i)}, y^{(i)})}{\partial \theta}$$
$$\theta^{k+1} = \theta^k - v^k$$

Nesterov Momentum: Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$ Comparisons between SGD and momentum SGD.

Momentum SGD



If many successive gradient point in exactly the same direction, then the size of each step is

$$\frac{\epsilon \|g\|}{1-\alpha}$$

AdaGrad

AdaGrad adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters

$$\theta^{k+1} = \theta^k - \eta^k \frac{\partial L(\theta, x^{(i)}, y^{(i)})}{\partial \theta}$$

where, $\eta^k = \frac{\eta}{\sqrt{\sum_{i=1}^{k}(g^i)^2 + \epsilon}}$, and ϵ is a smoothing term. Problem: gradually shrunk to an infinitesimally small number.

AdaGrad

Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient 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 gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   (Division and square root applied
    element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

RMSprop

RMSprop uses a moving average of squared gradients instead,

$$\triangle\theta^k = -\frac{\eta}{RMS[g]_k} g_k$$

$$\theta^{k+1} = \theta^k + \triangle\theta^k$$

where $RMS[g]_k = \sqrt{E[g^2]_k + \epsilon}$, and $E[g^2]_k = \rho E[g^2]_{k-1} + (1-\rho)g_k^2$

RMSprop

Algorithm 8.5 The RMSProp algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small
  numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  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: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while
```

Adam

Adaptive Moment Estimation (ADAM)

$$m^k = \beta_1 m^{k-1} + (1-\beta_1) g^k$$

$$v^k = \beta_2 v^{k-1} + (1-\beta_2)(g^k)^2$$

$$\hat{m}^k = m^k/(1-\beta_1)$$

$$\hat{v}^k = v^k/(1-\beta_2)$$

$$\theta^{k+1} = \theta^k - \eta \frac{\hat{m}^k}{\sqrt{\hat{v}^k} + \epsilon}$$

Adam

Algorithm 8.7 The Adam algorithm

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
  $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2n )
  Initialize time step
  while stopping crit
    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

- there is currently no consensus on this point.
- The choice of which algorithm to use, at this point, seems to depend largely on the users familiarity with the algorithm. (for ease of hyperparameter tuning)

Parameter Initialization Strategies

- Some optimization algorithms are not iterative by nature and simply solve for a solution point. e.g. Linear regression.
- Other optimization algorithms are iterative by nature but, when applied to the right class of optimization problems, converge to acceptable solutions in an acceptable amount of time regardless of initialization.
- Deep learning training algorithms usually do not have either of these luxuries.
- Training algorithms for deep learning models are usually iterative in nature and thus require the user to specify some initial point.

Parameter Initialization Strategies

- The initial point can determine whether the algorithm converges at all.
- When learning does converge, the initial point can determine how quickly learning converges and whether it converges to a point with high or low cost.
- Points of comparable cost can have wildly varying generalization error, and the initial point can affect the generalization as well.

Parameter Initialization Strategies

- Modern initialization strategies are simple and heuristic.
- Designing improved initialization strategies is a difficult task.
- A further difficulty is that some initial points may be beneficial from the viewpoint of optimization but detrimental from the viewpoint of generalization.
- Our understanding of how the initial point affects generalization is especially primitive.
- Perhaps the only property known with complete certainty is that the initial parameters need to break symmetry between different units.

Parameter Initialization Strategies

- We could use Gram-Schmidt orthogonalization on an initial weight matrix.
- Random initialization from a high-entropy distribution over a high-dimensional space.
- We set the biases for each unit to heuristically chosen constants (Forget Gate, or class prior).

Parameter Initialization Strategies

- We almost always initialize all the weights in the model to values drawn randomly from a Gaussian or uniform distribution.
- The scale of the initial distribution, however, does have a large effect on both the outcome of the optimization procedure and on the ability of the network to generalize.
- Larger initial weights will yield a stronger symmetry breaking effect.
- Too large may result in exploding values.

Parameter Initialization Strategies

- One heuristic is to initialize the weights of a fully connected layer with m inputs and n outputs is $U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$
- The normalized initialization is

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

  is designed to compromise between the goal of initializing all layers to have the same activation variance and the goal of initializing all layers to have the same gradient variance.
- The formula is derived using the assumption that the network consists only of a chain of matrix multiplications, with no nonlinearities.

Conclusion

References

Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016)
Deep Learning
MIT Press Chap8. P221-265

All the figures come from the textbook.

Thanks for your attention!

Q & A

liyongforevercas@163.com
www.foreverlee.net