Numerical Optimization

Unit 6: Optimization for Training Deep Models

Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, Chap 8

Kai-Chun Chang, Meng-Hsuan Yang, Che-Rung Lee

Department of Computer Science
National Tsing Hua University

October 18, 2022

Problem formulation

- The optimization for training deep models usually find some performance measure P to the cost function $J(\theta)$.
- For example, the goal of image classification is to minimize

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

  - L is the per-example loss function;
  - θ is the set of variables.
  - $f(x;\theta)$ is the predicted output when the input is x;
  - p_{data} is the real data; distribution
  - y is the target output.
- However, the true distribution $p_{(x,y)}$ is unknown, so we replace it with the empirical distribution $\hat{p}_{(x,y)}$ defined by the training set.

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

Outline

- Basic algorithm (search direction)
  - Mini-batch, surrogate function, early stop
  - Stochastic gradient descent
  - Momentum
  - Nesterov momentum
- Adaptive learning rate (step length)
  - AdaGrad
  - RMSProp
  - Adam and its variations
- Optimization strategies and meta-algorithms
  - Batch normalization
  - Coordinate descent
  - Polyak averaging
  - Supervised pretraining
  - Designing models to aid optimization
  - Continuation methods and curriculum learning

Minibatch algorithms

- The machine learning problems usually have many training data.
  - Using more than one but fewer than all the training examples.
- Optimization algorithms that use the part of the training examples called minibatch or minibatch stochastic methods
  - It is crucial that the minibatches are selected randomly, making two subsequent gradient estimates to be independent from each other
- For example, the minibatch algorithm for computing the approximation of gradient.

Using minibatch to estimate the gradient.

sampling a minibatch of examples $x^{(1)}, ...x^{(m)}$ with corresponding targets $y^{(i)}$ from $\hat{p}_{\text{data}}$, then computing the gradient (approximation) of the loss function

$$\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$$

Surrogate loss functions and early stopping

Surrogate loss function

- Sometimes, the loss function we actually care about is not the one that can be optimized efficiently.
  - Empirical data minimization is prone to over-fitting.
  - Many loss functions, such as 0-1 loss, have no useful derivatives.
- In such situations, people typically optimize a surrogate loss function instead.

Early stopping

- Optimization using for training algorithms does not usually halt at a local minimum.
- Machine learning algorithms usually minimize a surrogate loss function but halts when a convergence criterion based on early stopping is satisfied.

Basic algorithms

- Since the size of θ is large, even $O(n)$ algorithms cannot solve the problem efficiently.
- Stochastic gradient descent (SGD) is probably the most used optimization algorithms for ML and DL.
- Taking the average gradient on a minibatch of m examples drawn i.i.d from the data-generating distribution.
- Computing the optimal step length for SGD is impossible, so we need some empirical method to decide the step length, which is called the learning rate in ML or DL.

Decrease the learning rate ϵ_k

It is common to decay the learning rate linearly until iteration τ:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

with $\alpha = \frac{k}{\tau}$. After iteration τ, it is common to leave ϵ constant

Stochastic Gradient Descent algorithm

Algorithm 8.1 Stochastic gradient descent (SGD) update

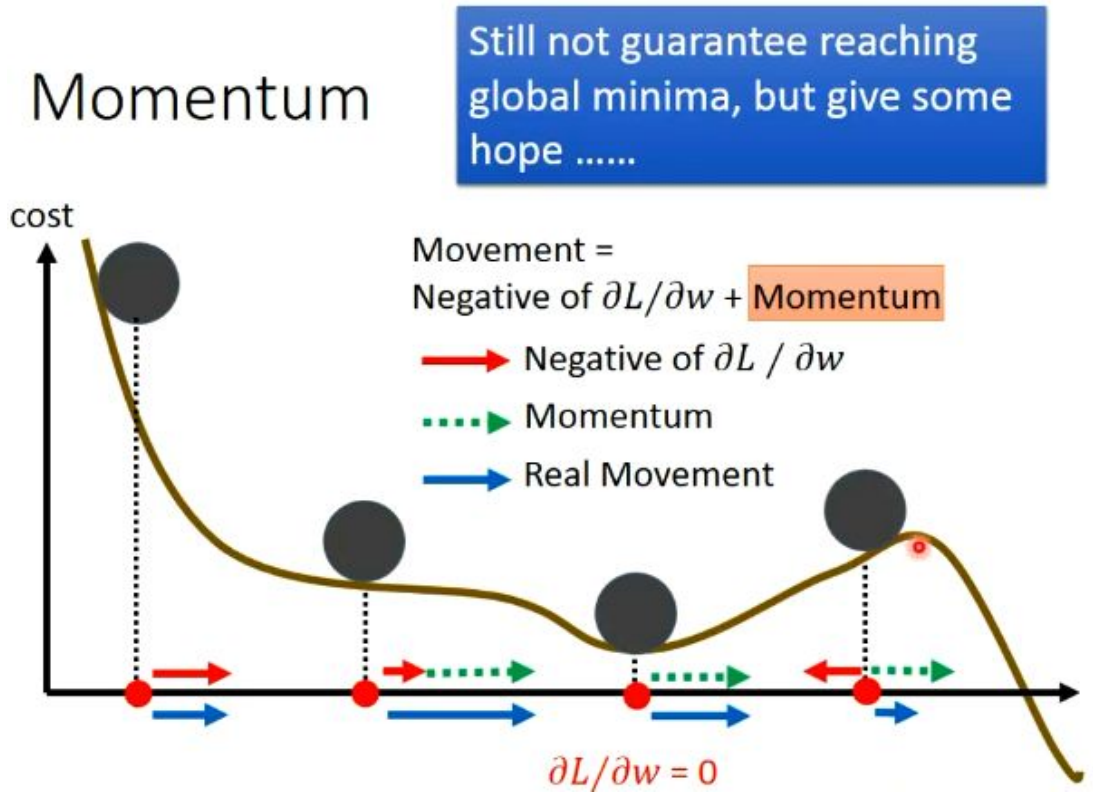
Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding 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: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

Convergence of SGD for convex problems

- When SGD is applied to a convex problem, after k iterations, the excess error, $J(\theta) - \min_\theta J(\theta)$, is $O(\frac{1}{\sqrt{k}})$.
- When J is strongly convex, which means $f - m/2\|x\|^2$ for $m > 0$ is convex, the excess error is $O(\frac{1}{k})$ after k iteration.

Momentum

- The method of momentum (Polyak, 1964) is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.
  - The name of momentum derives from a physical analogy, in which the negative gradient is a force moving a particle through parameter space, according to Newton's laws of motion.



Update rule of momentum

The velocity v accumulates the gradient elements

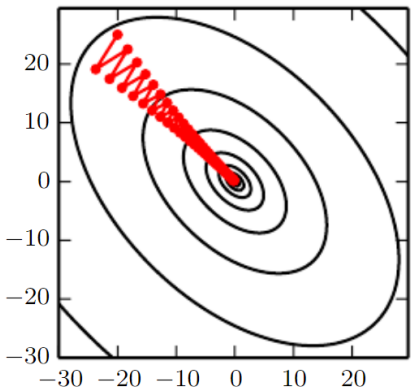
$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)} \right)$$

- The larger α is relative to ϵ, the more previous gradients affect the current direction.
- Common values of α used in practice include 0.5, 0.9, and 0.99.

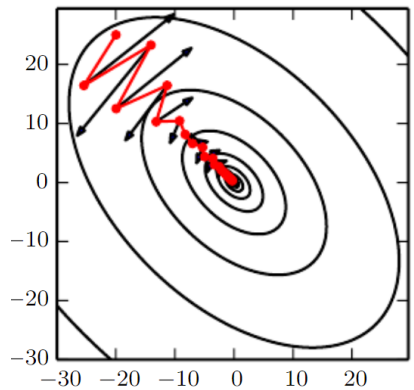
SGDwith momentum

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
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: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
end while



(a) SGD without momentum



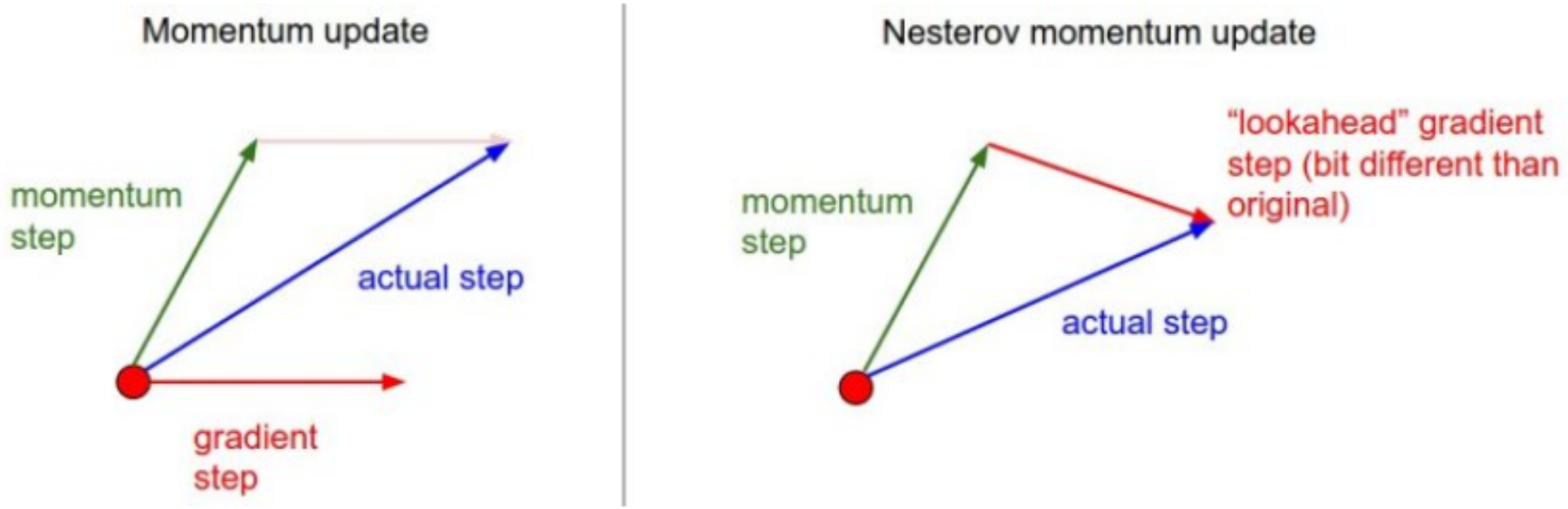
(b) SGD with momentun

Nesterov momentum

- Sutskever et al. (2013) introduced a variant of the momentum algorithm that was inspired by Nesterov's accelerated gradient method (Nesterov, 1983, 2004).
- The gradient is evaluated after the current velocity is applied.

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta + \alpha v), y^{(i)} \right)$$

- One can interpret Nesterov momentum as attempting to add a correction factor to the standard method of momentum



SGD with Nesterov momentum

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

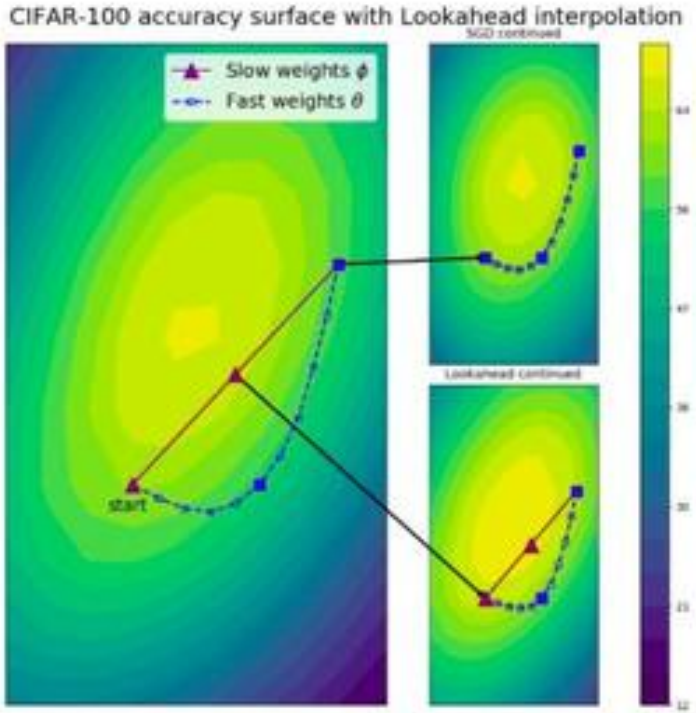
Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
end while

Convergence of SGD with Nesterov momentum

In the convex batch gradient case, Nesterov momentum brings the rate of convergence of the excess error from $O(\frac{1}{k})$ to $O(\frac{1}{k^2})$

Problem of momentum and lookahead optimizer

- Momentum direction is slow too. See `https://www.youtube.com/watch?v=TzSu14m0vqk`
- Lookahead optimizer [1] is designed to solve such problem.



Algorithm 1 Lookahead Optimizer:

Require: Initial parameters ϕ_0, objective function L
Require: Synchronization period k, slow weights step size α, optimizer A
for $t = 1, 2, \ldots$ **do**
  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
  for $i = 1, 2, \ldots, k$ **do**
    sample minibatch of data $d \sim \mathcal{D}$
    $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
  end for
  Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
end for
return parameters ϕ

Adaptive learning rate

- Model performances are significantly affected by hyperparameters, and learning rate is the most difficult to set.
- The cost function is often sensitive to some parameters. Then, can we just apply different learning rate to each parameter?

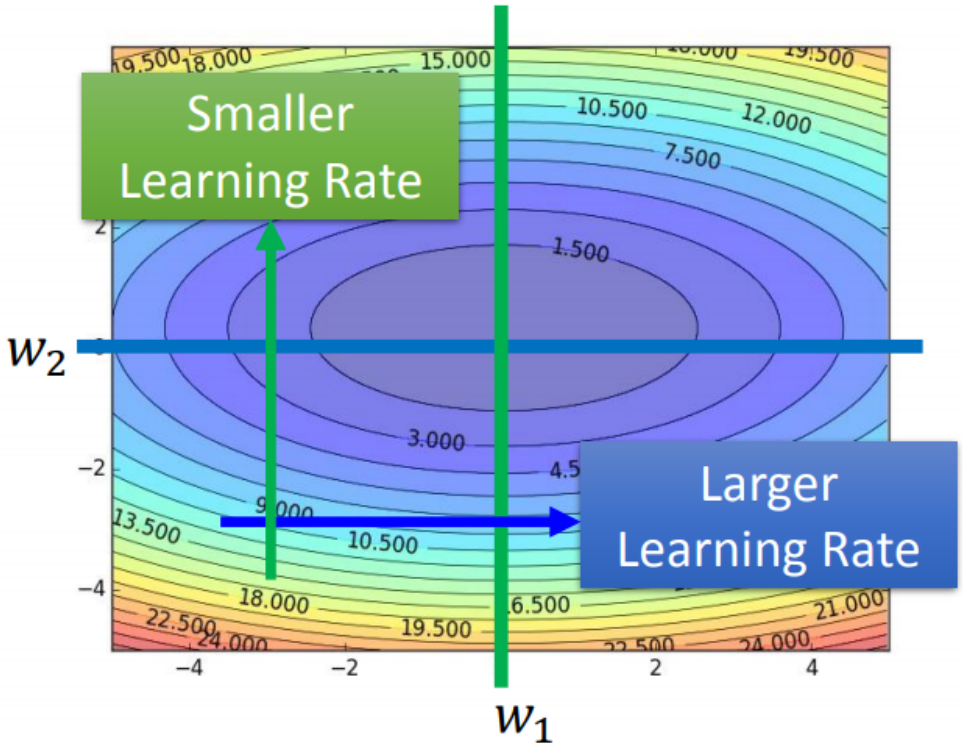


Figure: Direction in w_1 is smooth, and direction in w_2 is steep.[2]

AdaGrad

- AdaGrad was proposed by (Duchi et al., 2011)
- The operator $\odot$ means elementwise multiplication of two vectors.

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

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

AdaGrad - continued

- By accumulate g^2, the larger gradient of the parameter is, the smaller the step length it will take.
- In deep learning, the training process will be repeated over and over again which means r will become too large and take small step too early. This stops AdaGrad converging to local minimum.
- AdaGrad is designed to converge rapidly when applied to a convex function. For a non-convex function, the learning trajectory may pass through many different structures and eventually arrive at a region that is a locally convex bowl.

RMSProp

- RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl, as if it were an instance of the AdaGrad algorithm initialized within that bowl.
- ρ : the decay rate.

The RMSProp algorithm

1. Given ϵ, θ, ρ, and a small constant δ for numerical stability
2. Initialize gradient accumulation variable $r = 0$
3. Sample a minibatch $\{(x^{(i)},\ y^{(i)})\},\ i = 1, ..., m$
4. Compute gradient $g = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$
5. Update accumulation variable $r = \rho r + (1 - \rho) g^2$
6. Apply update $\theta = \theta - \frac{\epsilon}{\sqrt{r} + \delta} g$
7. Repeat Step.3 - Step.6 until stopping condition is fulfilled.

RMSProp with Nesterov momentum

- RMSProp can be combined with Nesterov momentum.
- v : the velocity, α : momentum coefficient

RMSProp with Nesterov momentum

1. Given $\epsilon, \theta, \rho, \alpha$, and a small constant δ for numerical stability
2. Initialize gradient accumulation variable $r = 0$, velocity $v = 0$
3. Sample a minibatch $\{(x^{(i)}, y^{(i)})\}, \ i = 1, ..., m$
4. Compute interim update $\hat{\theta} = \theta + \alpha v$
5. Compute gradient $g = \frac{1}{m} \nabla_{\hat{\theta}} \sum_i L(f(x^{(i)}, \hat{\theta}), y^{(i)})$
6. Update accumulation variable $r = \rho r + (1 - \rho) g^2$
7. Update velocity variable $v = \alpha v - \frac{\epsilon}{\sqrt{r} + \delta} g$
8. Apply update $\theta = \theta + v$
9. Repeat Step.3 - Step.8 until stopping condition is fulfilled.

Problem of RMSProp

- By introducing ρ, we can changing the gradient accumulation into an exponentially weighted moving average.
- This makes the earlier accumulated gradients decayed, and thus the training process can go well until converged.
- However, let us look an example. We usually let $\rho = 0.9$.
  In the first iteration : $r_1 = 0 + (1 - \rho)g_1^2 = 0.1g_1^2$
  Actually, we may expect r_1 should be g_1^2 in the first iteration, but it turns out to be $0.1g_1^2$ which is closed to zero. We say it is biased.
- We can cooperate the bias correction introduced in Adam into RMSProp.

Adam

ρ_1, ρ_2 : the exponential decay rate, usually take $\rho_1 = 0.9, \rho_2 = 0.999$

The Adam algorithm

1. Given $\epsilon, \theta, \rho_1, \rho_2$, and a small constant δ for numerical stability
2. Initialize 1st and 2nd moment variables $s = 0, r = 0$, time step $t = 0$
3. Sample a minibatch $\{(x^{(i)}, y^{(i)})\}, \ i = 1, ..., m$
4. Compute gradient $g = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$
5. $t = t + 1$
6. Update moment variables $s = \rho_1 s + (1 - \rho_1) g$, $r = \rho_2 r + (1 - \rho_2) g^2$
7. Correct bias $\hat{s} = \frac{s}{1-\rho_1^t}, \hat{r} = \frac{r}{1-\rho_2^t}$
8. Apply update $\theta = \theta - \frac{\epsilon}{\sqrt{\hat{r}} + \delta} \hat{s}$
9. Repeat Step.3 - Step.8 until stopping condition is fulfilled.

Adam and its variations

- Adam (Kingma and Ba, 2014) is just like RMSProp with momentum and bias correction, but $s = \rho_1 s + (1 - \rho_1) g$ is slightly different from RMSProp with momentum which is $s = \rho_1 s + g$.
- With bias correction $\hat{s} = \frac{s}{1-\rho_1^t}, \hat{r} = \frac{r}{1-\rho_2^t}$, training becomes more stable in early steps, it will not be too closed to zero.
- Adam usually performs well in deep learning. However, in some tasks (e.g. classification), Adam suffers from weak testing accuracy compared to SGD. Researchers still try to modify Adam to make it perform better. (e.g. RAdam[3], AdamW[4], AmsGrad[5], ...)
- Adam can cooperate with Nesterov momentum and it becomes Nadam[6].

Optimization strategies and meta-algorithms

- There are several optimization strategies that can be applied to many algorithms to improve the stability or performance.
  1. Batch normalization
  2. Coordinate descent
  3. Polyak averaging
  4. Supervised pretraining
  5. Designing models to aid optimization
  6. Continuation methods and curriculum learning

Difficulty of training very deep models

- Very deep models involve the composition of several layers. When updating variables, many functions composed together are changed simultaneously, using the updates that were computed under the assumption that the other functions remain constant.
- For example, for a non-activation model, the network becomes $\hat{y} = xw_1w_2w_3...w_l$. In the back propagation, it computes the gradient for each w. However, the gradient is only focusing on the single layer. If we adjust $w = w - \epsilon g$ simultaneously, the output become

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)(w_3 - \epsilon g_3)\cdots(w_l - \epsilon g_l), \tag{1}$$

  The second order or higher order terms, such as $\epsilon^2 g_1 g_2 \prod_{i=3}^{l} w_i$, may not be negligible if $\prod_{i=3}^{l} w_i$ is large.
- This makes it very hard to choose an appropriate learning rate, because the effects of an update to the parameters for one layer depends so strongly on all of the other layers.

Batch normalization

- Batch normalization reparameterizes every hidden layer's output H.

Batch Normalization (Ioffe and Szegedy, 2015)

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

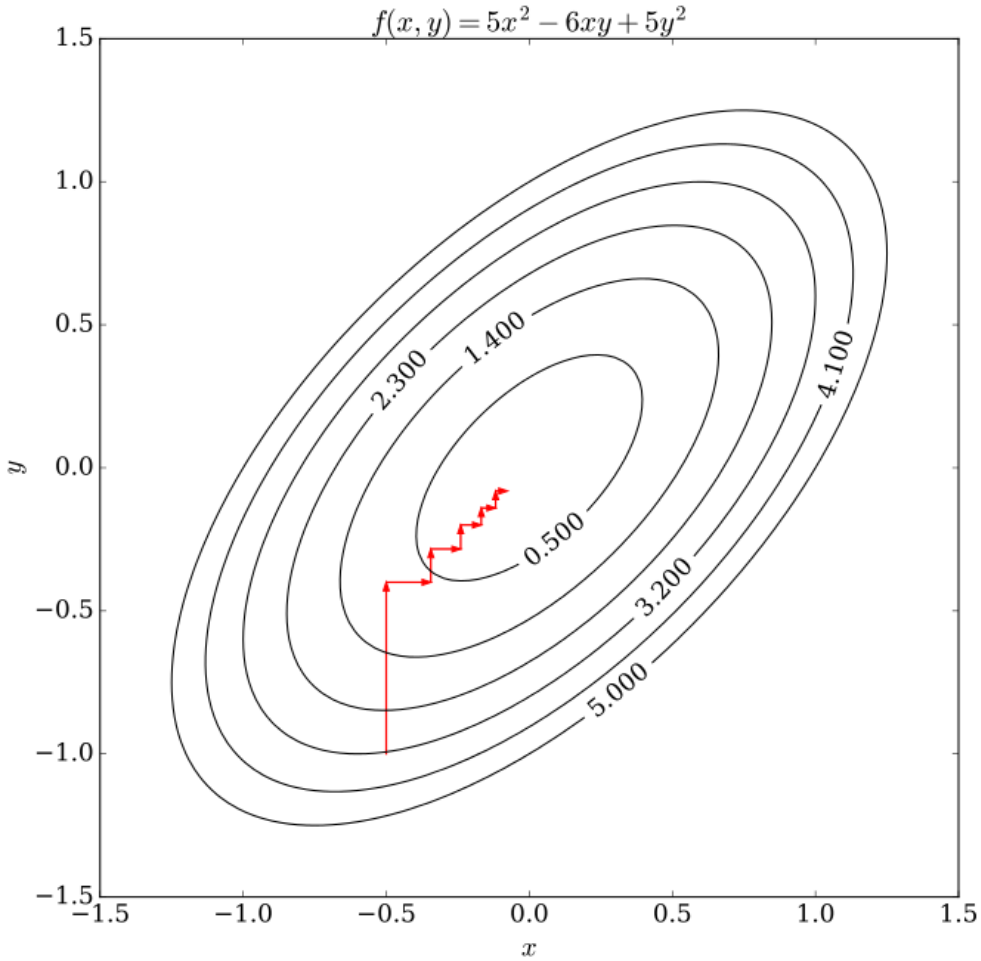
- Assume $\hat{x}_1 = BN(xw_1)$, $\hat{x}_2 = BN(\hat{x}_1 w_2)$, ..., $\hat{x}_l = BN(\hat{x}_{l-1} w_l)$. (1) becomes

$$\hat{y} = BN(\hat{x}_{l-1}(w_l - \epsilon g_l)),$$

  in which every layer's inputs are always been normalized.
- The μ, σ are computed in each minibatch. In testing time, μ, σ will be replaced by running averages that were collected during training time.

Coordinate descent

- The technique optimizes one coordinate at a time is called coordinate descent.
- For example, we can minimize $f(x)$ with respect to a variable x_0, then minimize it with respect to variable x_1, and so on, until we minimize all variables.
- Block coordinate decent is a more general method which it optimizes a subset of the variables at one time.



Polyak averaging

Polyak Averaging

- $\hat{\theta}^{(t)} = \frac{1}{t}\sum_i \theta^{(i)}$

- Polyak averaging averages several points in the trajectory of parameter space when optimizing a problem.
- Sometimes the optimization algorithm may jump back and forth across a valley, and Polyak average can prevent it from the situation.

Modified Polyak averaging

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

- In nonconvex problems, the optimization trajectory may be very complicated, and we can apply moving average for Polyak averaging.

Supervised pretraining

- Pretraining was a method training part of the model on target task and then combine them to train together.
- Recently, because of stronger computation power, transfer learning is a new approach of pretraining. Transfer learning is a technique that the weights were trained on large data set, and we use the weights as our model's initialization. Then, train the model on target tasks.

Designing models to aid optimization

- The best strategy to improve optimization is not always to pursue the best optimization algorithm. Actually, stochastic gradient descent with momentum is still used in state-of-the-art models, and what we can improve is not only the optimization algorithm but also the model architecture.
- For example, the skip connections between layers can mitigate gradient vanish problem. Furthermore, based on skip connections, residual network[7] can be better optimized by forcing the model learning residuals of the hidden layers.
- On the other hand, in GoogLeNet and deeply supervised nets, the auxiliary heads are added in the hidden layers to help optimization with additional gradients. (Auxiliary heads are classification layer which is usually in the end of a model.)

Continuation methods and curriculum learning

- The intuition is how human learns things. Teachers give simple examples when teaching a new concept, and introduce complicated examples later.
- Continuation methods constructs a series of functions $\{J^{(0)}, J^{(1)}, ..., J^{(n)}\}$ based on objective function J. $J^{(0)}$ is the simplest functions modified from J, and $J^{(n)}$ is the original objective function J. Then, let model learns from $J^{(0)}$ to $J^{(n)}$.
- Curriculum learning separates training samples into easy ones and difficult ones. In early training, increasing the influence of easier samples by having larger contribution to cost function, or sampling them more frequently. Some researches point out randomly mix easy and hard sample can improve performance.

References


M. R. Zhang, J. Lucas, G. Hinton, and J. Ba, "Lookahead optimizer: k steps forward, 1 step back," *arXiv preprint arXiv:1907.08610*, 2019.


H.-Y. Lee, "Tips for training dnn," 2017. `http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML17_2.html`.


L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.


I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.


S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.


T. Dozat, "Incorporating nesterov momentum into adam," in *4th International Conference on Learning Representations Workshop, ICLR Workshop 2016*, 2016.


K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.