Gradient-Based NN Optimization

An Overview

N. Vadivelu

University of Waterloo

December 4, 2019

Outline

Mini-Batch Gradient Descent

$$\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla_\theta L(\theta_t)$$

- η_t - learning rate. Commonly, $\eta_0 \in [1e-4, 1.0]$.
  - Needs to decay over training for convergence
  - Cyclic learning rates are common
- $L(\theta) = \frac{1}{|S|} \sum_{i \in S} f(y_i, \hat{y}_i)$, where S is a sample (or minibatch) of data
  - $L(\theta)$ is just a stochastic estimate of our true gradient
- Challenges:
  - Determining η_0 and its decay schedule
  - Escaping saddle points (main motivation)
  - Reducing $\nabla_\theta L(\theta)$'s variance
  - How do we use curvature information?

Outline

Momentum (aka Heavy-Ball)

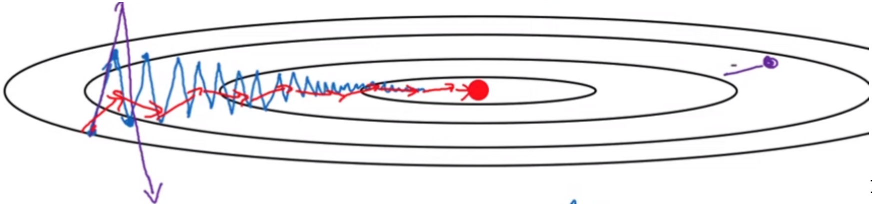
Polyak, 1964

$$v_{t+1} \leftarrow \mu v_t - \eta \nabla_\theta L(\theta_t)$$
$$\theta_{t+1} \leftarrow \theta_t + v_{t+1}$$

- μ - momentum. Commonly $\mu \in [0.8, 0.99]$
- Inspired by momentum in physics: μ is like friction, v_{t-1} is velocity, $\eta \nabla_\theta L(\theta)$ is acceleration.
  - Imagine a ball rolling down a hill–once it reaches flat ground, it will continue rolling for some time
  - Speeds up descent in directions where the gradient is consistent
  - This property allows it to escape saddle points
- Increasing μ has a similar effect to increasing η (Zhang et al. 2019)
- Polyak's original (equivalent) formulation was $\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta L(\theta) + \alpha(\theta_t - \theta_{t-1})$

More Momentum Intuition

- Click here for momentum animated in low dimensions



[1]

- blue line is normal GD, purple is increasing LR with GD, red line is momentum
- We want want a smaller learning rate on the vertical dimension and a larger one on the horizontal–momentum helps us achieve that
- We can now increase learning rate safely

[1]Image from Andrew Ng's Deep Learning Course

Momentum Pros/Cons

$$v_{t+1} \leftarrow \mu v_t - \eta \nabla_\theta L(\theta_t)$$
$$\theta_{t+1} \leftarrow \theta_t + v_{t+1}$$

- Advantages
  - Better convergence guarantees than SGD
  - Can get over saddle points
  - Decreases variance of our gradient estimator
  - Extends "perfect scaling" of learning rate (Shallue et al. 2019)
- Disadvantages
  - "Lags behind" true direction–can affect how fast it settles on a minimum
  - Adds another hyperparameter to tune

Alternate Momentum Formula

$$g_{t+1} \leftarrow \beta g_t + (1-\beta)\nabla_\theta L(\theta_t)$$
$$\theta_{t+1} \leftarrow \theta_t - \eta_t g_{t+1}$$

- We are doing an **exponential moving average (ema)**
  - Approximation to a simple moving average over $\frac{2}{1-\beta} - 1$ steps
  - Intuitively, $g_t \approx \hat{E}[\nabla_\theta L(\theta)]$, which makes our update

$$\theta_{t+1} \leftarrow \theta_t - \eta_t \hat{E}_{EMA}[\nabla_\theta L(\theta_t)]$$

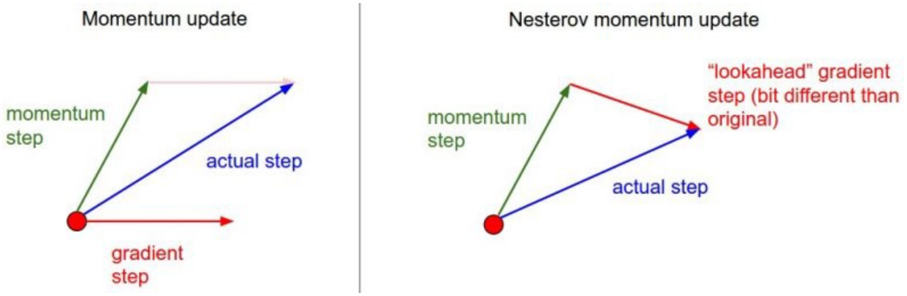
- This statement is only approximately correct, since $E[g_t] = (1-\beta^t)E[\nabla_\theta L(\theta_t)]$ (when $g_0 = 0$)
- This formulation is equivalent to the original, with $\mu = \eta_t \beta$ and learning rate $= \eta_t(1-\beta)$

Outline

Nesterov's Accelerated Gradient (NAG)

Nesterov, 1983; Sutskever et al. 2013

$$v_{t+1} \leftarrow \mu v_t - \eta \nabla_\theta L(\theta + \mu v_t)$$
$$\theta_{t+1} \leftarrow \theta_t + v_{t+1}$$



- Sutskever et al. related NAG to momentum through this formulation
- Superior convergence to momentum in theory and in practice

Outline

Motivation

- Our loss surface is known to have ill-conditioning, i.e. there are directions of very low and very high curvature
- This forces us to use a low learning rate that appeases all directions to prevent divergence
- Instead, in this section we aim scale each dimension of our parameter update seperately
- In each direction, the magnitude of the gradient tells us about the "steepness" of the curvature–we will exploit this information

AdaGrad - Rigorous Motivation

- Want to minimize **Regret**: $R(T) = \sum_{t=1}^{T} L(\theta_t) - \min_{\theta \in \Theta} \sum_{t=1}^{T} L(\theta)$
  - Gap between our model and the best model in hindsight summed
- For example, the **regret bound** for Online Composite Mirror Descent (a subgradient method) depended on $||g_t||_B^2$
  - $g_t = \nabla_\theta L(\theta_t)$, and $||x||_B^2 = \langle x, Bx \rangle$
  - Finding a better B could give us a better bound than $\mathcal{O}(\sqrt{T})$
- Duchi et al. found using $B = \sum_{t=1}^{T} g_t g_t^T =: G$ gives us a regret bound of $\mathcal{O}(\log T)$
- This makes our update rule:

$$\theta_{t+1} \leftarrow \theta_t - \eta_t G^{-1/2} \nabla_\theta L(\theta_t)$$

- Storing an inverting G is too expensive, so we use a diagonal version, whose entries can be computed as $g_t \odot g_t$ (elementwise square)

Adaptive (Sub)Gradient Method (AdaGrad)

Duchi et al. 2013

$$s_{t+1} \leftarrow s_t + \nabla_\theta L(\theta) \odot \nabla_\theta L(\theta) \qquad \text{(accumulate grad}^2\text{)}$$

$$\theta_{t+1} \leftarrow \theta_t + \frac{\eta}{\sqrt{s_{t+1} + \epsilon}} \nabla_\theta L(\theta)$$

- All operations are elementwise (i.e. elementwise division and square root of s_{t+1})
- ϵ is a small value to prevent division by 0
- Notice here we use a constant η throughout training
- In practice, our denominator grows too quickly, resulting in our algorithm "stopping" too early
- Shown to work well with sparse gradients

AdaGrad - Some Alternate Intuition

- We want to move more slowly in directions that are "steep" and more quickly in directions that are "shallow". We will use gradient magnitude in each direction as an indicator of this.
- We also want to decay our learning rate automatically. For SGD, the theoretical ideal decay is at rate $\mathcal{O}(1/t)$
  - Though this doesn't work that well in practice
- So, we scale each component of the gradient by the L2-norm of all the past gradients
- Suppose $\theta_t^{(i)}$ refers to the i^{th} component of θ_t, and $g_t^{(i)} = \frac{\partial L(\theta_t)}{\partial \theta_t^{(i)}}$:

$$\theta_{t+1}^{(i)} \leftarrow \theta_t^{(i)} - \eta \frac{g_t^{(i)}}{||all\ past\ partials||_2}$$

$$= \theta_t^{(i)} - \eta \frac{g_t^{(i)}}{\sqrt{\sum_{k=1}^{t} \left(g_k^{(i)}\right)^2}}$$

Outline

AdaDelta - Motivation

Zeiler 2012

- We don't want to accumulate gradients from the start of training, since the step size shrinks too fast
- η should be determined by the algorithm
- We want to resolve the units mismatch in AdaGrad/SGD. Consider SGD, where our update is $\Delta\theta = -\eta \nabla_\theta L(\theta_t)$:

$$\text{units of } \Delta\theta^{(i)} \propto \text{units of } \frac{\partial L(\theta)}{\partial \theta^{(i)}} \propto \frac{1}{\textit{units of } \theta^{(i)}}$$

- Update units should match parameter units, like in Newton's method:

$$\text{units of } \Delta\theta^{(i)} \propto \text{units of } \frac{\partial L(\theta)}{\partial \theta^{(i)}} \Big/ \frac{\partial^2 L(\theta)}{\partial \theta^{(i)2}} \propto \text{units of } \theta^{(i)}$$

- Since second-order methods are "correct" in this way, we rearrange the Newton's method update to isolate for our inverse second derivative:

$$\Delta\theta^{(i)} = \frac{\partial L(\theta)}{\partial \theta^{(i)}} \Big/ \frac{\partial^2 L(\theta)}{\partial \theta^{(i)2}} \implies \left(\frac{\partial^2 L(\theta)}{\partial \theta^{(i)2}}\right)^{-1} = \Delta\theta^{(i)} \Big/ \frac{\partial L(\theta)}{\partial \theta^{(i)}}$$

- Assuming curvature is locally smooth, we estimate the magnitude of $\Delta\theta^{(i)}$ by EMA.

AdaDelta

Zeiler 2012

- Let $g_t = \nabla_\theta L(\theta_t)$

$$\theta_{t+1} \leftarrow \theta_t - \sqrt{\frac{\hat{E}_{EMA}[(\Delta\theta)^2]_{t-1} + \epsilon}{\hat{E}_{EMA}[g^2]_t + \epsilon}} g_t$$

- Algorithmically, that looks like:

$$\begin{aligned}
v_t &\leftarrow \rho v_{t-1} + (1-\rho) g_t \odot g_t && \text{(update grad}^2\text{ EMA)} \\
\Delta\theta_t &\leftarrow \sqrt{\frac{d_{t-1} + \epsilon}{v_t + \epsilon}} g_t && \text{(compute update)} \\
d_t &\leftarrow \rho d_{t-1} + (1-\rho) \Delta\theta_t \odot \Delta\theta_t && \text{(update delta)} \\
\theta_{t+1} &\leftarrow \theta_t + \Delta\theta_t
\end{aligned}$$

AdaDelta - Discussion

Zeiler 2012

$$\theta_{t+1} \leftarrow \theta_t - \sqrt{\frac{\hat{E}_{EMA}[(\Delta\theta)^2]_{t-1} + \epsilon}{\hat{E}_{EMA}[g^2]_t + \epsilon}} g_t$$

- The paper defines root mean squared $RMS[x]_t = \sqrt{\hat{E}_{EMA}[x^2]_t + \epsilon}$ New Update:

$$\theta_{t+1} \leftarrow \theta_t - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

- Can we better estimate $RMS[\Delta\theta]_{t-1}$?
- We are still computing the L2-norm of a window past gradients. What if we used a different norm?

Outline

RMSProp

Hinton 2012 (unpublished)

- Inspired by RProp (Riedmiller & Braun, 1992), a full-batch neural network optimization algorithm.
  - Essentially, it gradient descent with just the sign of the gradient
- RMSProp is RProp designed for the online (minibatch) learning case

$$\theta_{t+1} \leftarrow \theta_t - \eta \frac{g_t}{\sqrt{\hat{E}_{EMA}[g_t^2] + \epsilon}}$$

Algorithmically:

$$v_t \leftarrow \beta v_{t-1} + (1-\beta) g_t \odot g_t$$
$$\theta_{t+1} \leftarrow \theta_t - \eta \frac{g_t}{\sqrt{v_t + \epsilon}}$$

- Hinton suggests $\beta = 0.9$

RMSProp Variants

Hinton 2012 (unpublished)

- Original:

$$\theta_{t+1} \leftarrow \theta_t - \eta \frac{g_t}{\sqrt{\hat{E}_{EMA}[g_t^2] + \epsilon}}$$

- Centered Second Moment:

$$\begin{aligned}\theta_{t+1} &\leftarrow \theta_t - \eta \frac{g_t}{\sqrt{\hat{E}_{EMA}[g_t^2] - \hat{E}_{EMA}[g_t]^2 + \epsilon}} \\ &= \theta_t - \eta \frac{g_t}{\sqrt{\widehat{Var}_{EMA}[g_t] + \epsilon}}\end{aligned}$$

- Intuitively, this version scales our gradient down by the noise, so we travel more slowly in directions we are less certain about
- Not clear if this version is better/worse (though it does use 2x memory)

Outline

Adaptive Moment Estimation (Adam)

Kingma & Ba 2015

$$\theta_{t+1} \leftarrow \theta_t - \eta \frac{\hat{E}_{EMA}[g_t]}{\sqrt{\hat{E}_{EMA}[g_t^2]} + \epsilon}$$

- It's just RMSProp with momentum and bias correction
  Algorithmically:

$$\begin{aligned}
g_t &\leftarrow \nabla_\theta L(\theta_{t-1}) \\
m_t &\leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t \\
\widehat{m}_t &\leftarrow m_t / (1-\beta_1^t) && \text{(bias correct 1st moment)} \\
\widehat{v}_t &\leftarrow v_t / (1-\beta_2^t) && \text{(bias correct 2nd moment)} \\
\theta_t &\leftarrow \theta_{t-1} - \eta \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}
\end{aligned}$$

- Kingma & Ba suggest $\beta_1 = 0.9$, $\beta_2 = 0.999$

Adam Explained

Kingma & Ba 2015

- When we initialize our EMA's with 0, they are biased towards 0, which Adam corrects. This is only matters early in training.
- Our update $\Delta\theta \leq \eta \min\left(1, \frac{1-\beta_1}{\sqrt{1-\beta_2}}\right)$
- Kingma & Ba claim $\frac{\widehat{m}}{\sqrt{\widehat{v}}}$ represents a *signal-to-noise ratio (SNR)*
  - This is not clear–Adam effectively does gradient-sign descent, so in a majority of cases, $\frac{\widehat{m}}{\sqrt{\widehat{v}}} = \pm 1$ (also mentioned in paper)
  - They divide by the 2nd moment, not the variance, so it isn't a true SNR
  - They claim the optimizer has a built in annealing effect, but in practice, you typically still need to decay your learning rate
  - Balles & Hennig 2018 examine Adam's SNR idea further.
- Adam is sometimes seen as approx. Natural Gradient (Amari 1998)
  - Empirically and theoretically, the empirical (diagonal) covariance matrix estimated by Adam is quite different from the (diagonal) fisher]

Adam Variations

Kingma & Ba 2015

- **AdaMax** (Kingma & Ba 2015) - A variation of Adam where the infinity norm of the past gradients is used instead of the L2-norm.
- **NAdam** (Dozat 2015) - Adam with Nesterov Momentum
- **AMSGrad** (Reddi et al. 2018) - Adam with a slight modification to improve robustness and guard against an adversarial loss case
  - $\widehat{v}_t = \max(\widehat{v}_{t-1}, v_t)$
- **RAdam** (Liu et al. 2019) - Rectified Adam
  - Simulates a warm-up heuristic automatically
  - In many applications, seems to be the current state of the art for neural network optimization

Outline

RAdam - Motivation

Liu et al. 2019

- It's common to use a "warm-up" during neural network training
  - A short period at the start of training where the learning rate is low
  - Prevents divergence while allowing a high learning rate early in training
  - Divergence is caused by intialization and high variance in the adaptive parameters in optimizers (found by Liu et al. 2019)
  - Want to make this automatic
- High variance in adaptive parameters makes optimizers prone to making "bad decisions" early in training–want to automatically avoid this
- EMA tries to efficiently simulate a simple moving average (SMA), but has higher variance
  - Recall that EMA with momentum β approximates an SMA of max. length $\frac{2}{1-\beta} - 1$

RAdam

Liu et al. 2019

Initially, $\rho_\infty \leftarrow \frac{2}{1-\beta_2} - 1$ (the max equivalent SMA length)

```
$\widehat{m}_t \leftarrow \textit{(same as Adam)}$
$\widehat{v}_t \leftarrow \textit{(same as Adam)}$
$\rho_t \leftarrow \rho_\infty - \frac{2t\beta_2^t}{1-\beta_2^t}$
if variance is tractable, i.e. $\rho_t > 4$:
    $r_t \leftarrow \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}}$
    $\theta_t \leftarrow \theta_{t-1} - \eta_t r_t \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t}}$
else:
    $\theta_t \leftarrow \theta_{t-1} - \eta_t \widehat{m}_t$
```

RAdam Explained

Liu et al. 2019

- If $\psi(.)$ is our second moment estimator, Liu et al. found the $Var[\psi(.)] \approx \frac{\rho_t}{2(\rho_t-2)(\rho_t-4)\sigma^2}$
  - Also, $Var[\sqrt{\psi(.)}]$ decreases at approximately a speed of $O\left(\frac{1}{\rho_t}\right)$
- Essentially, our rectifier term reduces the variance of our second moment estimator (especially early in training)
- For typical values of β_2, $Var[\psi(.)]$ becomes tractable within 4-6 steps
- This works pretty well in practice–it is worth using over Adam
- The paper also emphasizes tuning ϵ for Adam, as increasing it from $1e-8$ to $1e-4$ almost removed the need for warm-up

Other Optimization Methods

- **K-FAC** (Martens & Grosse 2015) - Approximate natural gradient descent for neural networks.
- **Large Batch Methods** - Typically, as we double batch size, we expect to double learning rate and halve convergence time. We aim to do so without sacrificing generalization:
  - **Layerwise Adaptive Learning Rates (LARS)** (You et. al 2017) - Succeeded in this task for AlexNet
  - **LAMB** (You et. al 2019) - Extended LARS to work on BERT
- **Shampoo** (Gupta et al. 2018) - Employs an AdaGrad style update with a non-diagonal non-low rank approximation of G.
- **SM3** (Anil et al. 2019) - Uses an AdaGrad style update while using sub-linear space.

Other Papers on Training Neural Networks

- **Measuring the Effects of Data Parallelism on Neural Network Training** (Shallue et al. 2019) - Explores the relationships between batch size and all other hyperparameters
- **Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model** (Zhang et al. 2019) - Establishes a noisy quadratic framework for the loss which accurately predicts loss behaviour for neural network.
- **Cyclical Learning Rates for Training Neural Networks** (Smith 2017)
- **A Disciplined Approach to Neural Network Hyperparameters...** (Smith 2018)
- **Super-Convergence** (Smith & Topin 2018) - This paper was rejected from ICLR due to lack of breadth of experiments, but the superconvergence results are reproducible and interesting

Thanks for listening!