Stochastic Gradient Methods for Neural Networks

Chih-Jen Lin
National Taiwan University

Last updated: May 11, 2020
Outline

1. Gradient descent
2. Mini-batch SG
3. Adaptive learning rate
4. Discussion
Outline

1. Gradient descent
2. Mini-batch SG
3. Adaptive learning rate
4. Discussion
Recall that the NN optimization problem is

$$\min_\theta f(\theta)$$

where

$$f(\theta) = \frac{1}{2C} \theta^T \theta + \frac{1}{l} \sum_{i=1}^{l} \xi(z^{L+1,i}(\theta); y^i, Z^{1,i})$$

Let’s simplify the loss part a bit

$$f(\theta) = \frac{1}{2C} \theta^T \theta + \frac{1}{l} \sum_{i=1}^{l} \xi(\theta; y^i, Z^{1,i})$$

The issue now is how to do the minimization...
Gradient Descent I

- This is one of the most used optimization method
- First-order approximation

\[ f(\theta + \Delta \theta) \approx f(\theta) + \nabla f(\theta)^T \Delta \theta \]

- Solve

\[
\min_{\Delta \theta} \quad \nabla f(\theta)^T \Delta \theta \\
\text{subject to} \quad \| \Delta \theta \| = 1 \quad (1)
\]

- If no constraint, the above sub-problem goes to \(-\infty\)
The solution of (1) is

$$\Delta \theta = - \frac{\nabla f(\theta)}{\|\nabla f(\theta)\|}$$

This is called steepest descent method

In general all we need is a descent direction

$$\nabla f(\theta)^T \Delta \theta < 0$$
Gradient Descent III

From

\[ f(\theta + \alpha \Delta \theta) = f(\theta) + \alpha \nabla f(\theta)^T \Delta \theta + \frac{1}{2} \alpha^2 \Delta \theta^T \nabla^2 f(\theta) \Delta \theta + \cdots, \]

if

\[ \nabla f(\theta)^T \Delta \theta < 0, \]

then with a small enough \( \alpha \),

\[ f(\theta + \alpha \Delta \theta) < f(\theta) \]
Line Search I

- Because we only consider an approximation

\[ f(\theta + \Delta \theta) \approx f(\theta) + \nabla f(\theta)^T \Delta \theta \]

we may not have the strict decrease of the function value

- That is,

\[ f(\theta) < f(\theta + \Delta \theta) \]

may occur

- In optimization we then need a step selection procedure
Gradient descent

Line Search II

- Exact line search

\[
\min_{\alpha} f(\theta + \alpha \Delta \theta)
\]

This is a one-dimensional optimization problem

- In practice, people use backtracking line search

- We check

\[
\alpha = 1, \beta, \beta^2, \ldots
\]

with \(\beta \in (0, 1)\) until

\[
f(\theta + \alpha \Delta \theta) < f(\theta) + \nu \nabla f(\theta)^T (\alpha \Delta \theta)
\]
Here

\[ \nu \in (0, \frac{1}{2}) \]

The convergence is well established.

For example, under some conditions, Theorem 3.2 of Nocedal and Wright (1999) has that

\[ \lim_{k \to \infty} \nabla f(\theta^k) = 0, \]

where \( k \) is the iteration index

This means we can reach a **stationary point** of a non-convex problem.
The standard back-tracking line search is simple and useful.

However, the convergence is slow for difficult problems.

Thus in many optimization applications, methods of using second-order information (e.g., quasi Newton or Newton) are preferred.

\[
f(\theta + \Delta \theta) \approx f(\theta) + \nabla f(\theta)^T \Delta \theta + \frac{1}{2} \Delta \theta^T \nabla^2 f(\theta) \Delta \theta
\]

These methods have fast final convergence.
Practical Use of Gradient Descent II

- An illustration (modified from Tsai et al. (2014))

![Graphs showing distance to optimum over time with labels: Slow final convergence and Fast final convergence.](image)
But fast final convergence may not be needed in machine learning

The reason is that an optimal solution $\theta^*$ may not lead to the best model

We will discuss such issues again later
Outline

1. Gradient descent
2. Mini-batch SG
3. Adaptive learning rate
4. Discussion
Estimation of the Gradient I

- Recall the function is
  \[ f(\theta) = \frac{1}{2C} \theta^T \theta + \frac{1}{l} \sum_{i=1}^{l} \xi(\theta; y^i, Z^{1,i}) \]

- The gradient is
  \[ \frac{\theta}{C} + \frac{1}{l} \nabla_{\theta} \sum_{i=1}^{l} \xi(\theta; y^i, Z^{1,i}) \]

- Going over all data is time consuming
Estimation of the Gradient II

What if we use a subset of data

\[ E(\nabla_\theta \xi(\theta; y, Z^1)) = \frac{1}{l} \sum_{i=1}^{l} \xi(\theta; y^i, Z^{1,i}) \]

We may just use a subset \( S \)

\[ \frac{\theta}{C} + \frac{1}{|S|} \nabla_\theta \sum_{i:i \in S} \xi(\theta; y^i, Z^{1,i}) \]
Algorithm 1

1. Given an initial learning rate $\eta$.
2. \textbf{while} do
3. Choose $S \subset \{1, \ldots, l\}$.
4. Calculate
   \[ \theta \leftarrow \theta - \eta \left( \frac{\theta}{C} + \frac{1}{|S|} \nabla_{\theta} \sum_{i:i \in S} \xi(\theta; y^{i}, Z^{1,i}) \right) \]
5. May adjust the learning rate $\eta$
6. \textbf{end while}

- It’s known that deciding a suitable learning rate is difficult.
Algorithm II

- Too small learning rate: very slow convergence
- Too large learning rate: the procedure may diverge
Stochastic Gradient “Descent” I

- In comparison with gradient descent you see that we don’t do line search
- Indeed we cannot. Without the full gradient, the sufficient decrease condition may never hold.

\[ f(\theta + \alpha \Delta \theta) < f(\theta) + \nu \nabla f(\theta)^T (\alpha \Delta \theta) \]

- Therefore, we don’t have a “descent” algorithm here
- It’s possible that

\[ f(\theta^{\text{next}}) > f(\theta) \]

- Though people frequently use “SGD,” it’s unclear if “D” is suitable in the name of this method
Momentum I

- This is a method to improve the convergence speed
- A new vector $\mathbf{v}$ and a parameter $\alpha \in [0, 1)$ are introduced

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \left( \frac{\theta}{C} + \frac{1}{|S|} \nabla_{\theta} \sum_{i:i \in S} \xi(\theta; y^i, Z^{1,i}) \right)$$

$$\theta \leftarrow \theta + \mathbf{v}$$
Essentially what we do is

\[ \theta \leftarrow \theta - \eta \text{(current sub-gradient)} \]
\[ -\alpha \eta \text{(prev. sub-gradient)} \]
\[ -\alpha^2 \eta \text{(prev. prev. sub-gradient)} \] — \cdots

There are some reasons why doing so can improve the convergence speed, though details are not discussed here.
Outline

1. Gradient descent
2. Mini-batch SG
3. Adaptive learning rate
4. Discussion
Adaptive learning rate

**AdaGrad I**

- Scaling learning rates inversely proportional to the square root of sum of past gradient squares (Duchi et al., 2011)

- Update rule:

\[
g \leftarrow \frac{\theta}{C} + \frac{1}{|S|} \nabla \theta \sum_{i:i \in S} \xi(\theta; y^i, Z_{1,i})
\]

\[
r \leftarrow r + g \odot g
\]

\[
\theta \leftarrow \theta - \frac{\epsilon}{\sqrt{r} + \delta} \odot g
\]

- \(r\): sum of past gradient squares
AdaGrad II

$\epsilon$ and $\delta$ are given constants

- $\odot$: Hadamard product (element-wise product of two vectors/matrices)

- A large $g$ component
  - $\Rightarrow$ a larger $r$ component
  - $\Rightarrow$ fast decrease of the learning rate

Conceptual explanation from Duchi et al. (2011):

- frequently occurring features $\Rightarrow$ low learning rates
- infrequent features $\Rightarrow$ high learning rates
“the intuition is that each time an infrequent feature is seen, the learner should take notice.”

- But how is this explanation related to $g$ components?
- Let’s consider linear classification. Recall our optimization problem is

$$\frac{\mathbf{w}^T \mathbf{w}}{2} + C \sum_{i=1}^{l} \xi(\mathbf{w}; y_i, x_i)$$
For methods such as SVM or logistic regression, the loss function can be written as a function of $\mathbf{w}^\top \mathbf{x}$

$$\xi(\mathbf{w}; y, \mathbf{x}) = \hat{\epsilon}(\mathbf{w}^\top \mathbf{x})$$

Then the gradient is

$$\mathbf{w} + C \sum_{i=1}^{l} \hat{\epsilon}'(\mathbf{w}^\top \mathbf{x}_i) \mathbf{x}_i$$

Thus the gradient is related to the density of features
AdaGrad V

- The above analysis is for linear classification
- But now we have a non-convex neural network!
- Empirically, people find that the sum of squared gradient since the beginning causes too fast decrease of the learning rate
RMSProp I

- Idea: they think AdaGrad’s learning rate may be too small before reaching a locally convex region
- That is, OK to sum all past gradient squares in convex, but not non-convex
- Thus they do “exponentially weighted moving average”
RMSProp II

- Update rule

\[
\begin{align*}
    r & \leftarrow \rho r + (1 - \rho) g \odot g \\
    \theta & \leftarrow \theta - \frac{\epsilon}{\sqrt{\delta + r}} \odot g
\end{align*}
\]

- AdaGrad:

\[
\begin{align*}
    r & \leftarrow r + g \odot g \\
    \theta & \leftarrow \theta - \frac{\epsilon}{\sqrt{r + \delta}} \odot g
\end{align*}
\]
Somehow the setting is a bit heuristic and the reason behind the change (from AdaGrad to RMSProp) is not really that strong.
ADAM (Adaptive Moments) I

The update rule (Kingma and Ba, 2015)

\[ g \leftarrow \frac{\theta}{C} + \frac{1}{|S|} \nabla \theta \sum_{i:i \in S} \xi(\theta; y^i, Z^{1,i}) \]

\[ s \leftarrow \rho_1 s + (1 - \rho_1) g \]

\[ r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g \]

\[ \hat{s} \leftarrow \frac{s}{1 - \rho_1^t} \]

\[ \hat{r} \leftarrow \frac{r}{1 - \rho_2^t} \]

\[ \theta \leftarrow \theta - \frac{\epsilon}{\sqrt{\hat{r}} + \delta} \odot \hat{s} \]
ADAM (Adaptive Moments) II

- $t$ is the current iteration index
- Roughly speaking, ADAM is the combination of
  - Momentum
  - RMSprop
- From Goodfellow et al. (2016),

\[
\frac{\epsilon}{\sqrt{\hat{r}} + \delta} \odot \hat{s}
\]

(i.e., the use of momentum combined with rescaling) “does not have a clear theoretical motivation”
The two steps

\[
\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}
\]

\[
\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}
\]

are called “bias correction”

Why “bias correction”?
They hope that

$$E[s_t] = E[g_t]$$

and

$$E[r_t] = E[g_t \odot g_t],$$

where $t$ is the iteration index.
For $s_t$, we have

$$s_t = \rho_1 s_{t-1} + (1 - \rho_1) g_t$$
$$= \rho_1 (\rho_1 s_{t-2} + (1 - \rho_1) g_{t-1}) + (1 - \rho_1) g_t$$
$$= (1 - \rho_1) \sum_{i=1}^t \rho_1^{t-i} g_i$$

We assume that $s$ is initialized as 0
ADAM (Adaptive Moments) VI

Then

\[ E[s_t] = E[(1 - \rho_1) \sum_{i=1}^{t} \rho_1^{t-i} g_i] \]

\[ = E[g_t](1 - \rho_1) \sum_{i=1}^{t} \rho_1^{t-i} \]

Note that we assume

\[ E[g_i], \forall i \geq 1 \]

are the same
Next, 

\[ (1 - \rho_1) \sum_{i=1}^{t} \rho_1^{t-i} \]

\[ = (1 - \rho_1)(1 + \cdots + \rho_1^{t-1}) \]

\[ = 1 - \rho_1^t \]

Thus 

\[ E[s_t] = E[g_t](1 - \rho_1^t) \]

and they do 

\[ \hat{s} \leftarrow \frac{s}{1 - \rho_1^t} \]
The above derivation on bias correction partially follows from https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization

The situation for $E[g_t \odot g_t]$ is similar

How about ADAM’s practical performance?

From Goodfellow et al. (2016), “generally regarded as being fairly robust to the choice of hyperparameters, though the learning rate may need to be changed from the default”
However, from the web page we referred to for deriving the bias correction, “The original paper ... showing huge performance gains in terms of speed of training. However, after a while people started noticing, that in some cases Adam actually finds worse solution than stochastic gradient”

One example of showing the above is Wilson et al. (2017)

We may do some experiments later
Outline

1. Gradient descent
2. Mini-batch SG
3. Adaptive learning rate
4. Discussion
Choosing Stochastic Gradient Algorithms

- From Goodfellow et al. (2016), “there is currently no consensus”
- Further, “the choice ... seemed to depend on the user’s familiarity with the algorithm”
- This isn’t very good. Can we have some systematic investigation?
The special property of data classification is essential

$$E(\nabla_{\theta} \xi(\theta; y, Z^1)) = \frac{1}{\ell} \nabla_{\theta} \sum_{i=1}^{\ell} \xi(\theta; y^i, Z^{1,i})$$

Indeed stochastic gradient is less used outside machine learning

- Easy implementation. It’s simpler than methods using, for example, second derivative
- Non-convexity plays a role
Why Stochastic Gradient Widely Used? II

- For convex, other methods are efficient to find the global minimum
- But for non-convex, efficiency to reach a stationary point is less useful
- A global minimum usually gives a good model (loss minimized), but for a stationary point we are less sure

- All these explain why SG is popular for deep learning
- What are your opinions? Any other reasons you can think of
We have shown several variants
Don’t you think some settings are a bit ad hoc?
There are reasons behind each change. But some are just heuristic
Can we try a paradigm completely different?
But before that we need some first-hand experiences and know implementation details.


