

Stochastic Gradient Methods for Neural Networks

Chih-Jen Lin
National Taiwan University

Last updated: August 3, 2025

Estimation of the Gradient I

- Recall the function is

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

- The gradient is

$$\begin{aligned} & \frac{\boldsymbol{\theta}}{C} + \frac{1}{l} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^l \xi(\boldsymbol{\theta}; \mathbf{y}^i, Z^{1,i}) \\ &= \frac{\boldsymbol{\theta}}{C} + \frac{1}{l} \sum_{i=1}^l \nabla_{\boldsymbol{\theta}} \xi(\boldsymbol{\theta}; \mathbf{y}^i, Z^{1,i}) \end{aligned}$$

Estimation of the Gradient II

- Going over all data is time consuming
- If data are from the same distribution

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

then we may just use a **subset** S (often called a batch)

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

Stochastic Gradient Algorithm I

- 1: Given an initial learning rate η .
- 2: **while do**
- 3: Choose $S \subset \{1, \dots, l\}$.
- 4: Calculate

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

- 5: May adjust the learning rate η
 - 6: **end while**
- It's known that deciding a suitable learning rate is difficult

Stochastic Gradient 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.

$$f(\boldsymbol{\theta} + \alpha \Delta \boldsymbol{\theta}) < f(\boldsymbol{\theta}) + \nu \nabla f(\boldsymbol{\theta})^T (\alpha \Delta \boldsymbol{\theta})$$

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

$$f(\boldsymbol{\theta}^{\text{next}}) > f(\boldsymbol{\theta})$$

- 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 of the stochastic gradient method
- A new vector $\mathbf{v}$ and a parameter $\alpha \in [0, 1)$ are introduced

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$$

Momentum II

- Essentially what we do is

$$\begin{aligned}\theta \leftarrow & \theta - \eta(\text{current sub-gradient}) \\ & - \alpha\eta(\text{prev. sub-gradient}) \\ & - \alpha^2\eta(\text{prev. prev. sub-gradient}) - \dots\end{aligned}$$

- Thus we are using an **exponentially weighted average** of sub-gradients
- Because at each iteration we select a subset of data to get an **approximate** gradient, the resulting directions may be noisy

Momentum III

- Conceptually, a moving average may make the direction closer to the true one, so we get faster convergence
- However, the rule in (1) may be biased toward the initial value
- Thus we need **bias correction**
- This will be discussed later

AdaGrad I

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

$$\begin{aligned}\mathbf{g} &\leftarrow \frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \mathbf{y}^i, Z^{1,i}) \\ \mathbf{r} &\leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\mathbf{r}} + \delta} \odot \mathbf{g}\end{aligned}$$

- $\mathbf{r}$: sum of past gradient squares

AdaGrad II

ϵ and δ are given constants

- $\odot$: Hadamard product (element-wise product of two vectors/matrices)
- A large $\mathbf{g}$ component
 $\Rightarrow$ a larger $\mathbf{r}$ 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

AdaGrad III

“the intuition is that each time an infrequent feature is seen, the learner should **take notice**.”

- But how is this explanation related to $\mathbf{g}$ components?
- Let's consider **linear** classification. Recall our optimization problem is

$$\frac{\mathbf{w}^T \mathbf{w}}{2} + C \sum_{i=1}^l \xi(\mathbf{w}; y_i, \mathbf{x}_i)$$

AdaGrad IV

- For methods such as SVM or logistic regression, the loss function can be written as a function of $\mathbf{w}^T \mathbf{x}$

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

Then the gradient is

$$\mathbf{w} + C \sum_{i=1}^I \hat{\epsilon}'(\mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i$$

- 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**

RMSPProp I

- The original reference seems to be the lecture slides at https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- 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**”

RMSPProp II

- Update rule

$$\begin{aligned} \mathbf{r} &\leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\delta + \mathbf{r}}} \odot \mathbf{g} \end{aligned}$$

- AdaGrad:

$$\begin{aligned} \mathbf{r} &\leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\mathbf{r} + \delta}} \odot \mathbf{g} \end{aligned}$$

RMSPProp III

- Somehow the setting is a bit heuristic and the reason behind the change (from AdaGrad to RMSPProp) is not really that strong

References I

- J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.