4. Optimization Methods

Dong-Gyu, Lee

Dept. of Statistics, KU

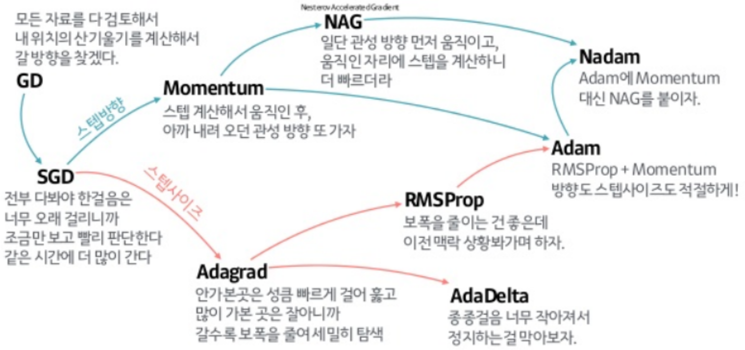
2020, Feb 18

Contents

Today's Goal

- In this time, we will focus on how to find the minimum value of the loss function through various algorithms.
- And we will look at the characteristics of each optimizer through a formula and a picture.

- The key points of our algorithms today are:



출처: 하용호, 자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다

Figure 1: Summary of Algorithms

Gradient Descent(GD)

- The formula below is the basic equation to use when finding the minimum value of the loss function.

$$\omega_{t+1} \leftarrow \omega_t - \eta \frac{\partial L(\omega_t)}{\partial \omega_t}$$

$$\omega_t : \textit{Parameters}, \quad L(\omega_t) : \textit{Loss Function}, \quad \eta : \textit{Learning Rate}$$

- This concept can also be felt through the Taylor approximation.

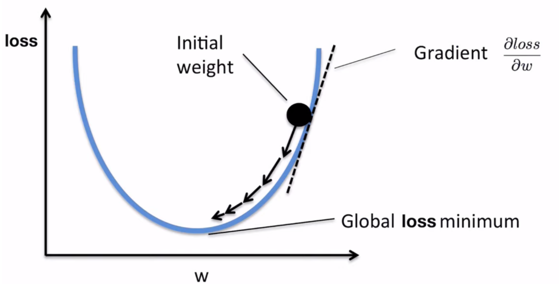
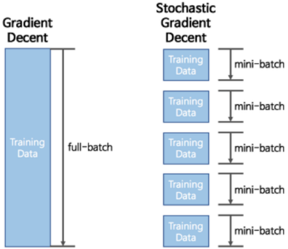


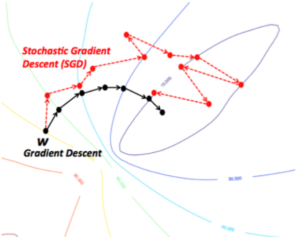
Figure 2: Gradient Descent

SGD

- Stochastic Gradient Descent(SGD)[6] is a way of updating parameters for every single piece of data.
- We will skip the BGD and MB SGD shown in the figure below because we learned about chapter 3.



(a) BGD & Mini Batch SGD



(b) Optimization Route

Figure 3: Comparison between BGD and MB SGD

Gradient Basis: Momentum

- By changing GD's differential term $\frac{\partial L(\omega)}{\partial \omega}$, the Momentum algorithm[7] compensates slightly for the shortcoming of falling into the local minimum, as if inertia worked.
- The algorithm formula is :

$$\nu_{t+1} \leftarrow \gamma \nu_t + \eta \frac{\partial L(\omega_t)}{\partial \omega_t}$$

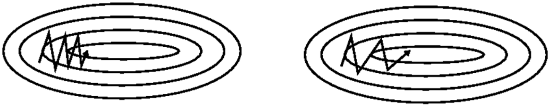
$$\omega_{t+1} \leftarrow \omega_t - \nu_{t+1}$$

$$\omega_t : Parameters, \quad L(\omega_t) : Loss\ Function, \quad \eta : Learning\ Rate,$$

$$\gamma : Momentum\ (HyperParameter) \quad \nu_t : Moment\ Parameter$$

Gradient Basis: Momentum

- Momentum is more effective when oscillating.



(a) SGD　　(b) Momentum

Figure 4: Comparison between SGD and Momentum

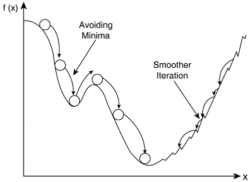


Figure 5: Expected Movement in Momentum

Gradient Basis: NAG

- The momentum algorithm calculates the gradient by substracting the gradient of current point to update the inertia parameter.
- Nesterov Accelerated Gradient(NAG)[4], on the other hand, updates the inertia parameter by subtracting the gradient of the expected destination.

$$\nu_{t+1} \leftarrow \gamma\nu_t + \eta\frac{\partial L(\omega_t - \gamma\nu_t)}{\partial\omega_t}$$

$$\omega_{t+1} \leftarrow \omega_t - \nu_{t+1}$$

$$\omega_t : \textit{Parameters}, \quad L(\omega_t) : \textit{Loss Function}, \quad \eta : \textit{Learning Rate},$$
$$\gamma : \textit{Momentum (HyperParameter)} \quad \nu_t : \textit{Moment Parameter}$$

Gradient Basis: NAG

- NAG can move more effectively than Momentum method.
- In the case of the Momentum method, there is a disadvantage that it can go much further by inertia even when it needs to stop.
- But in the case of the NAG method, parameters change the movement path after moving properly in the direction of momentum.

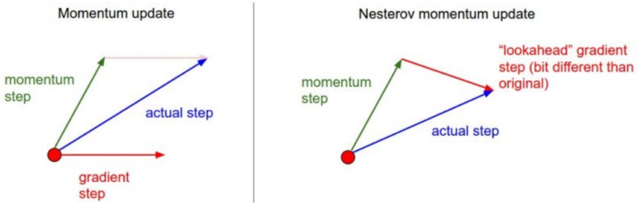
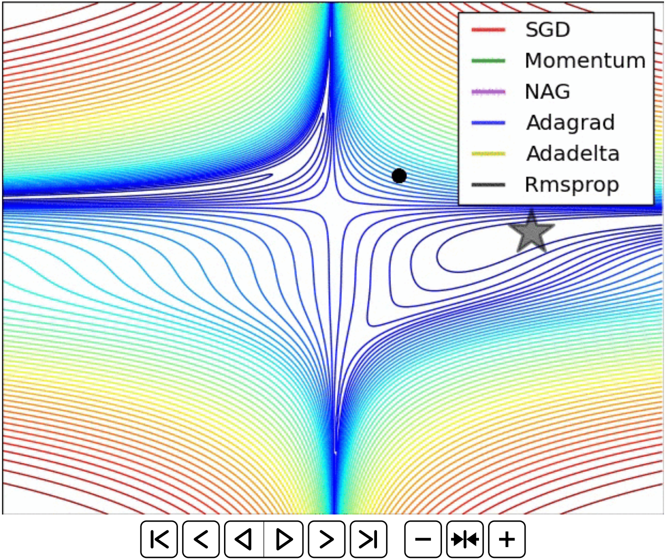


Figure 6: Comparison between Momentum and NAG

Interim Summary



Learning Rate Basis: AdaGrad

- Adaptive Gradient(AdaGrad)[2] is an algorithm associated with GD's learning rate term, η.
- In other words, the update is performed by giving different learning rates for each parameter.
- At this time gradient change and parameter change are inversely proportional.

$$g_{t+1} \leftarrow g_t + \frac{\partial L(\omega_t)}{\partial \omega_t} \odot \frac{\partial L(\omega_t)}{\partial \omega_t}$$
$$\omega_{t+1} \leftarrow \omega_t - \frac{\eta}{\sqrt{g_{t+1} + \epsilon}} \frac{\partial L(\omega_t)}{\partial \omega_t}$$

$$\omega_t : Parameters, \quad L(\omega_t) : Loss\ Function, \quad \eta : Learning\ Rate,$$
$$\epsilon : Stabilization\ Parameters \quad \odot : Hadamard\ Product$$
$$\underline{g_t : Leraning\ Parameter(Gradient\ Sqaured)}$$

Learning Rate Basis: AdaGrad

- But AdaGrad's final learning rate is a structure that adds up by squaring gradients.
- This is a fatal flaw leading to slow learning.

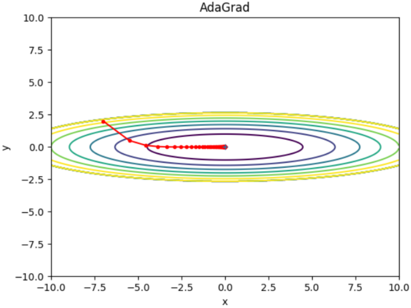


Figure 7: AdaGrad

Learning Rate Basis: RMSProp

- It is almost identical to adadelta, which is introduced next slide, but it is introduced here for easy understanding of the concept.
- Root Mean Square Propagation(RMSProp)[9] is a concept that prof. Hinton used to describe adadelta in Coursera.

$$
\begin{gathered}
g_t^2 \leftarrow \frac{\partial L(\omega_t)}{\partial \omega_t} \odot \frac{\partial L(\omega_t)}{\partial \omega_t} \\
E[g^2]_t \leftarrow \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \\
\omega_{t+1} \leftarrow \omega_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \frac{\partial L(\omega_t)}{\partial \omega_t}
\end{gathered}
$$

$$
\begin{gathered}
\omega_t : \textit{Parameters}, \quad L(\omega_t) : \textit{Loss Function}, \quad \eta : \textit{Learning Rate}, \\
\epsilon : \textit{Stabilization Parameters} \quad \underline{g_t : \textit{Gradient}} \\
\odot : \textit{Hadamard Product}, \quad \gamma : \textit{Hyper Parameter}
\end{gathered}
$$

Learning Rate Basis: RMSProp

- While a gradient squared are just added in AdaGrad, RMSProp takes the form of exponential moving averages.

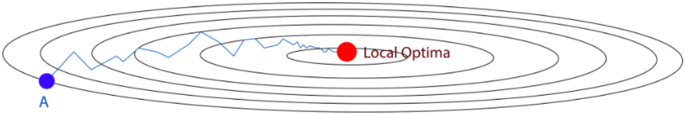


Figure 8: RMSProp

Learning Rate Basis: Adadelta

- Adadelta[10] is an algorithm designed to avoid a situation where the learning rate in adagrad continues to decrease and eventually converges to zero.
- The characteristics of Adadelta are that the exponential moving average is in both the numerator and denominator, and the learning rate is not exist in the algorithm.

Learning Rate Basis: Adadelta

- Assuming that γ, ϵ, and ω_1 are known, the algorithm is as follows :

  1. Initialize $E[g^2]_0 = 0,\ E[\Delta\omega^2]_0 = 0$
  2. For $t = 1 : T$, Do
  3. $\quad g_t^2 \leftarrow \frac{\partial L(\omega_t)}{\partial \omega_t} \odot \frac{\partial L(\omega_t)}{\partial \omega_t}$
  4. $\quad E[g^2]_t \leftarrow \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$
  5. $\quad \Delta\omega_t (= \omega_{t+1} - \omega_t) \leftarrow -\frac{\sqrt{E[\Delta\omega^2]_{t-1} + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} \frac{\partial L(\omega_t)}{\partial \omega_t}$
  6. $\quad E[\Delta\omega^2]_t \leftarrow \gamma E[\Delta\omega^2]_{t-1} + (1-\gamma)\Delta\omega_t^2$
  7. $\quad \omega_{t+1} \leftarrow \omega_t + \Delta\omega_t$

$$\omega_t : \textit{Parameters}, \quad L(\omega_t) : \textit{Loss Function} \quad \underline{g_t : \textit{Gradient}}$$

$$\epsilon : \textit{Stabilization Parameters}, \quad \odot : \textit{Hadamard Product}$$

$$\gamma : \textit{Hyper Parameter}$$

- In step 5, the gradient coefficient of Adadelta is calculated by **taking a suitable approximation instead of calculating the Hessian. And this is mathematically valid.**

Combined: Adam

- Adaptive Moment Estimation(Adam) algorithm[3], which is widely used today, is important because it greatly reduces the risk of falling into the local minima or staying saddle point.
- Adam is an algorithm that combines the AdaGrad and Momentum methods.
- All the "Adaptive" methodologies we have seen so far have the effect of maximizing any information about slight parameter change.

Combined: Adam

- Assuming that $(\beta_1,\ \beta_2) \in [0,1)$, ϵ, and ω_1 are known, the algorithm is as follows :

1. Initialize $m_0 = 0,\ v_0 = 0$
2. For $t = 1 : T$, Do
3. $\quad g_t^2 \leftarrow \frac{\partial L(\omega_t)}{\partial \omega_t} \odot \frac{\partial L(\omega_t)}{\partial \omega_t}$
4. $\quad m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
5. $\quad v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
6. $\quad \omega_{t+1} \leftarrow \omega_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} m_t$

ω_t : *Parameters*, $\quad L(\omega_t)$: *Loss Function* $\quad \underline{g_t \text{ : } \textit{Gradient}}$

ϵ : *Stabilization Parameters*, $\quad \odot$: *Hadamard Product*

$\beta_1,\ \beta_2$: *Hyper Parameter*, $\quad m_t$: *Momentum*

v_t : *Decay Parameter*

Combined: Adam

- In Keras, the initial values of the adam parameter are as follows: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ *None*
- Remember to set the initial value of the epsilon parameter.

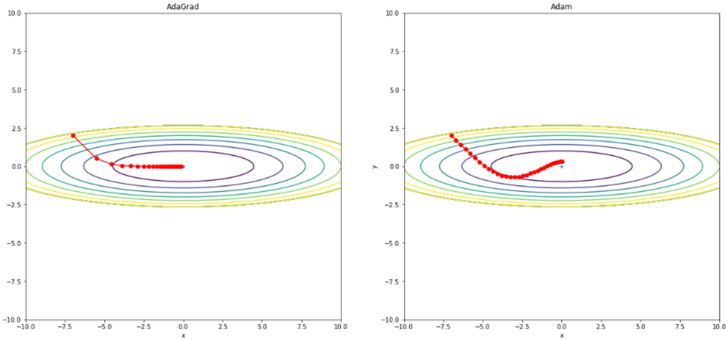
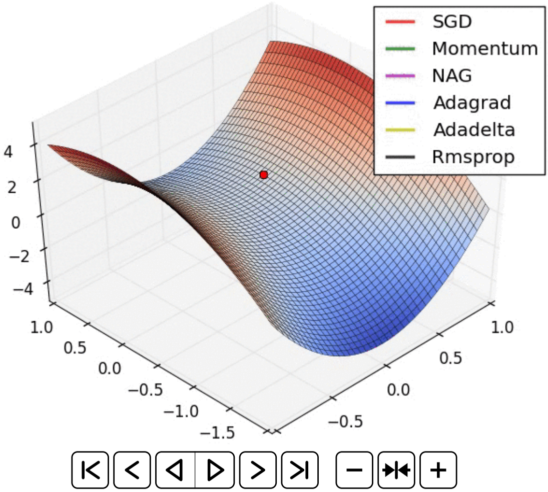


Figure 9: Adam

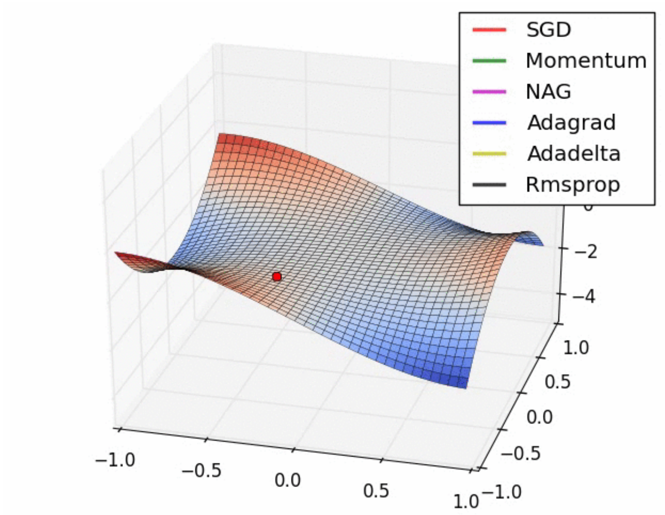
Combined: Adam

- In Adam, the β_1 and ϵ parameters are especially important.
- Empirically, when solving the regression problem, it is known that the ϵ is good at 10^{-2} or 10^{-4} values.
- And in the case of classification, ϵ may use 10^{-8}.
- If I want to feel the effect of inertia, then use adam, otherwise RMSProp is fine.

- The .gif below has some problems with Momentum and NAG.



Summary



SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

Besides..

- Another optimization algorithm, there are Adamax[3], NAdam[1], AMSGrad[5] and so on.
- In summary :
  1. Adamax : Replace with the expression $v_t = \beta_2^\infty v_{t-1} + (1-\beta_2^\infty)|g_t|^\infty$ in Adam alogrithm step 5. $(= max(\beta_2 \cdot v_{t-1}, |g_t|))$
  2. NAdam : NAG + Adam
  3. AMSGrad : Use the maximum of past squared gradients v_t rather than the exponential average to update the parameters.

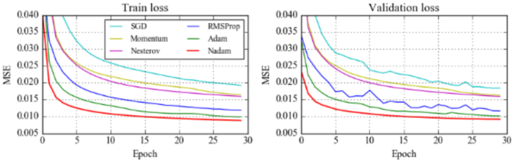


Figure 10: MSE Change for Each Epoch

Besides..

- The overall algorithm is summarized as follows :

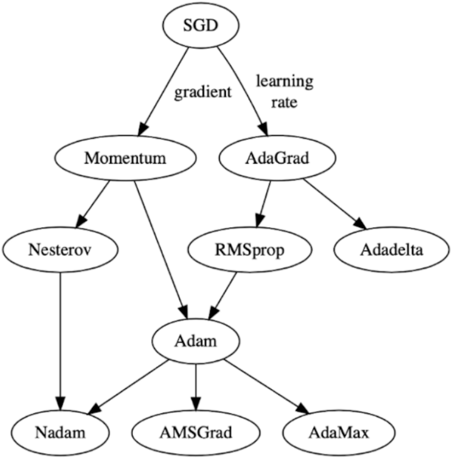


Figure 11: Total Summary

Additional Strategies for Optimization

1. Training data should "be mixed" at every epoch.
2. "Curriculum Learning" that learns easy data first and then difficult data is important.
3. First of all, "Batch Normalization(BN)" is the most important
4. For faster learning, "applying noise to the gradient" is also useful.
5. Be careful to have proper "Early Stopping" and "learning rate".
6. After the appropriate epoch, you can artificially slow down the learning speed through the "Learning Decay".
7. There is also a "Cyclical Learning Rate"[8] that gives a period to the value of learning rate.

- The application of the Cyclical Learning Rate is as follows :

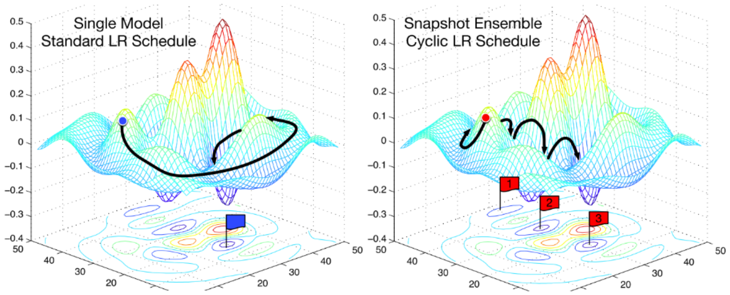


Figure 12: Snapshot Ensemble with CLR

Next

- Next time, we'll deal with the followings:
  1. Initial value problems in parameters.
  2. Type of loss function.
  3. More complex CNN models.
  4. About the RNN Model.
- Of course, Python code learning proceeds at the same time.

Reference I

[1] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o $(1/k\hat{}\ 2)$. In *Doklady an ussr*, volume 269, pages 543–547, 1983.

[5] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[6] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Reference II

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[8] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[9] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[10] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.