Gradient Descent Optimization Algorithms: Optimizers

Junkun Yuan
Zhejiang University
yuanjk@zju.edu.cn

1. Introduction

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. the parameters. Learning rate η determines the size of the steps we take to reach a (local) minimum. (From Sec. 1 of [7])

We summarize the optimizers with key phrases:

- **Batch Gradient Descent (BGD):** whole dataset for one update, very slow, massive memory.
- **Stochastic Gradient Descent (SGD):** one example for one update, faster, online, fluctuate.
- **Mini-batch Gradient Descent:** mini-batch examples for one update, trade-off between BGD and SGD.
- **Momentum:** add past update to current, increase for the same direction of gradient and reduced for the change, faster convergence, reduced oscillation.
- **Nesterov Accelerated Gradient (NAG):** look ahead, prevent from going too fast in Momentum.
- **Adagrad:** larger updates for infrequent, sparse data, learning rate divided by the RMS of past gradient.
- **Adadelta:** prevent infinitesimally small of learning rate, do not need default learning rate.
- **RMSprop:** similar to Adadelta except for the fix η.
- **Adam:** use exponentially decaying averages of both past squared gradients and past gradients.
- **Adamax:** v_t with ℓ_∞ converges more stably in Adam.
- **Nadam:** combine Adam with NAG for looking ahead.
- **SGDW/AdamW:** move regularization from gradient update to parameter update, decouple weight decay.

2. Gradient Descent Variants

2.1. Batch Gradient Descent (BGD)

Vanilla gradient descent (BGD) computes the gradient of cost function w.r.t. the parameters θ for the entire dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta). \tag{1}$$

As we calculate the gradients for the whole dataset to perform just one update, it can be very slow and is intractable for datasets that do not fit in memory. It converges to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces. (From Sec. 2.1 of [7])

2.2. Stochastic Gradient Descent (SGD)

SGD updates parameters for each example $(x^{(i)}, y^{(i)})$:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}). \tag{2}$$

BGD performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily. While BGD converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, when we slowly decrease the learning rate, SGD shows the same convergence behaviour as BGD, almost certainly converging to local or global minimum for non-convex and convex optimization respectively. (From Sec. 2.2 of [7])

2.3. Mini-batch Gradient Descent

Mini-batch gradient descent performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}). \tag{3}$$

It a) reduces the variance of parameter updates, leading to more stable convergence; and b) can make use of highly optimized matrix optimizations. (From Sec. 2.3 of [7])

Challenges: (1) Choosing a proper learning rate can be difficult. (2) Learning rate schedules are unadaptable to dataset. (3) The same learning rate applies to all parameter updates. (4) Saddle points are surrounded by a plateau of the same error, which makes it hard to escape, as the gradient is close to zero in all dimensions. (From Sec. 3 of [7])

3. Gradient Descent Optimization Algorithms

3.1. Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. **Momentum** [6] helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction of the update vector of the past time step to the current update vector:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{4}$$

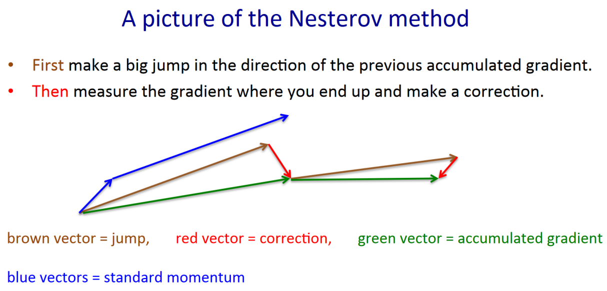
The momentum term γ is usually set to 0.9. Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity, if there is air resistance, i.e. $\gamma < 1$). The same thing happens to parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation. (From Sec. 4.1 of [7])

3.2. Nesterov Accelerated Gradient (NAG)

However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. We would like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again. We know that we will use our momentum term γv_{t-1} to move the parameters θ. Computing $\theta - \gamma v_{t-1}$ thus gives us an approximation of the next position of the parameters (the gradient is missing for the full update), a rough idea where our parameters are going to be. **Nesterov accelerated gradient (NAG)** [5] looks ahead by calculating the gradient not w.r.t. to current parameters but approximate future position:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{5}$$

It prevents us from going too fast and results in increased responsiveness as shown in Fig. 1. (From Sec. 4.2 of [7])



A picture of the Nesterov method

- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.

brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

Figure 1. **Nesterov Accelerated Gradient (NAG)**. While Momentum first computes the current gradient (small blue vector) and then takes a big jump in the direction of the updated accumulated gradient (big blue vector), NAG first makes a big jump in the direction of the previous accumulated gradient (brown vector), measures the gradient and then makes a correction (green vector). (From http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf).

3.3. Adagrad

Now that we are able to adapt our updates to the slope of our error function and speed up SGD in turn, we would also like to adapt our updates to each individual parameter to perform larger or smaller updates depending on their importance. **Adagrad** [2] does just this: It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data. Adagrad uses a different learning rate for every parameter θ_i at every time step t. We set $g_{t,i}$ to be the gradient of the objective function w.r.t. to the parameter θ_i at time step t:

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \tag{6}$$

The SGD update for every parameter θ_i at each time step t:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i} \tag{7}$$

In its update rule, Adagrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{8}$$

$G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each element i, i is the sum of the squares of the past gradients w.r.t. θ_i up to time step t, while ϵ is a smoothing term that avoids division by zero. Interestingly, without the square root operation, the algorithm performs much worse. As G_t contains the sum of the squares of the past gradients w.r.t. to all parameters θ along its diagonal, we can now perform an element-wise matrix-vector multiplication $\odot$ between G_t and g_t:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{9}$$

Adagrad's benefit is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that. (From Sec. 4.3 of [7])

3.4. Adadelta

Adagrad's main weakness is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge. **Adadelta** [8] is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w. Instead of inefficiently storing w previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average $E[g^2]_t$ at time step t then depends (as a fraction γ similarly to the Momentum term) only on the previous average and the current gradient:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \tag{10}$$

γ is set to 0.9. We replace diagonal matrix G_t in Adagrad with decaying average over past squared gradients $E[g^2]_t$:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{11}$$

As the denominator is just the root mean squared (RMS) error criterion of the gradient, we can rewrite it:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t \tag{12}$$

We define another exponentially decaying average, this time not of squared gradients but of squared parameter updates:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \tag{13}$$

The root mean squared error of parameter updates is thus:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2] + \epsilon} \tag{14}$$

Since $RMS[\Delta\theta]_t$ is unknown, we approximate it with the RMS of parameter updates until the previous time step. Replacing the learning rate η in the previous update rule with $RMS[\Delta\theta]_{t-1}$ finally yields the Adadelta update rule:

$$\begin{aligned}\Delta\theta_t &= -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned} \tag{15}$$

Adadelta does not need a default learning rate, as it has been eliminated from update rule. (From Sec. 4.4 of [7])

3.5. RMSprop

RMSprop [1] and Adadelta have both been developed around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates. It in fact is identical to the first update vector of Adadelta:

$$\begin{aligned}E[g^2]_t &= \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t\end{aligned} \tag{16}$$

γ is set to 0.9, and η is set to 0.001. (From Sec. 4.5 of [7])

3.6. Adam

Adaptive Moment Estimation (Adam) [3] computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t:

$$\begin{aligned}m_t &= \beta_1 m_{t-1} + (1-\beta_1)g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)g_t^2\end{aligned} \tag{17}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1). They counteract these biases by computing bias-corrected first and second moment estimates:

$$\begin{aligned}\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}\end{aligned} \tag{18}$$

They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{19}$$

β_1 is set to 0.9, β_2 is set to 0.999. (From Sec. 4.6 of [7])

3.7. AdaMax

We can generalize the v_t factor of Adam to the ℓ_p norm. Note that Kingma and Ba also parameterize β_2 as β_2^p:

$$v_t = \beta_2^p v_{t-1} + (1-\beta_2^p)|g_t|^p \tag{20}$$

Norms for large p values generally become numerically unstable, which is why ℓ_1 and ℓ_2 norms are most common in

[1] http://www.cs.toronto.edu/%7Etijmen/csc321/slides/lecture_slides_lec6.pdf

practice. However, ℓ_∞ also generally exhibits stable behavior. So the authors propose **AdaMax** [3] and show that v_t with ℓ_∞ converges more stably. To avoid confusion with Adam, we use u_t to denote infinity norm-constrained v_t:

$$\begin{aligned} u_t =& \beta_2^\infty u_{t-1} + (1-\beta_2^\infty)|g_t|^\infty \\ =& \max(\beta_2 \cdot u_{t-1}, |g_t|) \end{aligned} \tag{21}$$

We can now plug this into Adam update equation by replacing $\sqrt{\hat{v}_t} + \epsilon$ with u_t to obtain the AdaMax update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t \tag{22}$$

As u_t relies on max operation, it is not as suggestible to bias towards zero as m_t and v_t in Adam, that's why we do not need to compute a bias correction for u_t. Set $\eta = 0.002$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. (From Sec. 4.7 of [7])

3.8. Nadam

Adam can be viewed as a combination of RMSprop and momentum: RMSprop contributes exponentially decaying average of past squared gradients, while momentum accounts for exponentially decaying average of past gradients. Since NAG is superior to momentum, **Nadam (Nesterov-accelerated Adaptive Moment Estimation)** [1] thus combines Adam and NAG. Recall momentum rule of Eq. (4):

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t) \tag{23}$$

NAG then performs a more accurate step by looking ahead:

$$\begin{aligned} g_t =& \nabla_{\theta_t} J(\theta_t - \gamma m_{t-1}) \\ m_t =& \gamma m_{t-1} + \eta g_t \\ \theta_{t+1} =& \theta_t - m_t \end{aligned} \tag{24}$$

We now apply the look-ahead momentum vector directly:

$$\begin{aligned} g_t =& \nabla_{\theta_t} J(\theta_t) \\ m_t =& \gamma m_{t-1} + \eta g_t \\ \theta_{t+1} =& \theta_t - (\gamma m_t + \eta g_t) \end{aligned} \tag{25}$$

To add Nesterov momentum to Adam, we can similarly replace previous momentum vector with current momentum vector. Combing Eq. (17), Eq. (18), and Eq. (19) of Adam:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \tag{26}$$

We replace $\frac{\beta_1 m_{t-1}}{1-\beta_1^t}$ with $\hat{m}_{t-1}$ in Eq. (26) and yield:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1 \hat{m}_{t-1} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \tag{27}$$

It looks similar to our expanded momentum in Eq. (23). We add Nesterov momentum just as we did in Eq. (25), yielding the Nadam update rule (from Sec. 4.8 of [7]):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \tag{28}$$

3.9. SGDW/AdamW

In practice, we often add $\frac{1}{2}\lambda\|\theta\|^2$ regularization term to objective function for guiding the training process. It is equivalent to add $\lambda\theta$ to the gradient for each algorithm:

$$g_t = \nabla_\theta J(\theta) + \lambda\theta_{t-1} \tag{29}$$

However, for the algorithms with weight decay, e.g., SGD (Momentum) and Adam, the regularization can influence the effectiveness of the weight decay. **SGDW and AdamW** [4] are thus proposed to decouple these two factors by moving the regularization term from gradient update to parameter update. In this way, the update rule of SGDW algorithm can be redefined from Eq. (4) to the following equation:

$$\begin{aligned} g_t =& \nabla_\theta J(\theta) + \lambda\theta_{t-1} \\ m_t =& \gamma m_{t-1} + \eta g_t \\ \theta_t =& \theta_{t-1} - m_t - \eta\lambda\theta_{t-1} \end{aligned} \tag{30}$$

Similarly, update rule of AdamW is redefined from Eq. (19):

$$\begin{aligned} g_t =& \nabla_\theta J(\theta) + \lambda\theta_{t-1} \\ m_t =& \beta_1 m_{t-1} + (1-\beta_1)g_t \\ v_t =& \beta_2 v_{t-1} + (1-\beta_2)g_t^2 \\ \hat{m}_t =& \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t =& \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} =& \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t - \eta\lambda\theta_{t-1} \end{aligned} \tag{31}$$

4. Resources

- A terrific introduction to gradient descent optimization algorithms [7] (this paper is mostly based on it).

References

[1] Timothy Dozat. Incorporating nesterov momentum into adam. 2016. 4

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 2

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3, 4

[4] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 4

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

[6] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999. 2

[7] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 1, 2, 3, 4

[8] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 3