Photo by Julian Zett on Unsplash

10 Gradient Descent Optimisation Algorithms + Cheat Sheet

Gradient descent optimisation algorithms you should know for deep learning

Raimi Karim
Nov 22, 2018 · 10 min read

(I maintain a cheat sheet of these optimisers including RAdam in my blog here.)

Change logs:
6 Oct 2019: Corrected the idea of EMA of gradients.
22 Sep 2019: Rearranged the order in which optimisers appear and removed 'evolutionary map'.

Gradient descent is an optimisation method for finding the minimum of a function. It is commonly used in deep learning models to update the weights of the neural network through backpropagation.

In this post, I will summarise the common gradient descent optimisation algorithms used in popular deep learning frameworks (e.g. TensorFlow, Keras, PyTorch, Caffe). The purpose of this post is to make it easy to read and digest (using consistent nomenclature) since there aren't many of such summaries out there, and as a cheat sheet if you want to implement them from scratch.

I have implemented SGD, momentum, Nesterov, RMSprop and Adam in a linear regression problem using gradient descent demo here using JavaScript.

What do gradient descent optimisers do?

There are 3 main ways how these optimisers can act upon gradient descent:
(1) modifying the learning rate component, $\boldsymbol{\alpha}$, or
(2) modifying the gradient component, $\boldsymbol{\partial L/\partial w}$, or
(3) both.
See the last term in Eqn. 1 below:

$$w_{\text{new}} = w - \alpha \frac{\partial L}{\partial w}$$

Eqn. 1: The terms in stochastic gradient descent

> **Learning rate schedulers vs. Gradient descent optimisers**
> *The main difference between these two is that gradient descent optimisers adapt the learning rate component by multiplying the learning rate with a factor that is a function of the gradients, whereas learning rate schedulers multiply the learning rate by a factor which is a constant or a function of the time step.*

For (1), these optimisers multiply a positive factor to the learning rate, such that they become smaller (e.g. RMSprop). For (2), optimisers usually make use of the moving averages of the gradient (momentum), instead of just taking one value like in vanilla gradient descent. Optimisers that act on both (3) are like Adam and AMSGrad.

Optimiser	Year	Learning Rate	Gradient
Momentum	1964		✓
AdaGrad	2011	✓	
RMSprop	2012	✓	
Adadelta	2012	✓	
Nesterov	2013		✓
Adam	2014	✓	✓
AdaMax	2015	✓	✓
Nadam	2015	✓	✓
AMSGrad	2018	✓	✓

Fig. 2: Gradient descent optimisers, the year in which the papers were published, and the components they act upon

<change log 22 Sep 2019: removed evolutionary map>

Notations

- t — time step
- w — weight/parameter which we want to update
- α — learning rate
- $\partial L/\partial w$ — gradient of L, the loss function to minimise, w.r.t. to w
- I have also standardised the notations and Greek letters used in this post (hence might be different from the papers) so that we can explore how optimisers 'evolve' as we scroll.

. . .

Content

. . .

1. Stochastic Gradient Descent

The vanilla gradient descent updates the current weight w using the current gradient $\partial L/\partial w$ multiplied by some factor called the learning rate, α.

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

2. Momentum

Instead of depending only on the current gradient to update the weight, gradient descent with momentum (Polyak, 1964) replaces the current gradient with V (which stands for velocity), the exponential moving average of current and past gradients (i.e. up to time t). Later in this post you will see that this momentum update becomes the standard update for the gradient component.

$$w_{t+1} = w_t - \alpha V_t$$

where

$$V_t = \beta V_{t-1} + (1-\beta)\frac{\partial L}{\partial w_t}$$

and V initialised to 0.

Common default value:

- $\beta = 0.9$

> *Note that many articles reference the momentum method to the publication by Ning Qian, 1999. However, the paper titled Sutskever et al. attributed the classical momentum to a much earlier publication by Polyak in 1964, as cited above. (Thank you to James for pointing this out.)*

3. AdaGrad

Adaptive gradient, or AdaGrad (Duchi et al., 2011), works on the learning rate component by dividing the learning rate by the square root of *S*, which is the cumulative sum of current and past squared gradients (i.e. up to time *t*). Note that the gradient component remains unchanged like in SGD.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

where

$$S_t = S_{t-1} + \left[\frac{\partial L}{\partial w_t}\right]^2$$

and *S* initialised to 0.

Notice that ε is added to the denominator. Keras calls this the *fuzz factor*, a small floating point value to ensure that we will never have to come across division by zero.

Default values (from Keras):

- $\alpha = 0.01$
- $\varepsilon = 10^{-7}$

4. RMSprop

Root mean square prop or RMSprop (Hinton et al., 2012) is another adaptive learning rate that is an improvement of AdaGrad. Instead of taking cumulative sum of squared gradients like in AdaGrad, we take the exponential moving average of these gradients.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

where

$$S_t = \beta S_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right]^2$$

and *S* initialised to 0.

Default values (from Keras):

- $\alpha = 0.001$
- $\beta = 0.9$ (recommended by the authors of the paper)
- $\varepsilon = 10^{-6}$

5. Adadelta

Like RMSprop, Adadelta (Zeiler, 2012) is also another improvement from AdaGrad, focusing on the learning rate component. Adadelta is probably short for 'adaptive delta', where *delta* here refers to the difference between the current weight and the newly updated weight.

The difference between Adadelta and RMSprop is that Adadelta removes the use of the learning rate parameter completely by replacing it with *D,* the exponential moving average of squared *deltas*.

$$w_{t+1} = w_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

where

$$D_t = \beta D_{t-1} + (1-\beta)[\Delta w_t]^2$$
$$S_t = \beta S_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right]^2$$

with *D* and *S* initialised to 0, and

$$\Delta w_t = w_t - w_{t-1}$$

Default values (from Keras):

- $\beta = 0.95$

- $\varepsilon = 10^{-6}$

6. Nesterov Accelerated Gradient (NAG)

After Polyak had gained his momentum (pun intended 😬), a similar update was implemented using Nesterov Accelerated Gradient (Sutskever et al., 2013). This update utilises V, the exponential moving average of what I would call *projected gradients*.

$$w_{t+1} = w_t - \alpha V_t$$

where

$$V_t = \beta V_{t-1} + (1-\beta)\frac{\partial L}{\partial w^*}$$

and V initialised to 0.

The last term in the second equation is a *projected gradient*. This value can be obtained by going 'one step ahead' using the previous velocity (Eqn. 4). This means that for this time step t, we have to carry out another forward propagation before we can finally execute the backpropagation. Here's how it goes:

1. Update the current weight w to a *projected weight w^** using the previous velocity.

$$w^* = w_t - \alpha V_{t-1}$$

Eqn. 4

2. Carry out forward propagation, but using this *projected weight*.

3. Obtain the *projected gradient ∂L/∂w**.

4. Compute V and w accordingly.

Common default value:

- $\beta = 0.9$

> *Note that the original Nesterov Accelerated Gradient paper (Nesterov, 1983) was not about* stochastic *gradient descent and did not explicitly use the gradient descent equation.*

> *Hence, a more appropriate reference is the above-mentioned publication by Sutskever et al. in 2013, which described NAG's application in stochastic gradient descent. (Again, I'd like to thank James's comment on HackerNews for pointing this out.)*

7. Adam

Adaptive moment estimation, or Adam (Kingma & Ba, 2014), is a combination of momentum and RMSprop. It acts upon
(i) the gradient component by using *V*, the exponential moving average of gradients (like in momentum) and
(ii) the learning rate component by dividing the learning rate *α* by square root of *S*, the exponential moving average of squared gradients (like in RMSprop).

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon} \cdot \hat{V}_t$$

where

$$\hat{V}_t = \frac{V_t}{1 - \beta_1^t}$$
$$\hat{S}_t = \frac{S_t}{1 - \beta_2^t}$$

are the bias corrections, and

$$V_t = \beta_1 V_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t}$$
$$S_t = \beta_2 S_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2$$

with *V* and *S* initialised to 0.

Proposed default values by the authors:

- $\alpha = 0.001$

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\varepsilon = 10^{-8}$

8. AdaMax

AdaMax (Kingma & Ba, 2015) is an adaptation of the Adam optimiser by the same authors using infinity norms (hence 'max'). *V* is the exponential moving average of gradients, and *S* is the exponential moving average of past *p*-norm of gradients, approximated to the max function as seen below (see paper for convergence proof).

$$w_{t+1} = w_t - \frac{\alpha}{S_t} \cdot \hat{V}_t$$

where

$$\hat{V}_t = \frac{V_t}{1 - \beta_1^t}$$

is the bias correction for *V* and

$$V_t = \beta_1 V_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t}$$
$$S_t = \max(\beta_2 S_t, \left| \frac{\partial L}{\partial w_t} \right|)$$

with *V* and *S* initialised to 0.

Proposed default values by the authors:

- $\alpha = 0.002$
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$

9. Nadam

Nadam (Dozat, 2015) is an acronym for Nesterov and Adam optimiser. The Nesterov component, however, is a more efficient modification than its original implementation.

First we'd like to show that the Adam optimiser can also be written as:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon}\left(\beta_1 \hat{V}_{t-1} + \frac{1-\beta_1}{1-\beta_1^t} \cdot \frac{\partial L}{\partial w_t}\right)$$

Eqn. 5: Weight update for Adam optimiser

Nadam makes use of Nesterov to update the gradient one step ahead by replacing the previous *V_hat* in the above equation to the current *V_hat*:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon}\left(\beta_1 \hat{V}_t + \frac{1-\beta_1}{1-\beta_1^t} \cdot \frac{\partial L}{\partial w_t}\right)$$

where

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t}$$
$$\hat{S}_t = \frac{S_t}{1-\beta_2^t}$$

and

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$
$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2$$

with *V* and *S* initialised to 0.

Default values (taken from Keras):

- $\alpha = 0.002$

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\varepsilon = 10^{-7}$

10. AMSGrad

Another variant of Adam is the AMSGrad (Reddi et al., 2018). This variant revisits the adaptive learning rate component in Adam and changes it to ensure that the current *S* is always larger than the previous time step.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon} \cdot V_t$$

where

$$\hat{S}_t = \max(\hat{S}_{t-1}, S_t)$$

and

$$V_t = \beta_1 V_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t}$$
$$S_t = \beta_2 S_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2$$

with *V* and *S* initialised to 0.

Default values (taken from Keras):

- $\alpha = 0.001$
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\varepsilon = 10^{-7}$

· · ·

Intuition

Here I'd like to share with you some intuition why gradient descent optimisers use exponential moving average for the gradient component and root mean square for the learning rate component.

Why take *exponential moving average* of gradients?

We need to update the weight, and to do so we need to make use of some value. The only value we have is the current gradient, so let's utilise this to update the weight.

But taking only the current gradient value is not enough. We want our updates to be 'better guided'. So let's include previous gradients too.

One way to 'combine' the current gradient value and information of past gradients is that we could take a simple average of all the past and current gradients. But this means each of these gradients are equally weighted.

What we could do is to take the exponential moving average, where past gradient values are given higher weights (importance) than the current one. Intuitively, discounting the importance given to the current gradient would ensure that the weight update will not be sensitive to the current gradient.

Why divide learning rate by *root mean square* of gradients?

The goal is to adapt the learning rate component. Adapt to what? The gradient. All we need to ensure is that when the gradient is large, we want the update to be small (otherwise, a huge value will be subtracted from the current weight!).

In order to create this effect, let's *divide* the learning rate α by the current gradient to get an adapted learning rate.

Bear in mind that the learning rate component must always be positive (because the learning rate component, when multiplied with the gradient component, should have the same sign as the latter). To ensure it's always positive, we can take its absolute value or its square. Let's take the square of the current gradient and 'cancel' back this square by taking its square root.

But like momentum, taking only the current gradient value is not enough. We want our updates to be 'better guided'. So let's make use of previous gradients too. And, as discussed above, we'll take the exponential moving average of past gradients ('mean square'), then taking its square root ('root'), hence 'root mean square'. All optimisers in this post which act on the learning rate component does this, except for AdaGrad (which takes cumulative sum of squared gradients).

. . .

Cheat Sheet

Vanilla SGD

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

Momentum

$$w_{t+1} = w_t - \alpha V_t$$

$$V_t = \beta V_{t-1} + (1-\beta) \frac{\partial L}{\partial w_t}$$

Nesterov

$$w_{t+1} = w_t - \alpha V_t$$

$$V_t = \beta V_{t-1} + (1-\beta) \frac{\partial L}{\partial w^*}$$

$$w^* = w_t - \alpha V_{t-1}$$

Adagrad

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$S_t = S_{t-1} + \left[\frac{\partial L}{\partial w_t}\right]^2$$

RMSprop

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$S_t = \beta S_{t-1} + (1-\beta) \left[\frac{\partial L}{\partial w_t}\right]^2$$

Adadelta

$$w_{t+1} = w_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$D_t = \beta D_{t-1} + (1-\beta)[\Delta w_t]^2$$

$$S_t = \beta S_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right]^2$$

Adam

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon} \cdot \hat{V}_t$$

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t}$$

$$\hat{S}_t = \frac{S_t}{1-\beta_2^t}$$

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2$$

AdaMax

$$w_{t+1} = w_t - \frac{\alpha}{S_t} \cdot \hat{V}_t$$

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t}$$

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$S_t = \max(\beta_2 S_{t-1}, \left|\frac{\partial L}{\partial w_t}\right|)$$

Nadam

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon}\left(\beta_1 \hat{V}_{t-1} + \frac{1-\beta_1}{1-\beta_1^t} \cdot \frac{\partial L}{\partial w_t}\right)$$

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t}$$

$$\hat{S}_t = \frac{S_t}{1-\beta_2^t}$$

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2$$

AMSGrad

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon} \cdot V_t$$

$$\hat{S}_t = \max(\hat{S}_{t-1}, S_t)$$

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\left[\partial L\right]^2$$

$$S_t = \rho_2 S_{t-1} + (1 - \rho_2)\left[\frac{}{\partial w_t}\right]$$

(gist for the above)

Please reach out to me if something is amiss, or if something in this post can be improved! ✌️

. . .

References

An overview of gradient descent optimization algorithms (ruder.io)

Why Momentum Really Works (distill.pub)

Related Articles on Deep Learning

Animated RNN, LSTM and GRU

Line-by-Line Word2Vec Implementation (on word embeddings)

Step-by-Step Tutorial on Linear Regression with Stochastic Gradient Descent

Counting No. of Parameters in Deep Learning Models

Attn: Illustrated Attention

Illustrated: Self-Attention

. . .

Thanks to Ren Jie, Derek, William Tjhi, Chan Kai, Serene and James for ideas, suggestions and corrections to this article.

Follow me on Twitter @remykarem or LinkedIn. You may also reach out to me via raimi.bkarim@gmail.com. Feel free to visit my website at remykarem.github.io.

Machine Learning Gradient Descent Deep Learning Keras TensorFlow