The Gradient Descent Optimization Algorithms

Haobin Tan | 17. December 2019

SEMINAR:NEURONALE NETZE UND KÜNSTLICHE INTELLIGENZ

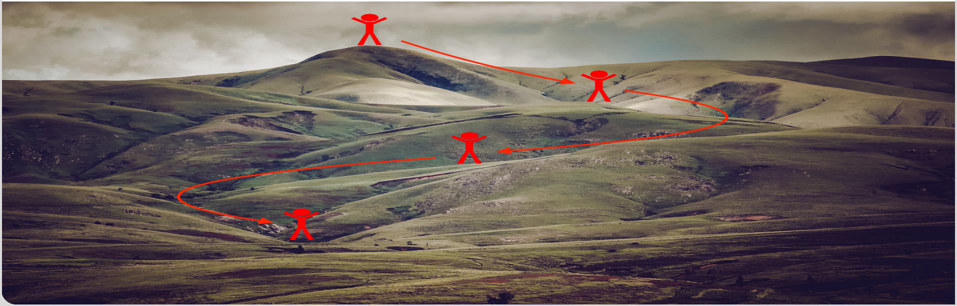


Table of Contents

Notation

- $\theta \in \mathbb{R}^d$: model parameters
- $J(\theta)$: loss function
- x's: input variables/features
- y's: output/"target" variables
- $(x^{(i)}, y^{(i)})$: i-th training example
- η: learning rate

Table of Contents

Gradient descent

- **First-order** optimization algorithm for finding the minimum of the loss function
- Iteratively updates the parameters in **opposite direction** of the gradient
- Update rule:

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

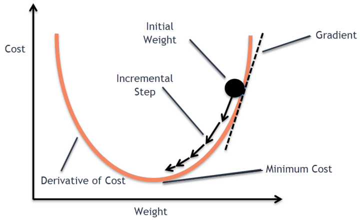


Figure: Source: Stochastic vs Batch Gradient Descent

Gradient descent variants

Difference: Amount of data used per update

- Batch Gradient Descent (BGD)
- Stochastic Gradient Descent (SGD)
- Mini-Batch Gradient Descent (MBGD)

Batch Gradient Descent

Batch Gradient Descent (BGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2}$$

- Computes gradient with the **whole** training dataset

Batch Gradient Descent

Batch Gradient Descent (BGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2}$$

- Computes gradient with the **whole** training dataset
- Pros:
  - Guarantees to converge

Batch Gradient Descent

Batch Gradient Descent (BGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2}$$

- Computes gradient with the **whole** training dataset
- Pros:
  - Guarantees to converge
- Cons:
  - Very slow
  - Intractable for very large dataset
  - No online learning

Batch Gradient Descent

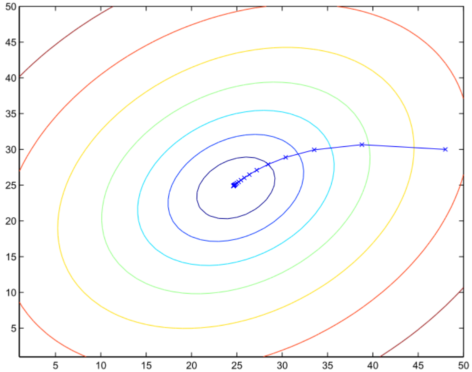


Figure: Source: Ng [2000]

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right) \tag{3}$$

- Performs update for **each** training examples $(x^{(i)}, y^{(i)})$

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right) \tag{3}$$

- Performs update for **each** training examples $(x^{(i)}, y^{(i)})$
- Pros:
  - Fast
  - Allows online learning

Stochastic Gradient Descent: Cons

- High variance updates

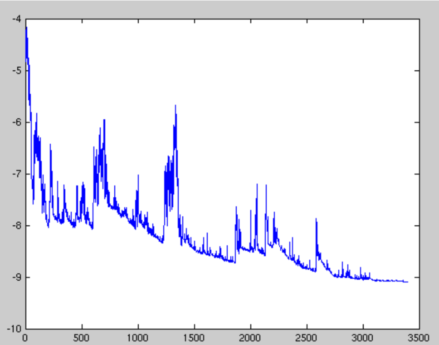


Figure: Source: Ruder [2016]

Stochastic Gradient Descent: Cons

- Complicates convergence

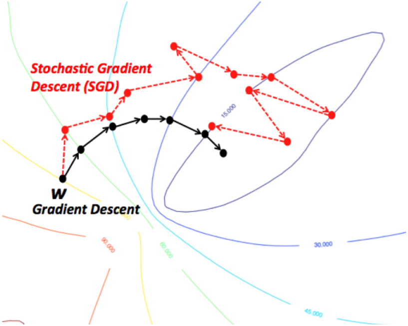


Figure: Source:
`https://wikidocs.net/3413`

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent (MBGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i:i+b)}; y^{(i:i+b)}\right) \tag{4}$$

- *b*: Batch size (usually $50 \sim 256$)

- Computes gradient for **small random sets** of training examples

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent (MBGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i:i+b)}; y^{(i:i+b)}\right) \tag{4}$$

- b: Batch size (usually $50 \sim 256$)

- Computes gradient for **small random sets** of training examples
- Pros:
  - Reduces variance of updates
  - Performance boost from hardware optimization of matrix operations

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent (MBGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i:i+b)}; y^{(i:i+b)}\right) \tag{4}$$

- b: Batch size (usually $50 \sim 256$)

- Computes gradient for **small random sets** of training examples
- Pros:
  - Reduces variance of updates
  - Performance boost from hardware optimization of matrix operations
- **Typically the algorithm of choice**

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent (MBGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i:i+b)}; y^{(i:i+b)}\right) \tag{4}$$

- *b*: Batch size (usually 50 $\sim$ 256)

- Computes gradient for **small random sets** of training examples
- Pros:
  - Reduces variance of updates
  - Performance boost from hardware optimization of matrix operations
- **Typically the algorithm of choice**
- **Usually referred to as SGD**

Comparison and trade-offs

Method	Accuracy	Update Speed	Memory Usage	Online Learning
BGD	very good	slow	high	no
SGD	good (with annealing)	high	low	yes
MBGD	good	medium	medium	yes

Table: Gradient descent variants comparison

Comparison and trade-offs

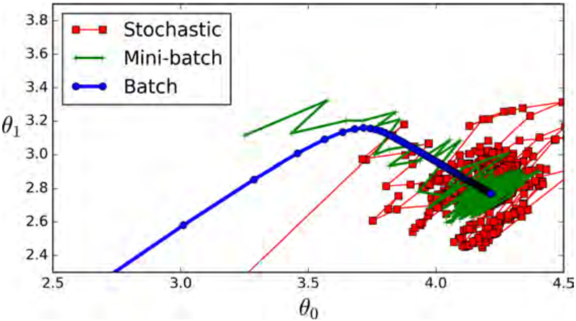


Figure: Source: Géron [2017]

Table of Contents

Choosing the proper learning rate

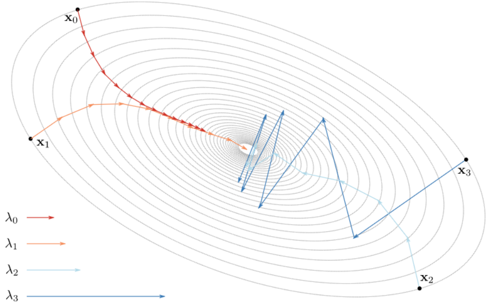


Figure: Source: `https://blog.yani.io/sgd/`

- Too small (λ_0)

Choosing the proper learning rate

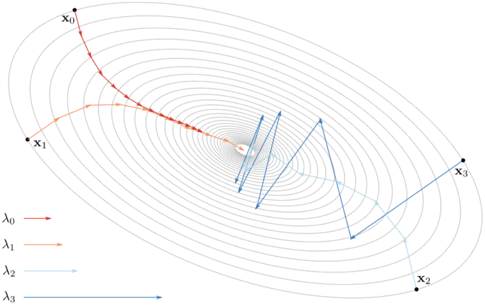


Figure: Source: `https://blog.yani.io/sgd/`

- Proper (λ_1)

Choosing the proper learning rate

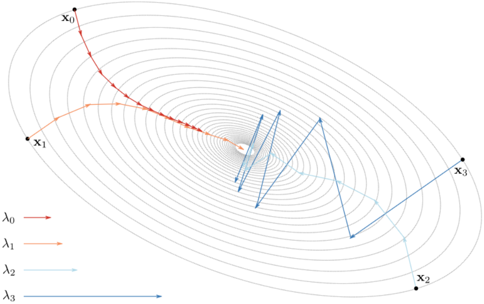


Figure: Source: `https://blog.yani.io/sgd/`

- Too large (λ_2, λ_3)

Non-convex Loss function

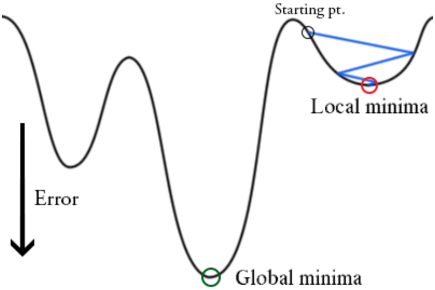


Figure: Source: Non-convex optimization

- Stucks in local minima

Non-convex Loss function

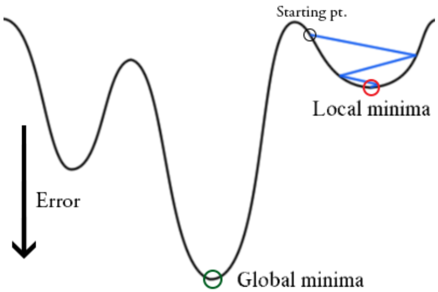


Figure: Source: Non-convex optimization

- Stucks in local minima
- Can not get to the global minima

Table of Contents

Optimization aspects

Update rule of gradient descent:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Optimization aspects

Update rule of gradient descent:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Consider as

$$\theta = \theta - \text{step size} \cdot \text{step direction}$$

Optimization aspects

Update rule of gradient descent:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Consider as

$$\theta = \theta - \text{step size} \cdot \textbf{step direction}$$

- **Step direction**
  - Momentum
  - Nesterov Momentum

Optimization aspects

Update rule of gradient descent:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Consider as

$$\theta = \theta - \textbf{step size} \cdot \text{step direction}$$

- Step direction
  - Momentum
  - Nesterov Momentum
- **Step size**
  - AdaGrad
  - Adadelta
  - RMSprop
  - Adam

Momentum

Momentum (Sutskever et al. [2013])

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

- $\mu \in [0, 1)$: “friction” coefficient (usually 0.9)

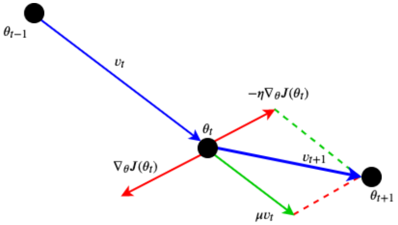


Figure: Momentum update at step t

Momentum

- **Accelerates** if the gradients changes **in the same direction** ($\rightarrow$ faster convergence)
- **Reduces** the updates if the gradient **changes direction** ($\rightarrow$ less fluctuations)

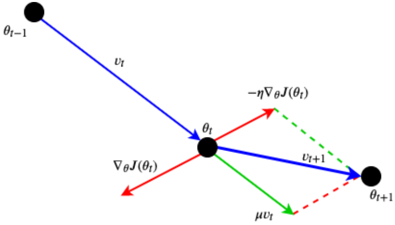


Figure: Momentum update at step t

Momentum

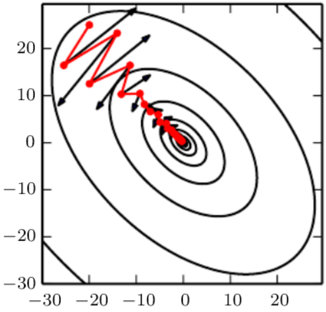


Figure: Source: Goodfellow et al. [2016]

- $\rightarrow$: SGD without momentum
- $\rightarrow$: SGD with momentum

Nesterov Momentum

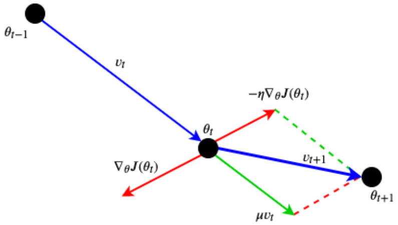
Nesterov Momentum (Sutskever et al. [2013])

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \nabla J_\theta \left(\theta_t + \mu v_t\right) \\ \theta_{t+1} &= \theta_t + v_{t+1}, \end{aligned} \tag{5}$$

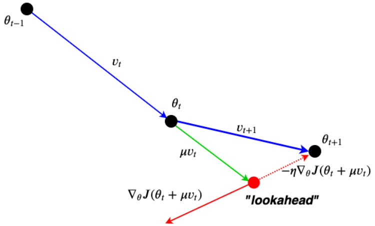
- μ: as in Momentum

Nesterov Momentum vs. Momentum

- Momentum
  - computes gradient at **current** position



- Nesterov momentum
  - computes gradient at ***"lookahead"*** position



AdaGrad

Adaptive **Grad**ient Algorithm

AdaGrad (Duchi et al. [2011])

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \tag{6}$$

- g_t: gradient of the loss function $J(\theta)$ w.r.t the parameter θ at step t
- ϵ: smoothing item, aims to prevent division by 0 (usually 10^{-7})
- Division and square root: element-wise operation
- $$G_t = \begin{pmatrix} \sum_{\tau=1}^{t} g_{\tau,1}^2 & 0 & \cdots & 0 \\ 0 & \sum_{\tau=1}^{t} g_{\tau,2}^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{\tau=1}^{t} g_{\tau,d}^2 \end{pmatrix} \in \mathbb{R}^{d \times d}$$

AdaGrad

AdaGrad (Duchi et al. [2011])

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \tag{6}$$

- Each parameter θ_i is updated with different learning rate, depending on the past gradients (G_t)
  - **Large** updates for **infrequent** parameters
  - **Small** updates for **frequent** parameters

AdaGrad

AdaGrad (Duchi et al. [2011])

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \tag{6}$$

- Each parameter θ_i is updated with different learning rate, depending on the past gradients (G_t)
  - **Large** updates for **infrequent** parameters
  - **Small** updates for **frequent** parameters

$\rightarrow$ **adaptive**

AdaGrad

AdaGrad (Duchi et al. [2011])

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \tag{6}$$

- Pros
  - Lesser need to manually tune learning rate
  - Suitable for sparse data
  - Improve robustness
- Cons
  - Continual shrinking of learning rate
  - Still need to manually select a global learning rate

Adadelta

- Proposed by Zeiler [2012]
- Extension of AdaGrad
- Improve upon two main disadvantages of AdaGrad
  - Continual shrinking of learning rate throughout training
  - Necessity of a manually selected global learning rate

Adadelta: Accumulate over window

- AdaGrad: accumulates **all** previous squared gradients

Adadelta: Accumulate over window

- AdaGrad: accumulates **all** previous squared gradients
- Adadelta: restricts the window of past accumulated squared gradients to a **fixed** size

Adadelta: Accumulate over window

- AdaGrad: accumulates **all** previous squared gradients
- Adadelta: restricts the window of past accumulated squared gradients to a **fixed** size
  - Approximated with Exponentially Weighted Moving Average (EWMA):

$$E\left[g^2\right]_t = \mu E\left[g^2\right]_{t-1} + (1-\mu)g_t^2, \quad \mu \in [0,1) \tag{7}$$

Adadelta: Accumulate over window

- AdaGrad: accumulates **all** previous squared gradients
- Adadelta: restricts the window of past accumulated squared gradients to a **fixed** size
  - Approximated with Exponentially Weighted Moving Average (EWMA):

$$E\left[g^2\right]_t = \mu E\left[g^2\right]_{t-1} + (1-\mu)g_t^2, \quad \mu \in [0,1) \tag{7}$$

  - Replace G_t in AdaGrad (Equation (6)):

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{8}$$

Adadelta: Accumulate over window

- AdaGrad: accumulates **all** previous squared gradients
- Adadelta: restricts the window of past accumulated squared gradients to a **fixed** size
  - Approximated with Exponentially Weighted Moving Average (EWMA):

$$E\left[g^2\right]_t = \mu E\left[g^2\right]_{t-1} + (1-\mu)g_t^2, \quad \mu \in [0,1] \tag{7}$$

  - Replace G_t in AdaGrad (Equation (6)):

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{8}$$

  - Denominator is Root Mean Square (RMS) of the gradient:

$$\mathbf{RMS}[g]_t = \sqrt{E[g^2]_t + \epsilon} \tag{9}$$

Adadelta: Accumulate over window

- AdaGrad: accumulates **all** previous squared gradients
- Adadelta: restricts the window of past accumulated squared gradients to a **fixed** size
  - Approximated with Exponentially Weighted Moving Average (EWMA):

$$E\left[g^2\right]_t = \mu E\left[g^2\right]_{t-1} + (1-\mu)g_t^2, \quad \mu \in [0,1) \tag{7}$$

  - Replace G_t in AdaGrad (Equation (6)):

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{8}$$

  - Denominator is Root Mean Square (RMS) of the gradient:

$$\mathbf{RMS}[g]_t = \sqrt{E[g^2]_t + \epsilon} \tag{9}$$

  - Finally

$$\Delta\theta_t = -\frac{\eta}{\mathbf{RMS}[g]_t} \cdot g_t \tag{10}$$

Adadelta: Correct units with Hessian approximation

- Update rule of gradient descent:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \nabla_\theta J(\theta_t) \\ &= \theta_t - \eta \cdot g_t\end{aligned}$$

- Newton's method:

$$\begin{aligned}\theta_{t+1} &= \theta_t - D^2 J(\theta_t)^{-1} \nabla_\theta J(\theta_t) \\ &= \theta_t - \mathbf{H}(J(\theta_t))^{-1} \cdot g_t\end{aligned} \tag{11}$$

Adadelta: Correct units with Hessian approximation

- Update rule of gradient descent:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \nabla_\theta J(\theta_t) \\ &= \theta_t - \eta \cdot g_t\end{aligned}$$

- Newton's method:

$$\begin{aligned}\theta_{t+1} &= \theta_t - D^2 J(\theta_t)^{-1} \nabla_\theta J(\theta_t) \\ &= \theta_t - \mathbf{H}(J(\theta_t))^{-1} \cdot g_t\end{aligned} \tag{11}$$

$\rightarrow$ $\mathbf{H}(J(\theta_t))^{-1}$ as "automatically adaptive" learning rate

Adadelta: Correct Units with Hessian Approximation

- How to compute $\mathbf{H}(J(\theta_t))^{-1}$?

Adadelta: Correct Units with Hessian Approximation

- How to compute $\mathbf{H}(J(\theta_t))^{-1}$?
- Diagonal approximation to the Hessian proposed by Becker et al. [1988]:

$$\Delta\theta_t = -\frac{1}{|\mathrm{diag}(\mathbf{H}_t)| + \mu} g_t \tag{12}$$

  - $\mathbf{H}_t := \mathbf{H}(J(\theta_t))$
  - $\mathrm{diag}(H_t)$: diagonal Hessian
  - μ: Smoothing item, aims to prevent division by 0

Adadelta: Correct Units with Hessian Approximation

- How to compute $\mathbf{H}(J(\theta_t))^{-1}$?
- Diagonal approximation to the Hessian proposed by Becker et al. [1988]:

$$\Delta\theta_t = -\frac{1}{\mathbf{H}_t} g_t \tag{12}$$

  - $\mathbf{H}_t := \mathbf{H}(J(\theta_t))$
  - $\text{diag}(H_t)$: diagonal Hessian
  - μ: Smoothing item, aims to prevent division by 0

Adadelta: Correct units with Hessian approximation

(Assuming a diagonal Hessian):

$$\mathbf{H}_t^{-1} \approx \frac{1}{\mathbf{H}_t} = \frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} \tag{13}$$

Adadelta: Correct units with Hessian approximation

(Assuming a diagonal Hessian):

$$\mathbf{H}_t^{-1} \approx \frac{1}{\mathbf{H}_t} = \frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} \tag{13}$$

Rearrange Newton's method:

$$\Delta\theta_t \approx \frac{g_t}{\mathbf{H}_t} = \frac{\frac{\partial J}{\partial \theta_t}}{\frac{\partial^2 J}{\partial \theta_t^2}} \tag{14}$$

$$\Rightarrow \frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} = \frac{\Delta\theta_t}{\frac{\partial J}{\partial \theta_t}} \tag{15}$$

Adadelta: Correct Units with Hessian Approximation

$$\frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} = \frac{\Delta \theta_t}{\frac{\partial J}{\partial \theta_t}}$$

Adadelta: Correct Units with Hessian Approximation

$$\frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} = \frac{\Delta \theta_t}{\frac{\partial J}{\partial \theta_t}}$$

- Estimate $\frac{\partial J}{\partial \theta_t}$ with EWMA of the previous gradient:

$$\frac{\partial J}{\partial \theta_t} \approx \textbf{RMS}[g]_t$$

Adadelta: Correct Units with Hessian Approximation

$$\frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} = \frac{\Delta \theta_t}{\textbf{RMS}[g]_t}$$

Adadelta: Correct Units with Hessian Approximation

$$\frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} = \frac{\Delta\theta_t}{\mathbf{RMS}[g]_t}$$

- Estimate $\Delta\theta_t$ with EWMA of the previous $\Delta\theta$ (assuming the curvature is locally smooth):

$$E\left[\Delta\theta^2\right]_{t-1} = \mu E\left[\Delta\theta^2\right]_{t-2} + (1-\mu)\Delta\theta_{t-1}^2$$

$$\mathbf{RMS}[\Delta\theta]_{t-1} = \sqrt{E\left[\Delta\theta^2\right]_{t-1} + \epsilon}$$

$$\Delta\theta_t \approx \mathbf{RMS}[\Delta\theta]_{t-1}$$

Adadelta: Correct Units with Hessian Approximation

$$\frac{1}{\frac{\partial^2 J}{\partial \theta_t^2}} = \frac{\mathbf{RMS}[\Delta\theta]_{t-1}}{\mathbf{RMS}[g]_t} \tag{16}$$

Adadelta: Update Rule

$$\begin{aligned}\theta_{t+1} &= \theta_t - \mathbf{H}_t^{-1} g_t \\ &\overset{(13)}{\approx} \theta_t - \frac{1}{\mathbf{H}_t} g_t \\ &\overset{(16)}{=} \frac{\mathbf{RMS}[\Delta\theta]_{t-1}}{\mathbf{RMS}[g]_t} g_t\end{aligned}$$

Adadelta (Zeiler [2012])

$$\theta_{t+1} = \theta_t - \frac{\mathbf{RMS}[\Delta\theta]_{t-1}}{\mathbf{RMS}[g]_t} g_t \tag{17}$$

RMSprop

- Unpublished optimization algorithm
- Proposed by Geoff Hinton in his Coursera course[1]

RMSprop

$$
\begin{aligned}
E\left[g^2\right]_t &= \mu E\left[g^2\right]_{t-1} + (1-\mu) g_t^2 \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \\
&\overset{(9)}{=} \theta_t - \frac{\eta}{\mathbf{RMS}[g]_t} \cdot g_t
\end{aligned} \tag{18}
$$

- μ: Decaying hyperparameter (typically 0.9)
- ϵ: Smoothing item, aims to prevent division by 0
- Good default value for η: 0.001

[1] www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adam

- **Ada**ptive **M**oment Estimation

Adam

- **Ada**ptive **M**oment Estimation
- Combination of Momentum and RMSprop

Adam

- **Ada**ptive **M**oment Estimation
- Combination of **Momentum** and RMSprop
  - Stores moving average of past gradients (like Momentum)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{19}$$

- m_t: first moment (mean) of gradients
- β_1: decaying rate (default: 0.9)

Adam

- **Ada**ptive **M**oment Estimation
- Combination of Momentum and **RMSprop**
  - Stores moving average of past gradients (like Momentum)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \tag{19}$$

    - m_t: first moment (mean) of gradients
    - β_1: decaying rate (default: 0.9)
  - Stores moving average of past squared gradients (like RMSprop)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2 \tag{20}$$

    - v_t: second moment (uncentered variance) of gradients
    - β_2: decaying rate (default: 0.999)

Adam: Bias correction

- m_t and v_t are initialized as $\mathbf{0}$-vectors. $\rightarrow$ biased towards 0 at the beginning

Adam: Bias correction

- m_t and v_t are initialized as **0**-vectors. $\rightarrow$ biased towards 0 at the beginning
- Bias correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{21}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{22}$$

Adam: Update rule

Adam (Kingma and Ba [2014])

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{23}$$

- ϵ: Smoothing item, aims to prevent division by 0 (default: 10^{-8})

Adam

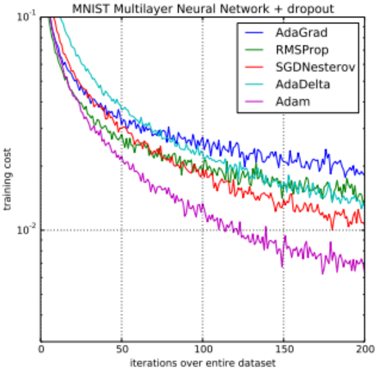


Figure: Source: Kingma and Ba [2014]

Summary

Algorithm	Update rule
Vanilla gradient descent	$g_t = \nabla_\theta J(\theta_t)$ $\Delta\theta_t = -\eta g_t$
Momentum	$\Delta\theta_t = \mu v_t - \eta g_t$
Nesterov Momentum	$\Delta\theta_t = v_{t+1} = \mu v_t - \eta \nabla J_\theta \left(\theta_t + \mu v_t\right)$
AdaGrad	$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$
Adadelta	$\Delta\theta_t = -\frac{\mathbf{RMS}[\Delta\theta]_{t-1}}{\mathbf{RMS}[g]_t} g_t$
RMSprop	$\Delta\theta_t = \frac{\eta}{\mathbf{RMS}[g]_t} \cdot g_t$
Adam	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$

Table: Optimization algorithms summary

Optimizer selection

Which algorithm should we choose?

Optimizer selection

Which algorithm should we choose?

- No consensus on this point
- Seems to be heavily reliant on the practitioner's familiarity with the algorithm

Optimizer selection: Suggestions

- The most popular optimization algorithms actively in use:
  - SGD
  - SGD with momentum
  - RMSprop
  - RMSprop with momentum
  - Adadelta
  - Adam
- Adaptive learning rate methods (AdaGrad, Adadelta, RMSprop, Adam) have fairly robust performance
  - Adam is slightly better than RMSprop
- Input data is sparse $\rightarrow$ **Adaptive learning rate methods**
- Care about fast convergence and train a deep or complex neural network $\rightarrow$ Prefer **adaptive learning rate methods**

Thanks for your attention!

Reference I

Sue Becker, Yann Le Cun, et al. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

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.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

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

Reference II

Andrew Ng. Cs229 lecture notes. *CS229 Lecture notes*, 1(1):1–3, 2000.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

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

Appendix

Table of Contents

Exponentially Weighted Moving Average

Exponentially Weighted Moving Average (EWMA)

$$S_t = \begin{cases} 0 & t = 0 \\ \beta S_{t-1} + (1-\beta) Y_t & t > 0 \end{cases} \tag{24}$$

- $\beta \in [0, 1)$: Degree of weighting decrease
- Y_t: Real measurement value
- S_t: Weighted average value

Exponentially Weighted Moving Average

Exponentially Weighted Moving Average (EWMA)

$$S_t = \begin{cases} 0 & t = 0 \\ \beta S_{t-1} + (1-\beta) Y_t & t > 0 \end{cases} \tag{24}$$

- $\beta \in [0, 1)$: Degree of weighting decrease
- Y_t: Real measurement value
- S_t: Weighted average value
  - Approximately averaging over about $\frac{1}{1-\beta}$ timestamps values
    - $\beta = 0.9 \equiv 10$ previous timestamps

Exponentially Weighted Moving Average

Exponentially Weighted Moving Average (EWMA)

$$S_t = \begin{cases} 0 & t = 0 \\ \beta S_{t-1} + (1-\beta) Y_t & t > 0 \end{cases} \tag{24}$$

- $\beta \in [0, 1)$: Degree of weighting decrease
- Y_t: Real measurement value
- S_t: Weighted average value
  - Approximately averaging over about $\frac{1}{1-\beta}$ timestamps values
    - $\beta = 0.9 \equiv 10$ previous timestamps
    - $\beta = 0.98 \equiv 50$ previous timestamps

Exponentially Weighted Moving Average

Exponentially Weighted Moving Average (EWMA)

$$S_t = \begin{cases} 0 & t = 0 \\ \beta S_{t-1} + (1-\beta) Y_t & t > 0 \end{cases} \tag{24}$$

- **Smoothing**
  - **Greater** β: Adapts more **slowly** to changes $\rightarrow$ **smoother**
  - **Smaller** β: Adapts more **quickly** to changes $\rightarrow$ **noiser, more outliners**

Exponentially Weighted Moving Average

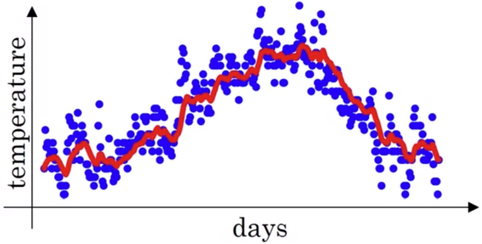


Figure: Source: Deep Learning Specialization

- $\beta = 0.9$

Exponentially Weighted Moving Average

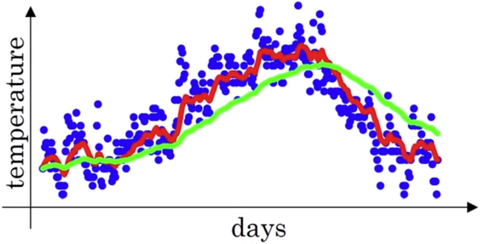


Figure: Source: Deep Learning Specialization

- $\beta = 0.9$
- $\beta = 0.98$

Exponentially Weighted Moving Average

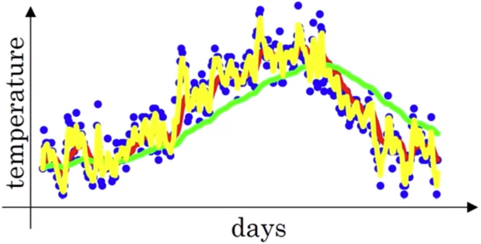


Figure: Source: Deep Learning Specialization

- $\beta = 0.9$
- $\beta = 0.98$
- $\beta = 0.5$

Exponentially Weighted Moving Average

- Bias towards 0 at the beginning

Exponentially Weighted Moving Average

- Bias towards 0 at the beginning
- Bias correction:

Exponentially Weighted Moving Average

- Bias towards 0 at the beginning
- Bias correction:

Exponentially Weighted Moving Average (EWMA) with bias correction

$$v_t = \frac{\beta v_{t-1} + (1-\beta)\theta_t}{1-\beta^t} \tag{25}$$

Table of Contents

Problem of Units

- At each update, we apply $\Delta\theta$ to θ

Problem of Units

- At each update, we apply $\Delta\theta$ to θ
- Unit of θ and $\Delta\theta$ should match

Problem of Units

- At each update, we apply $\Delta\theta$ to θ
- Unit of θ and $\Delta\theta$ should match
- The units in first-order methods relate to the gradient, not the parameter (assuming unitless $J(\theta)$):

$$\text{units of } \Delta\theta \propto \text{units of } g \propto \frac{\partial J}{\partial \theta} \propto \frac{1}{\text{units of } \theta}$$

Problem of Units

- At each update, we apply $\Delta\theta$ to θ
- Unit of θ and $\Delta\theta$ should match
- The units in first-order methods relate to the gradient, not the parameter (assuming unitless $J(\theta)$):

$$\text{units of } \Delta\theta \propto \text{units of } g \propto \frac{\partial J}{\partial \theta} \propto \frac{1}{\text{units of } \theta}$$

- Second-order methods have the correct units for the parameter updates:

$$\text{units of } \Delta\theta \propto \mathbf{H}^{-1} g \propto \frac{\frac{\partial J}{\partial \theta}}{\frac{\partial^2 J}{\partial \theta^2}} \propto \text{units of } \theta$$

Adadelta: Ensures Correct Units

- Recall: After accumulating over window, we have Equation (10):

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

Adadelta: Ensures Correct Units

- Recall: After accumulating over window, we have Equation (10):

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

- In order to maintain the correct units, we need to replace η with a quantity proportional to $\Delta\theta_t$

Adadelta: Ensures Correct Units

- Recall: After accumulating over window, we have Equation (10):

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

- In order to maintain the correct units, we need to replace η with a quantity proportional to $\Delta\theta_t$
- Estimate $\Delta\theta_t$ with $\mathbf{RMS}[\Delta\theta]_{t-1}$ (assuming locally smooth curvature)

Adadelta: Ensures Correct Units

- Recall: After accumulating over window, we have Equation (10):

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

- In order to maintain the correct units, we need to replace η with a quantity proportional to $\Delta\theta_t$
- Estimate $\Delta\theta_t$ with $\mathbf{RMS}[\Delta\theta]_{t-1}$ (assuming locally smooth curvature)
- Update rule of Adadelta:

$$\theta_{t+1} = \theta_t - \frac{\mathbf{RMS}[\Delta\theta]_{t-1}}{\mathbf{RMS}[g]_t} g_t$$

Table of Contents

Newton's Method for Optimization

- Univariate:

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)} \tag{26}$$

Newton's Method for Optimization

- Univariate:

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)} \tag{26}$$

- Multivariate:

$$x_{t+1} = x_t - [\mathbf{H}(f(x_t))]^{-1} \nabla f(x_t) \tag{27}$$

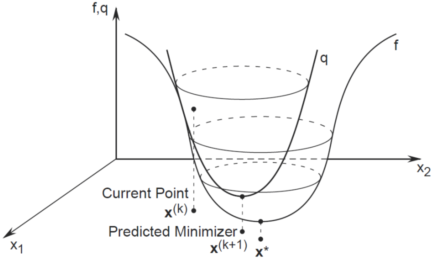


Figure: Source: Taylor Series approximation, newton's method and optimization

Newton's Method Derivation

- Taylor series approximation

$$f(x_{t+1}) = f(x_t + \Delta x) \approx f\left(x_t\right) + f'\left(x_t\right)\Delta x + \frac{1}{2}f''\left(x_t\right)\Delta x^2$$

Newton's Method Derivation

- Taylor series approximation

$$f(x_{t+1}) = f(x_t + \Delta x) \approx f(x_t) + f'(x_t)\,\Delta x + \frac{1}{2} f''(x_t)\,\Delta x^2$$

- Find Δx such that $(x_p + \Delta x)$ is the solution to minimizing the equation

Newton's Method Derivation

- Taylor series approximation

$$f(x_{t+1}) = f(x_t + \Delta x) \approx f(x_t) + f'(x_t)\,\Delta x + \frac{1}{2} f''(x_t)\,\Delta x^2$$

- Find Δx such that $(x_p + \Delta x)$ is the solution to minimizing the equation

$$\frac{d}{d\Delta x}\left(f(x_t) + f'(x_t)\,\Delta x + \frac{1}{2} f''(x_t)\,\Delta x^2 \right) \stackrel{!}{=} 0$$

$$f'(x) + f''(x_t)\,\Delta x \stackrel{!}{=} 0$$

$$\Delta x = -\frac{f'(x_t)}{f''(x_t)}$$

Newton's Method Derivation

- Taylor series approximation

$$f(x_{t+1}) = f(x_t + \Delta x) \approx f\left(x_t\right) + f'\left(x_t\right)\Delta x + \frac{1}{2}f''\left(x_t\right)\Delta x^2$$

- Find Δx such that $(x_p + \Delta x)$ is the solution to minimizing the equation

$$\frac{d}{d\Delta x}\left(f\left(x_t\right) + f'\left(x_t\right)\Delta x + \frac{1}{2}f''\left(x_t\right)\Delta x^2\right) \stackrel{!}{=} 0$$

$$f'\left(x\right) + f''\left(x_t\right)\Delta x \stackrel{!}{=} 0$$

$$\Delta x = -\frac{f'\left(x_t\right)}{f''\left(x_t\right)}$$

Multivariate:

$$\Delta x = -\left[\mathbf{H}(f\left(x_t\right))\right]^{-1}\nabla f\left(x_t\right)$$

Table of Contents

Sparse Data Examples

$x_{t,1}$	$x_{t,2}$	$x_{t,3}$	y_t
1	0	0	1
0.5	0	1	-1
-0.5	1	0	1
0	0	0	-1
0.5	0	0	1
1	0	0	-1
-1	1	0	1
-0.5	0	1	-1

- Frequent, irrelevant
- Infrequent, predictive
- Infrequent, predictive

Sparse Data Examples

Text data:

The most unsung birthday in American business and technological history this year may be the 50th anniversary of the Xerox 914 photocopier.[a]

[a] *The Atlantic*, July/August 2010.

Figure: Source: Machine Learning [2]

[2] `https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/`