Recipe of Gradient Descent

*Make **Grey** Out of **Black***

(Nov 16, 2016)

YANG Jiancheng

Outline

- I. Notes on Gradient Descent
- II. Recipe on SGD Family
  - Overview
  - Momentum
  - Nesterov accelerated gradient (NAG)
  - Separate / Adaptive Learning Rates: Adagrad, RMSprop, Adadelta and Adam
- III. Second Order Method
- IV. A Bag of Tricks

• I. Notes on Gradient Descent

- **Brief Review**

a) Full Batch

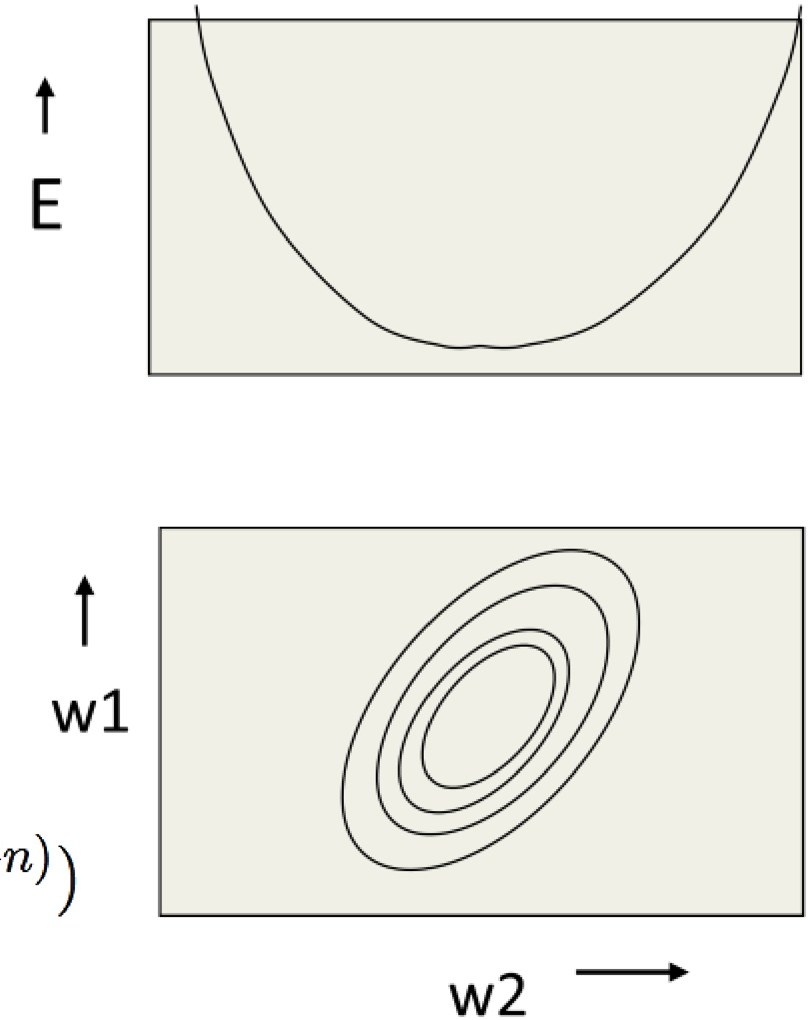
$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

b) SGD (Online)

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

c) Mini Batch SGD

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



Linear Neuron Error Surface

• I. Notes on Gradient Descent

- **Highlights**

a) Key Idea behind Stochastic / Mini Batch
- ☐ *Average the moving*
- ☐ Algorithmically: not only on the batch size!

b) Gradient check (More on Gradient checks)
- ☐ Use the centered formula
- ☐ Compare the relative error
- ☐ Check double float and numerical issue
- ☐ Kinks

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h}$$

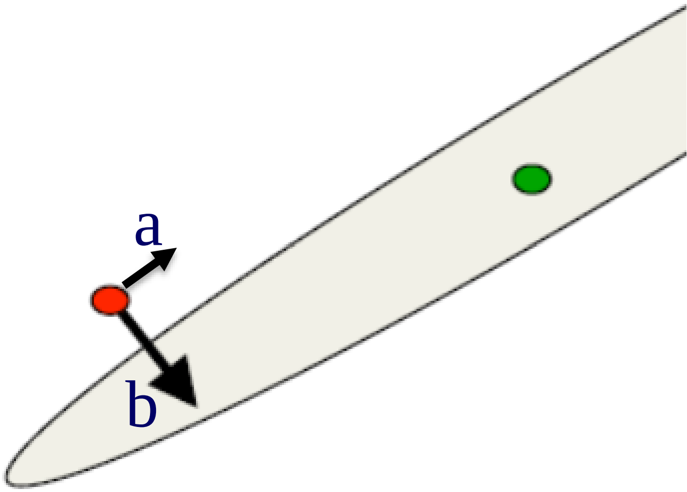
c) Sanity check on the implement:
- ☐ Assume the initial loss
- ☐ Overfit a tiny dataset

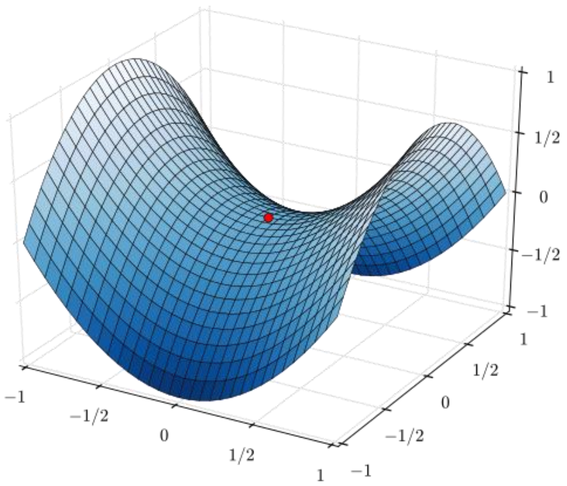
• I. Notes on Gradient Descent

- **Issue**

a) Picky learning rate
b) For global learning rate, hard to:
   - ***move faster*** when the distance is long, while the gradient is ***small***
   - ***slow down*** when the distance is short, while the gradient is ***large***

c) Saddle points

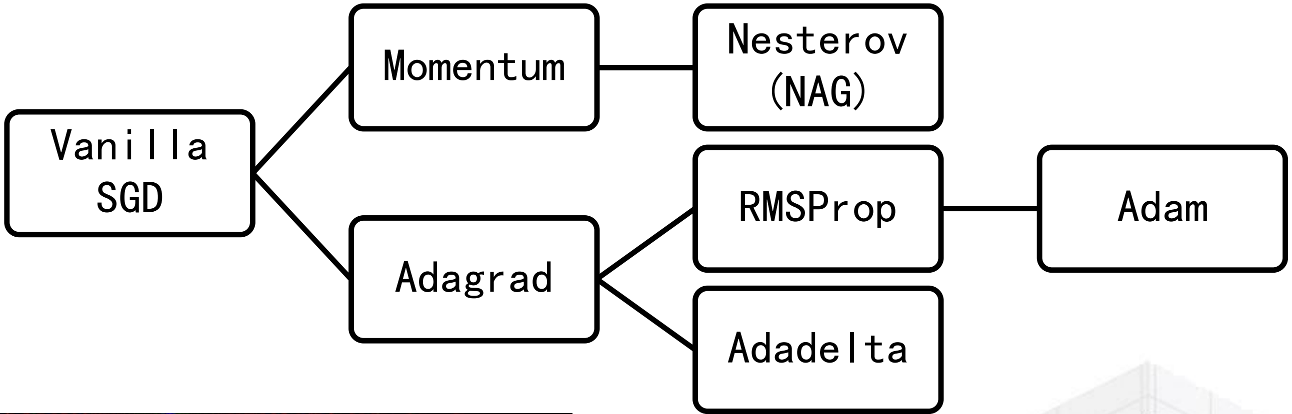


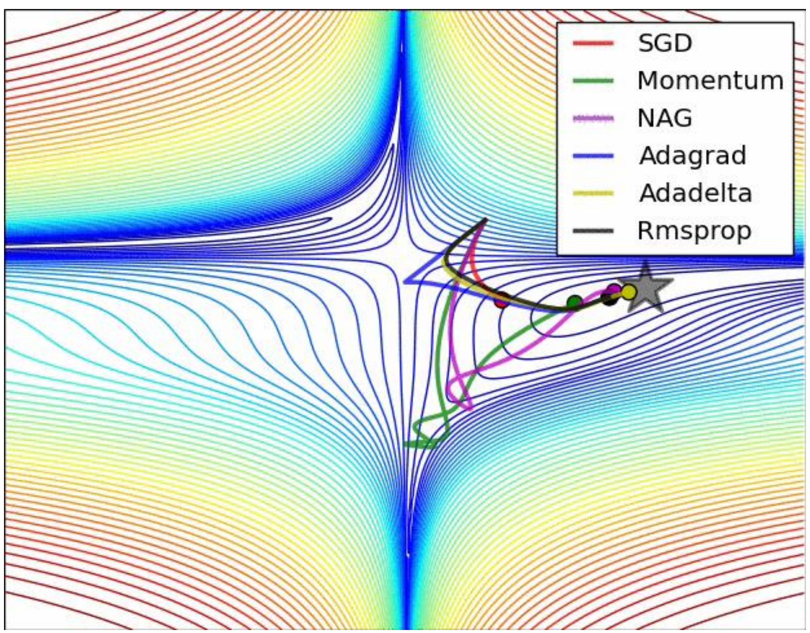


For non-linear neuron, the surface is locally quadratic, with same speed issue.

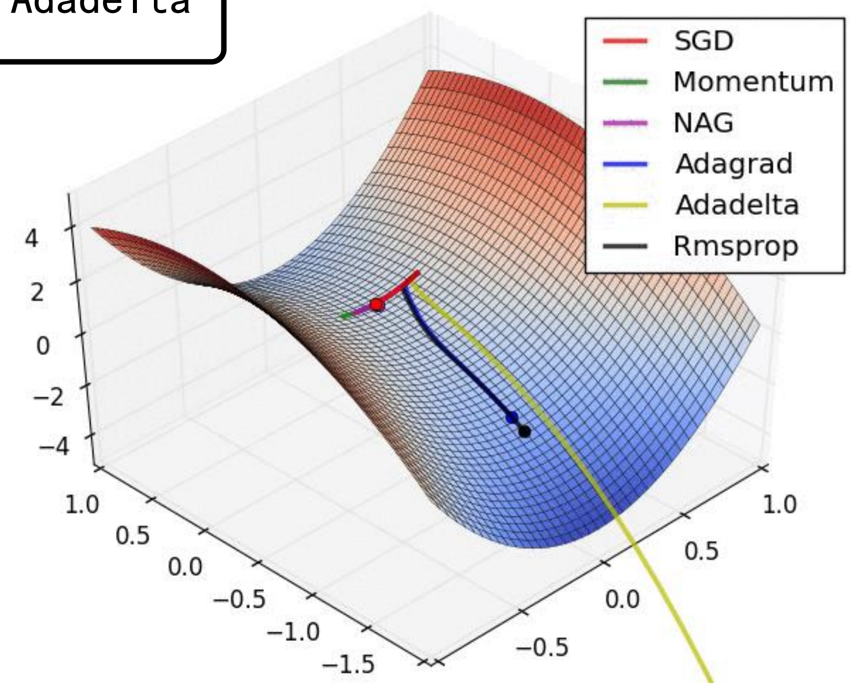
• II. Recipe on SGD Family

- **Overview**





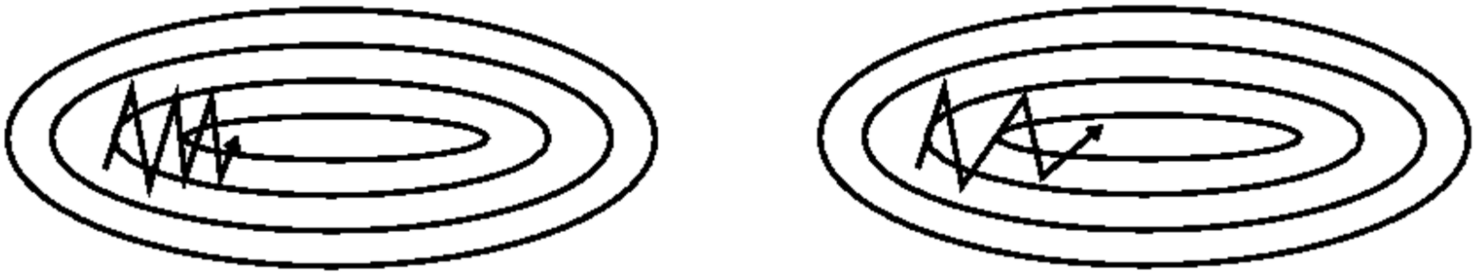
Optimization Speed (contours view)



Optimization Speed (saddle point)

• II. Recipe on SGD Family

- **Momentum**



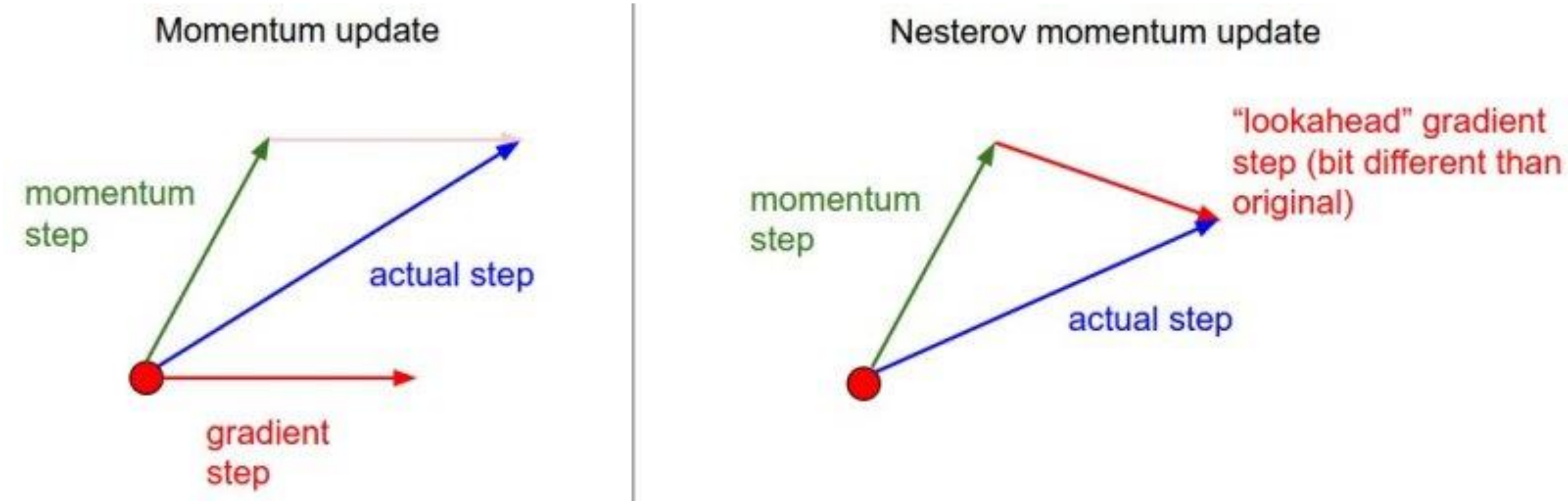
SGD without momentum   SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$

• II. Recipe on SGD Family

- **Nesterov accelerated gradient (NAG)**



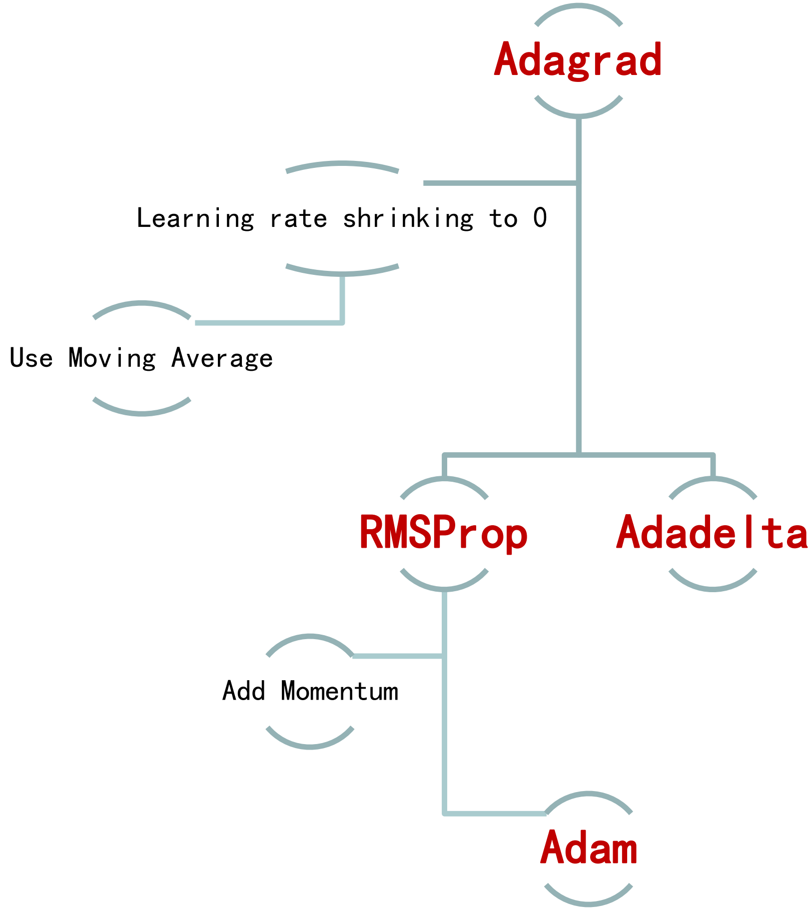
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \qquad\qquad v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

• II. Recipe on SGD Family

- **Separate / Adaptive Learning Rates**

Key idea: Performing larger updates for infrequent and smaller updates for frequent parameters



• II. Recipe on SGD Family

- **Separate / Adaptive Learning Rates: Adagrad**

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}}$$

$$G_t = \sum_{\tau=0}^{t-1} g_\tau^2$$

- Too aggressive and stops learning too early
- Accumulation of the squared gradients increased G_t monotonically
- Learning rate shrinks to infinitesimally small

• II. Recipe on SGD Family

- **Separate / Adaptive Learning Rates: RMSProp**

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

- Proposed by Geoff Hinton in his Coursera class
- Use the decay to simulate the moving windows
- Good choice for Recurrent Neural Network

• II. Recipe on SGD Family

- **Separate / Adaptive Learning Rates: Adadelta**

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon} \qquad E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} \qquad E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma) \Delta\theta_t^2$$

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

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

- No need for a initial learning rate
- Look like RMSProp, developed separately

• II. Recipe on SGD Family

- **Separate / Adaptive Learning Rates: Adam**

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

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

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

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

- Added momentum to gradient
- Use bias correction mechanism

• III. Second Order Method

a) Newton's method

$$x \leftarrow x - [Hf(x)]^{-1} \nabla f(x)$$

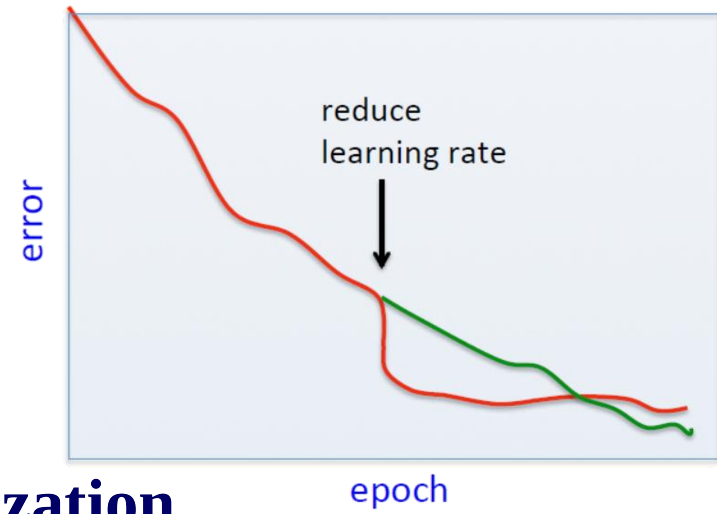
- Hard to get the Hessian

b) L-BFGS

- Computed over the entire training set

c) In practice, it is currently not common to see L-BFGS or similar second-order methods applied to large-scale Deep Learning

• IV. A Bag of Tricks



a) Turn down the learning rate
- A quick win, but slower learning
- Don't too often!

b) Normalization and Batch Normalization

c) Modest Initialization

d) Prefer random search to grid search

e) Shuffling and Curriculum Learning
- Often good to shuffle the training data before every epoch
- Supplying the training data in a meaningful order may actually help

f) Early stopping is beautiful free lunch (Geoff Hinton)

g) Gradient Noise
- More robust to poor initialization

h) Model Ensemble

• IV. A Bag of Tricks

- **Code Tips when Training using Keras**

a) model.fit(shuffle=True) will shuffle the training data, before every epoch

b) Default with shuffle

c) Order between “validation_split” and “shuffle”:

- **Will do split first!**
- If you want to make the validation set different every time, you should keep your own validation set, with “validation_data”

Bibliography

- How to make the learning go faster by Geoffrey Hinton (Neural Network for Machine Learning Week 6)
- An overview of gradient descent optimization algorithms
- CS231n Course Notes on gradient based optimization

Thanks for listening!