Course: Deep Learning

Optimization Algorithms

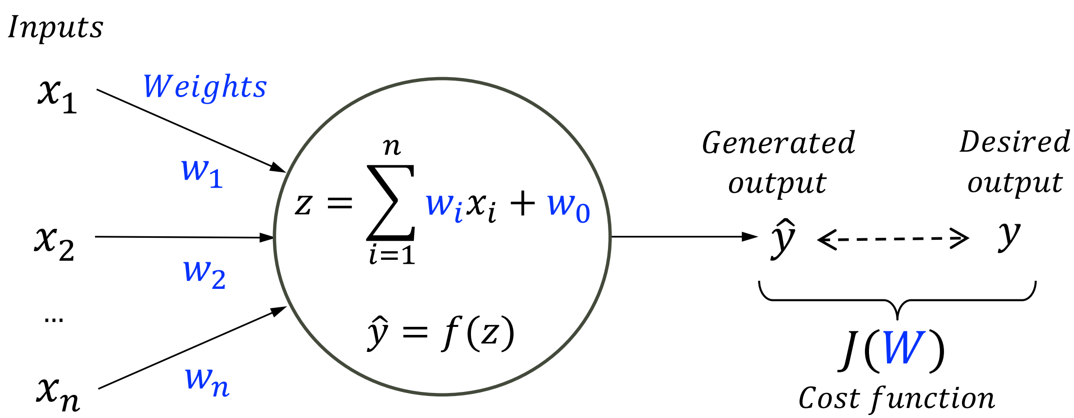
Martin Molina

2025

Department of Artificial Intelligence

Universidad Politécnica de Madrid

Training a neural network as an optimization problem

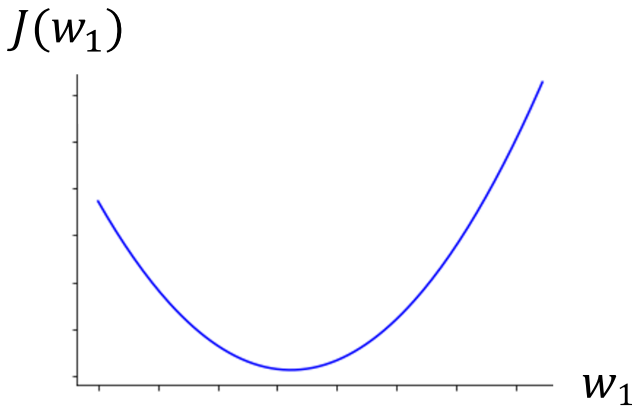


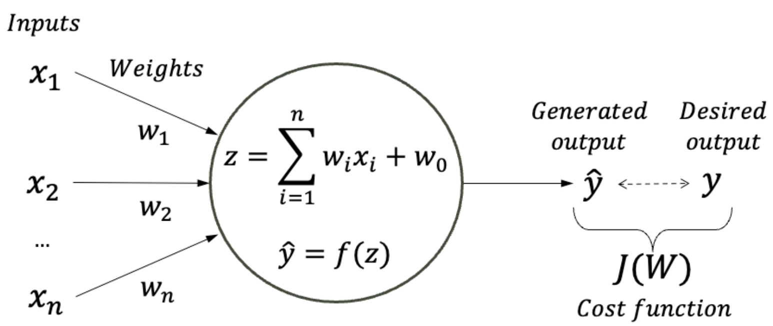
Optimization problem:

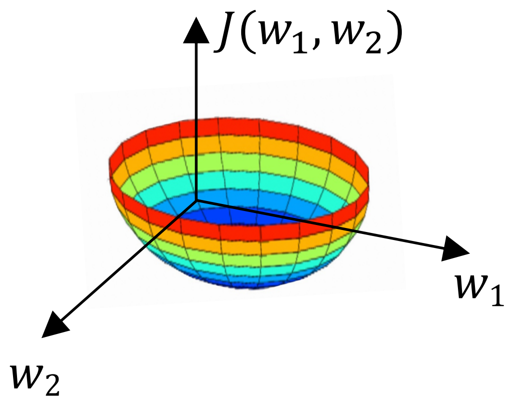
$$\boxed{\arg\min_{W} \; J(W)}$$

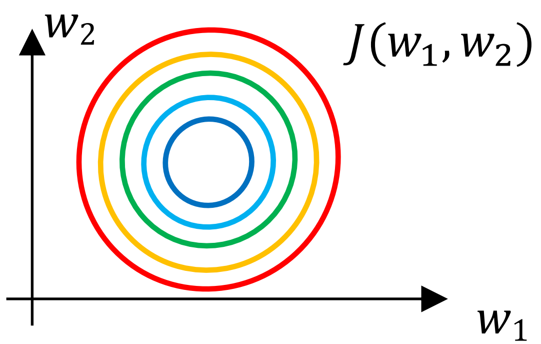
Find the values of W that minimize $J(W)$

Graphical views of the cost function $J(W)$





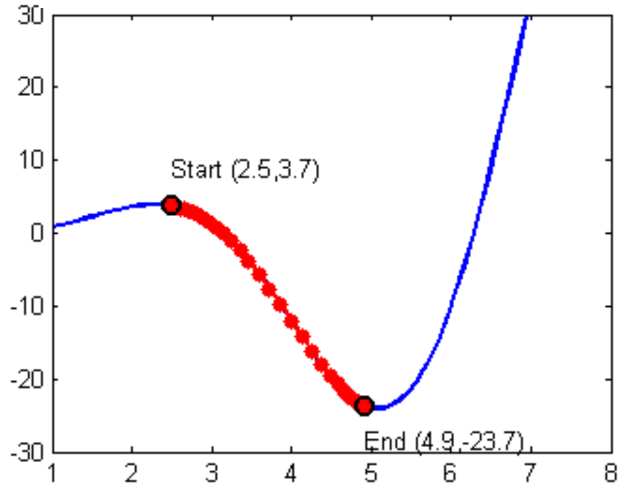




Figures from: Andrew Ng, (2017). Neural Networks and Deep Learning. Coursera.

Graphical view of the optimization process

$J(w_1)$



w_1

Credit image: Adam Harley

Graphical views of the optimization process

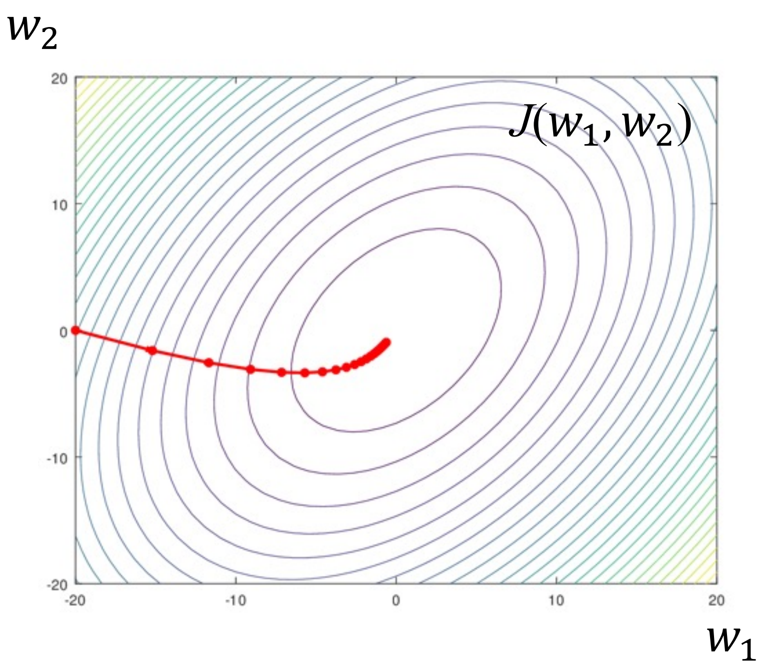


Image source: https://hvidberrrg.github.io

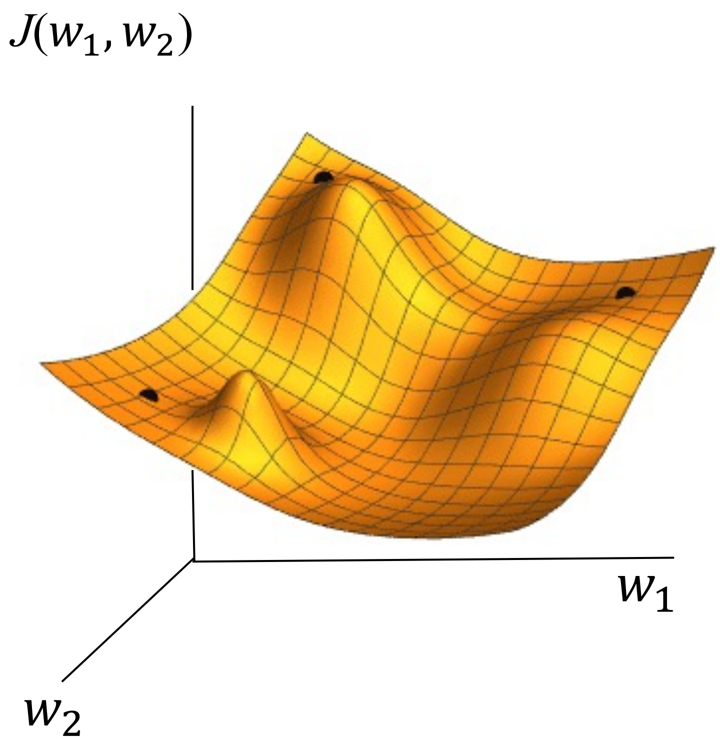
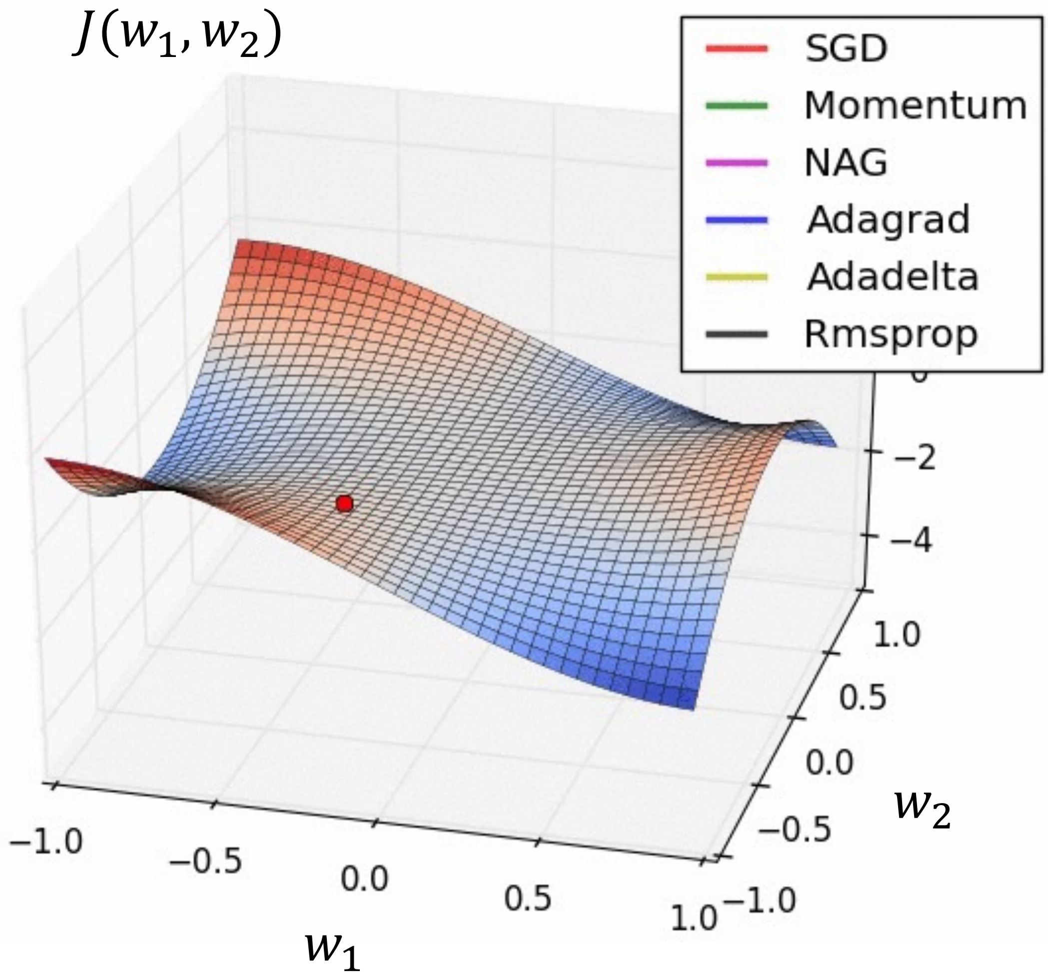


Image source: commons.wikimedia.org

Optimization algorithms



SGD: Stochastic Gradient Descent

Image credit: Alec Radford

Algorithms

- Gradient descent
- Momentum
- RMSprop
- Adam

Algorithms

- **Gradient descent**
- Momentum
- RMSprop
- Adam

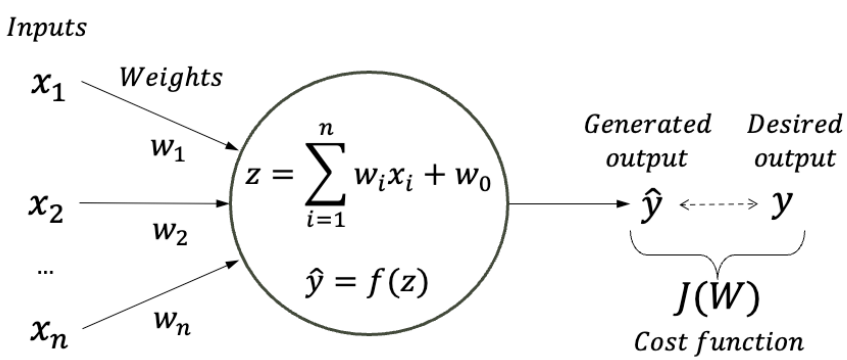
Gradient descent is an iterative process that updates parameters step by step

On iteration k:

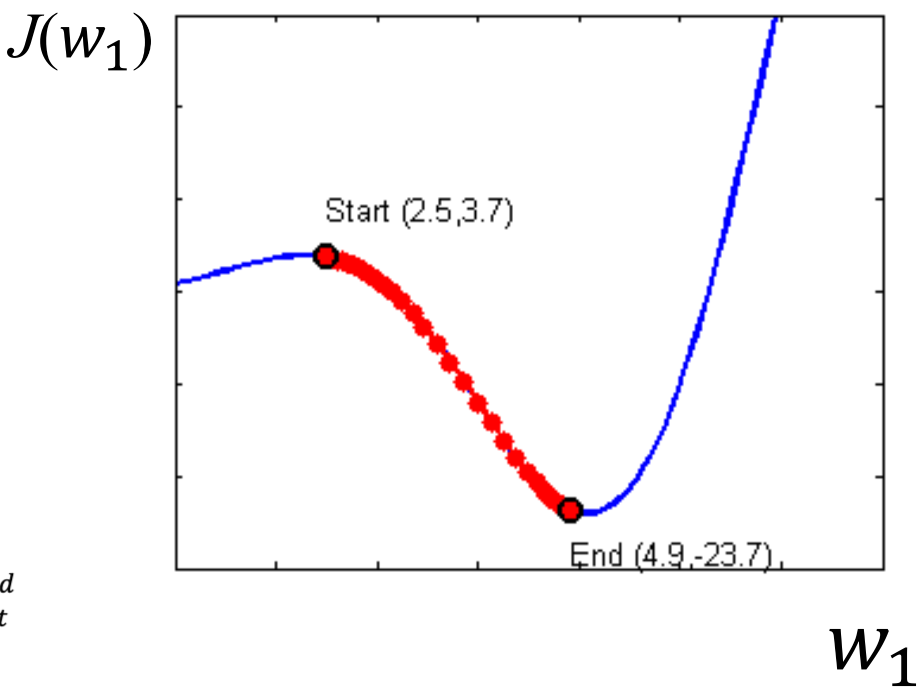
$$W \leftarrow W - \alpha \cdot \partial W$$

α: *Learning rate* ($\alpha = 0.05$)

∂W: *Gradient vector* $\nabla_w J(W)$



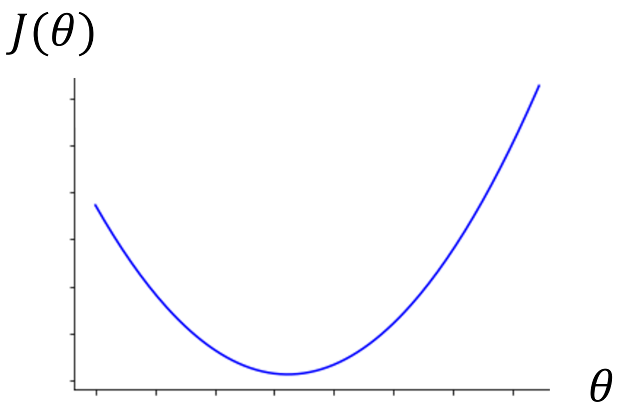
Credit image: Adam Harley



Simplified notation: Parameter $\boldsymbol{\theta}$

On iteration k:

$$\theta \leftarrow \theta - \alpha \cdot \partial\theta$$



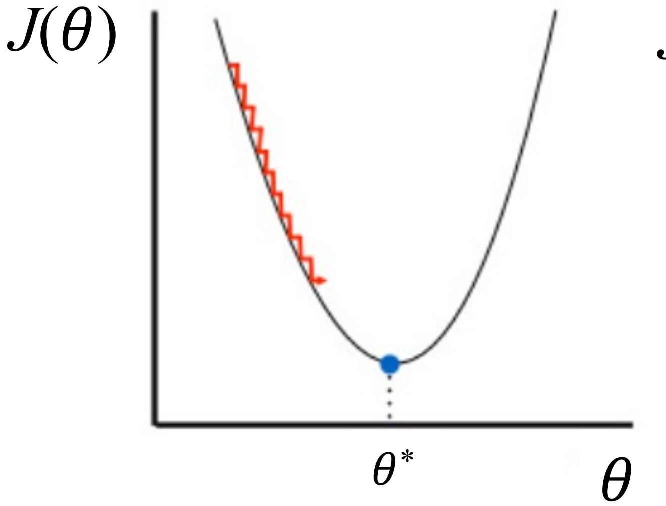
θ: $Parameters\ (weights\ W)$

α: $Learning\ rate\ (\alpha = 0.05)$

$\partial\theta$: $Gradient\ vector\ \ \nabla_\theta J(\theta)$

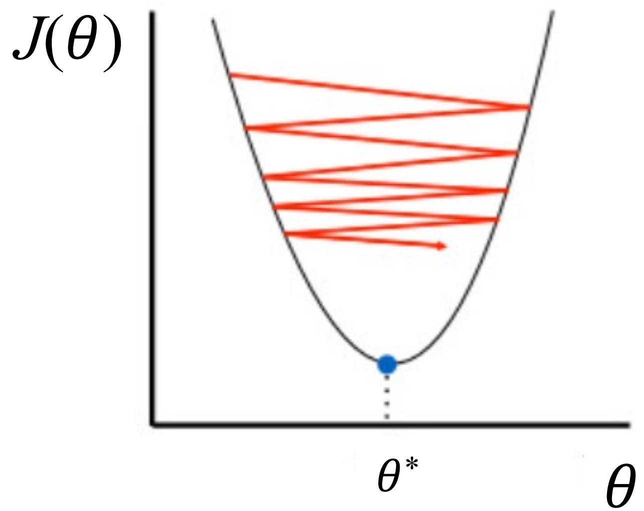
How to choose the value for the learning rate α?

There are two extreme values for learning rates:



Too small α

Slow convergence



Too big α

Divergence or
slow convergence
(overshooting)

Figures from: http://m.blog.csdn.net/article/details?id=50276775

Impact of learning rate on the training process

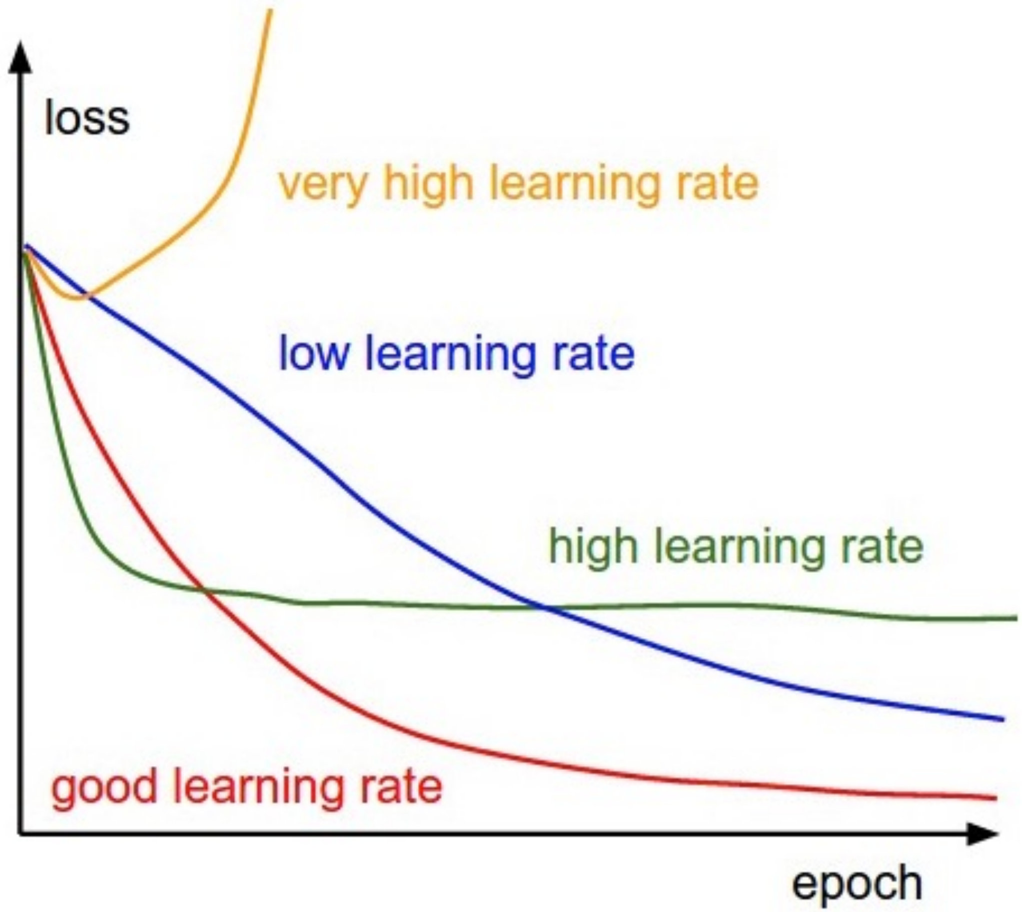
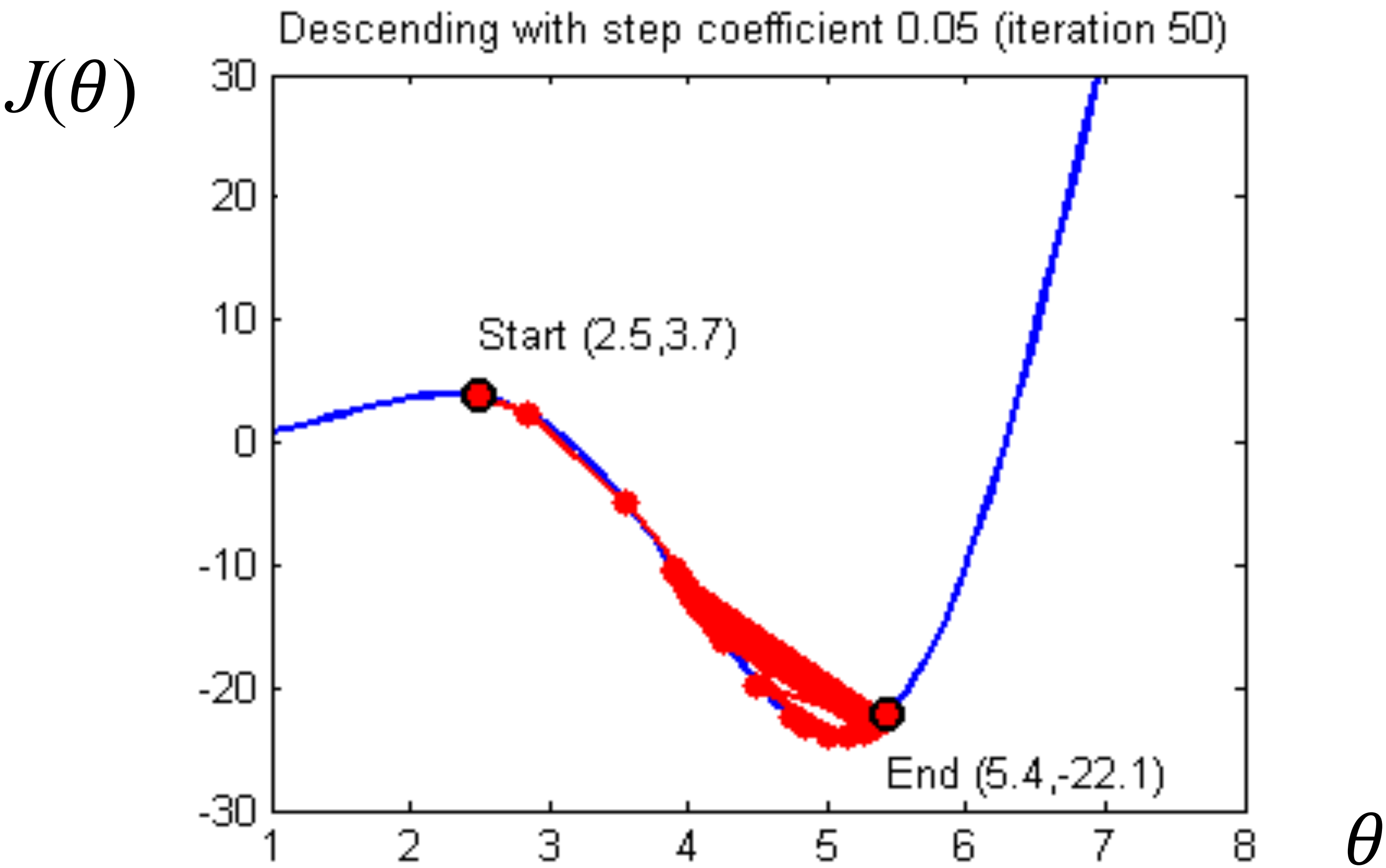


Figure from: Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition

How is the learning rate value programmed in TensorFlow/Keras?

```
model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
              metrics=["categorical_accuracy"])
```

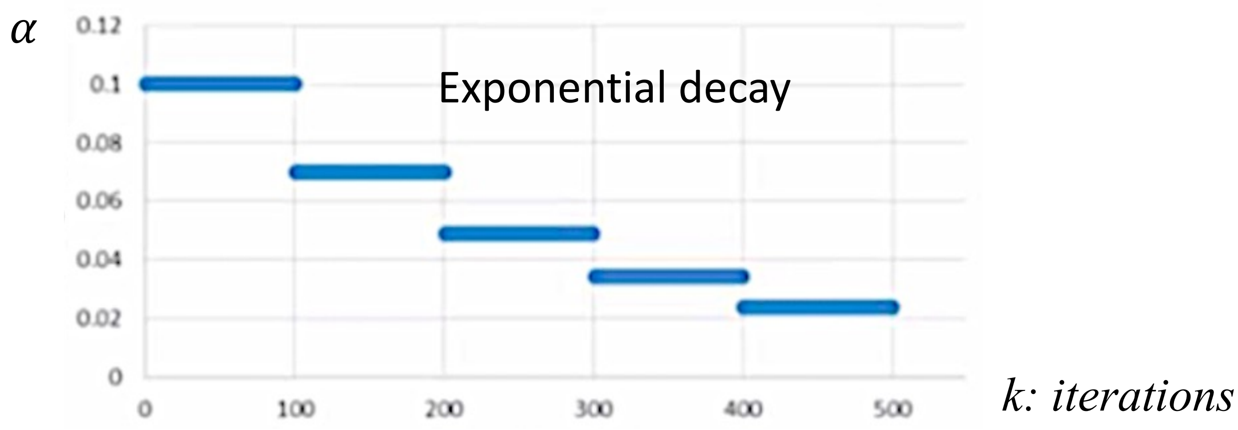
Example



Credit image: Adam Harley

Learning rate decay

Instead of using a constant learning rate, the learning rate decreases as the training progresses



On iteration k :

$$\alpha = \alpha_0 r^{\left\lfloor \frac{k}{n} \right\rfloor}$$

α_0: *initial learning rate*
r: *decay rate*
n: *decay steps*
$\lfloor x \rfloor$: *floor function* $(e.g., \lfloor 3.6 \rfloor = 3)$

Example:

$\alpha_0 = 0.1$
$r = 0.7$
$n = 100$

How is the learning rate decay programmed in TensorFlow/Keras?

```
initial_learning_rate = 0.1

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=10000,
    decay_rate=0.96,
    staircase=True)

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=lr_schedule),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

$$\left.\begin{aligned} &\alpha_0 = 0.1 \ \ (initial\ learning\ rate) \\ &r = 0.96 \ \ (decay\ rate) \\ &n = 10{,}000\ (decay\ steps) \end{aligned}\right\}$$

Decay every 10,000 steps with a rate of 0.96

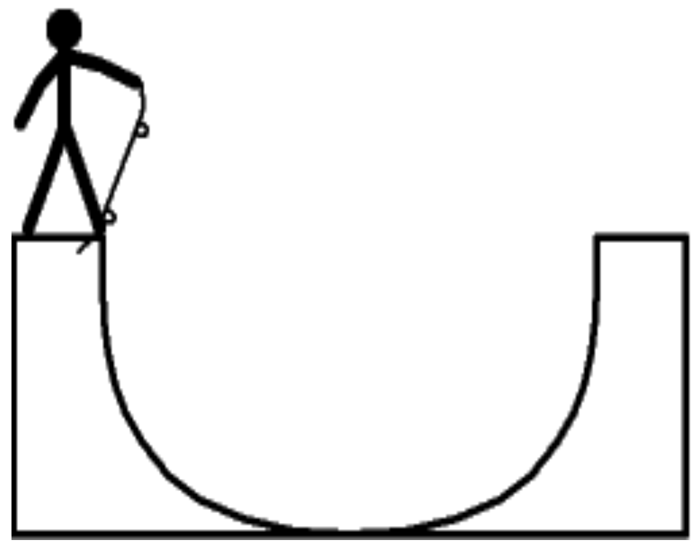
https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/ExponentialDecay

Algorithms

- Gradient descent
- **Momentum**
- RMSprop
- Adam

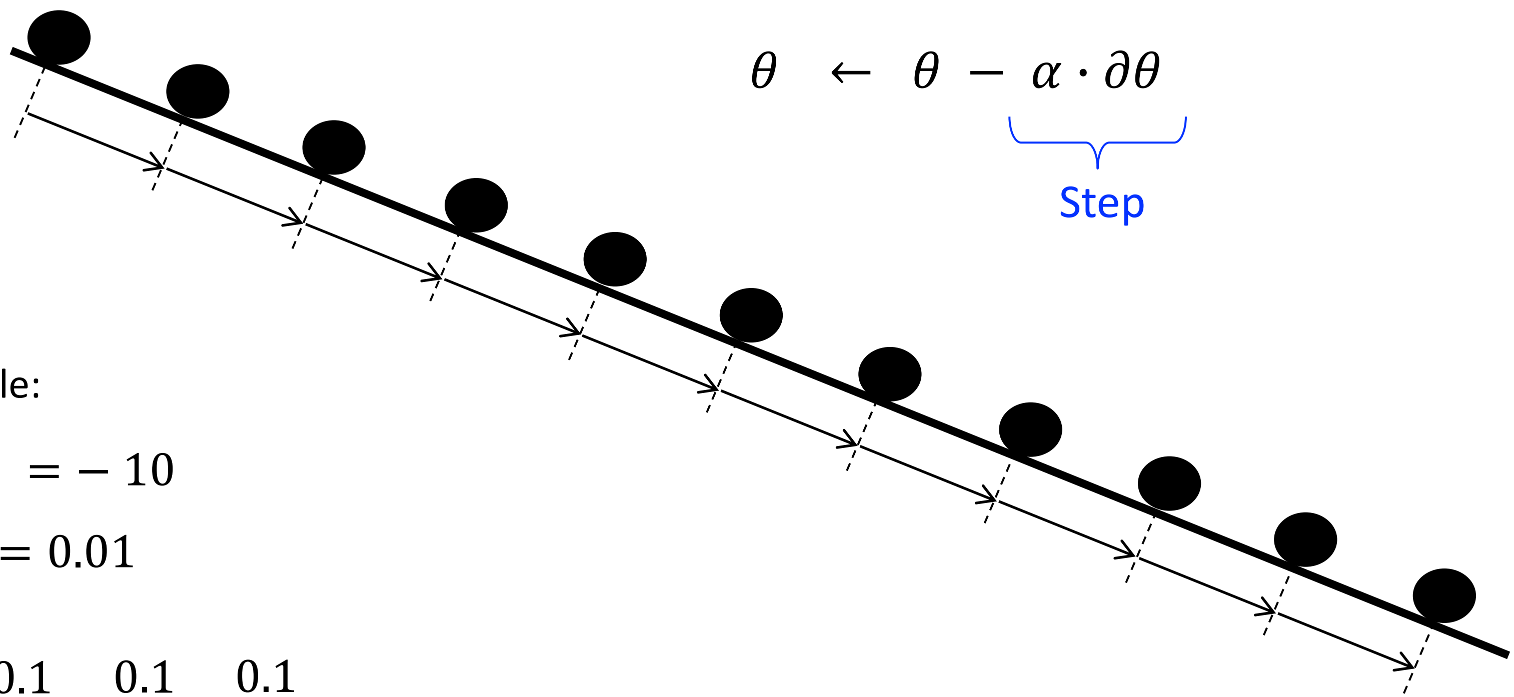
Momentum optimization

- Creates an accelerating motion that can improve convergence



Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1-17.

Steps are constant when the slope is constant (in regular gradient descent algorithm)

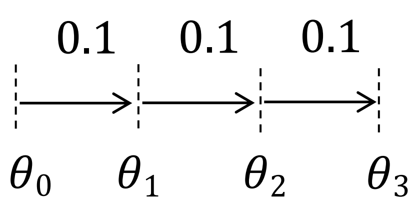


$$\theta \;\leftarrow\; \theta - \underbrace{\alpha \cdot \partial\theta}_{\text{Step}}$$

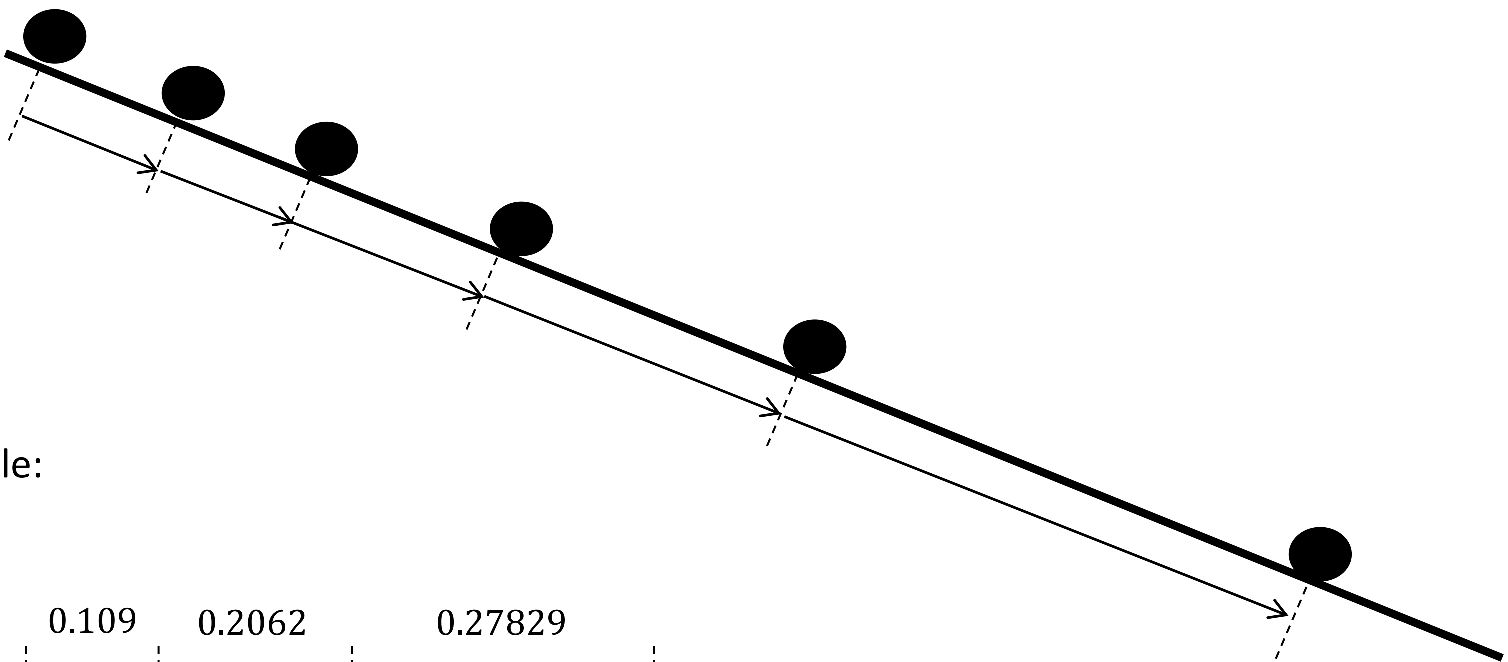
Example:

$$\partial\theta = -10$$

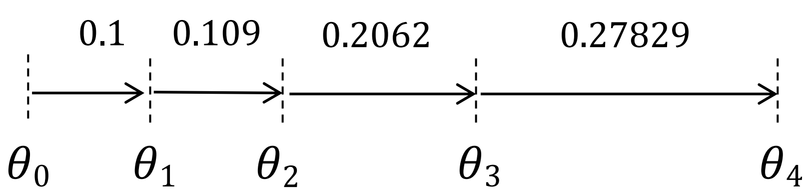
$$\alpha = 0.01$$



Steps are increasing with the momentum optimization



Example:



This could be useful to avoid local minimum values

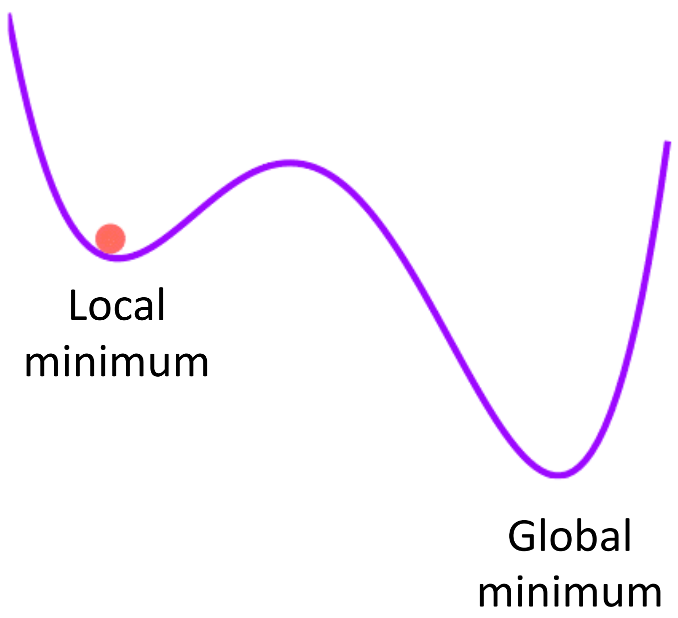


Figure from: http://www.experientiadocet.com/2012_02_01_archive.html

Regular gradient descent

On iteration k:

1. $m \leftarrow \alpha\, \partial\theta$
2. $\theta \leftarrow \theta - m$

On iteration k:

$$\theta_{(k)} = \theta_{(k-1)} - \alpha \cdot \partial\theta_{(k)}$$

Earlier gradients are ignored

$$\partial\theta_{(k-1)}, \partial\theta_{(k-2)}, \partial\theta_{(k-3)}, \dots$$

Momentum optimization

On iteration k:

1. $m \leftarrow \beta\, m + \alpha\, \partial\theta$
2. $\theta \leftarrow \theta - m$

On iteration k:

$$\begin{aligned}\theta_{(k)} = \theta_{(k-1)} - \alpha \cdot \partial\theta_{(k)} &- \\ - \alpha \cdot \beta \cdot \partial\theta_{(k-1)} &- \\ - \alpha \cdot \beta^2 \cdot \partial\theta_{(k-2)} &- \\ - \alpha \cdot \beta^3 \cdot \partial\theta_{(k-3)} &- \dots\end{aligned}$$

Past gradients are accumulated in m

Momentum optimization

On iteration k:

1. $m \leftarrow \beta\, m + \alpha\, \partial\theta$
2. $\theta \leftarrow \theta - m$

Momentum: $\beta = 0.9$

Alternative formulation:

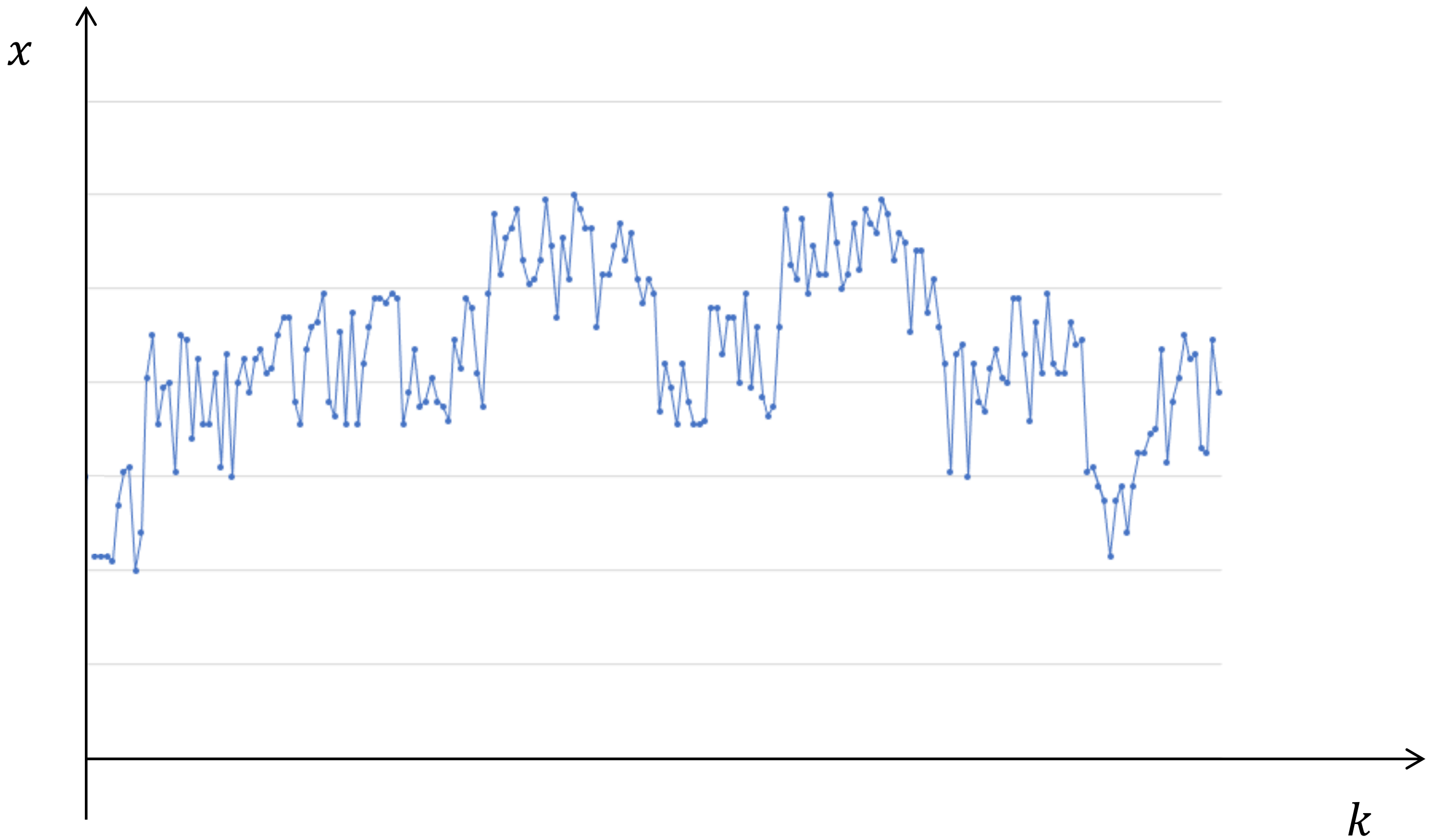
On iteration k:

1. $m \leftarrow \beta\, m + (1 - \beta) \cdot \partial\theta$
2. $\theta \leftarrow \theta - \alpha\, m$

Better separation of the effect of α and β

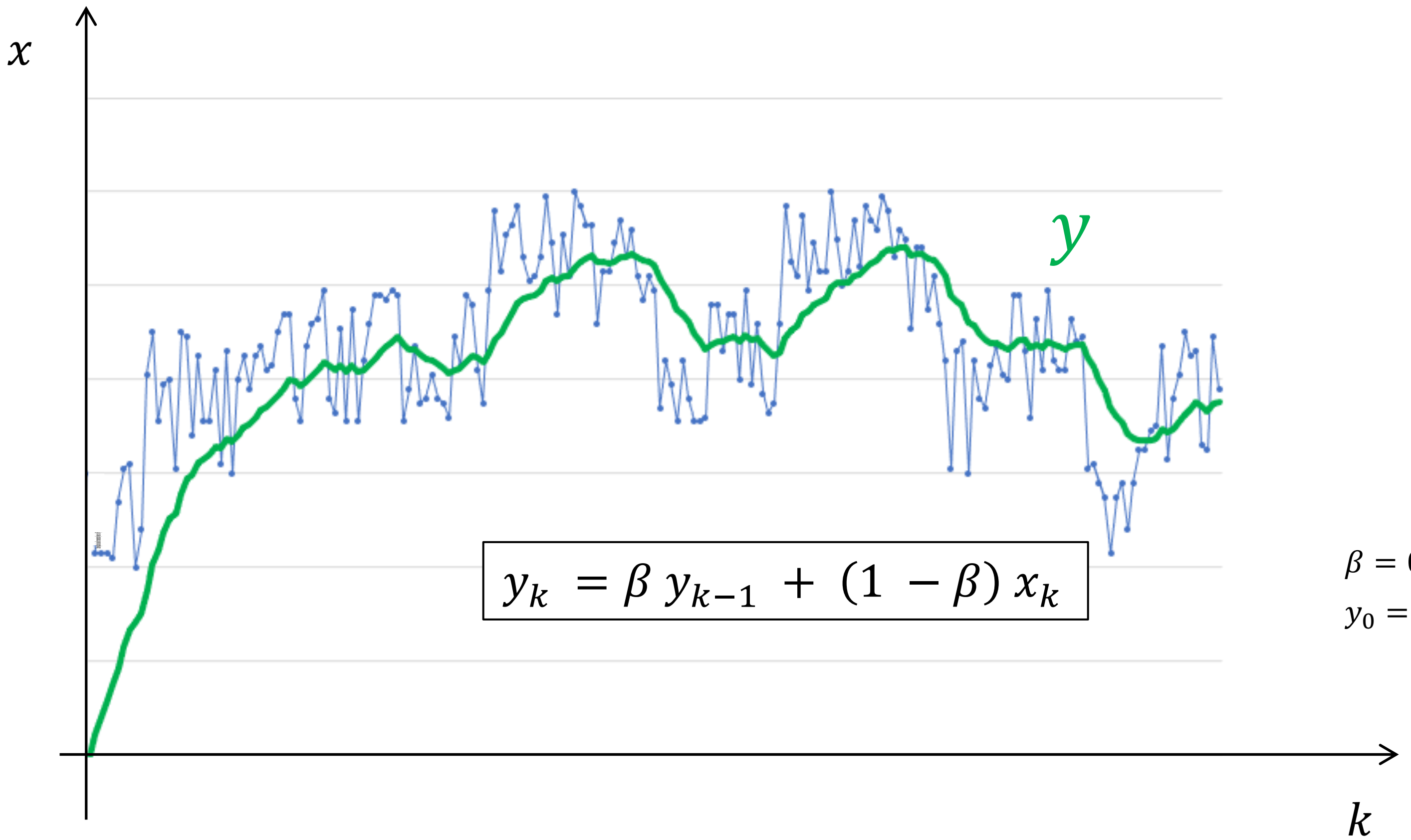
$\longleftarrow$ Exponential weighted average of $\partial\theta$

Example: A sequence of quantitative values



$k = 1, 2, 3, \ldots$

Exponential weighted average



$\beta = 0.9$
$y_0 = 0$

$$y_k = (1-\beta)\,[\,x_k + \beta\, x_{k-1} + \beta^2\, x_{k-2} + \beta^3\, x_{k-3} + \cdots + \beta^{k-1}\, x_1] =$$

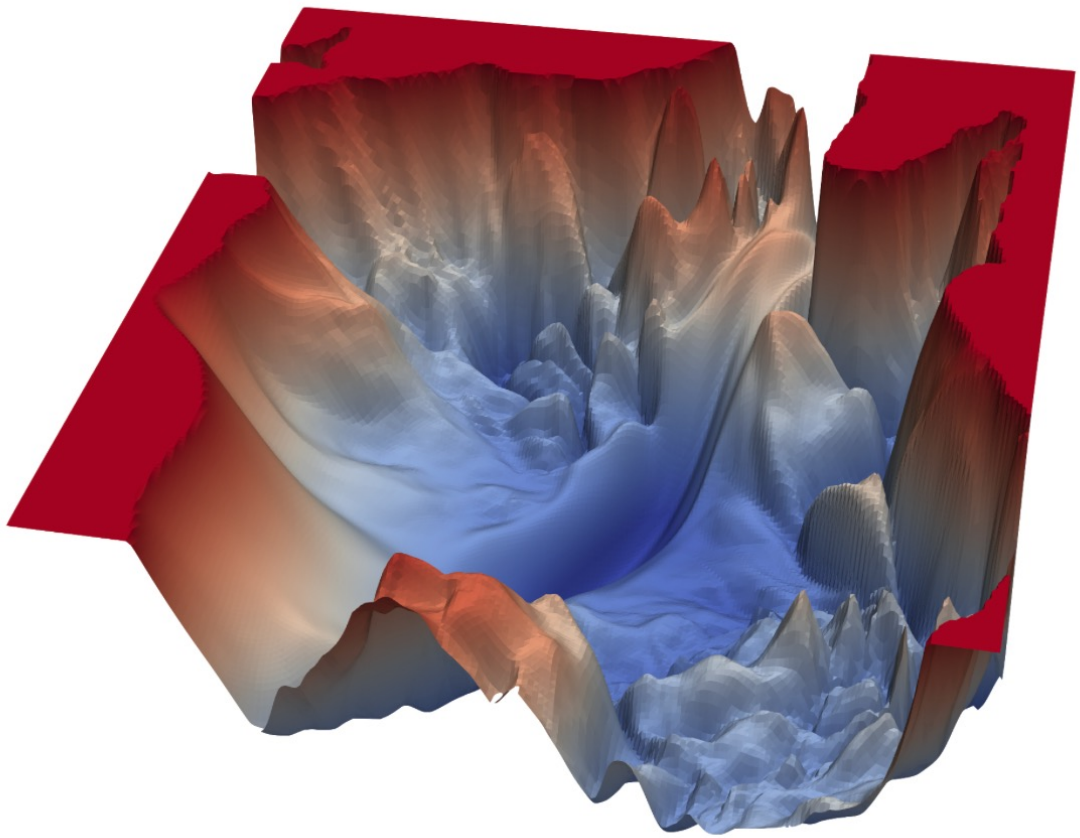
How is the Momentum Optimizer programmed in TensorFlow/Keras?

```
model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9),
              metrics=["categorical_accuracy"])
```

Algorithms

- Gradient descent
- Momentum
- **RMSprop**
- Adam

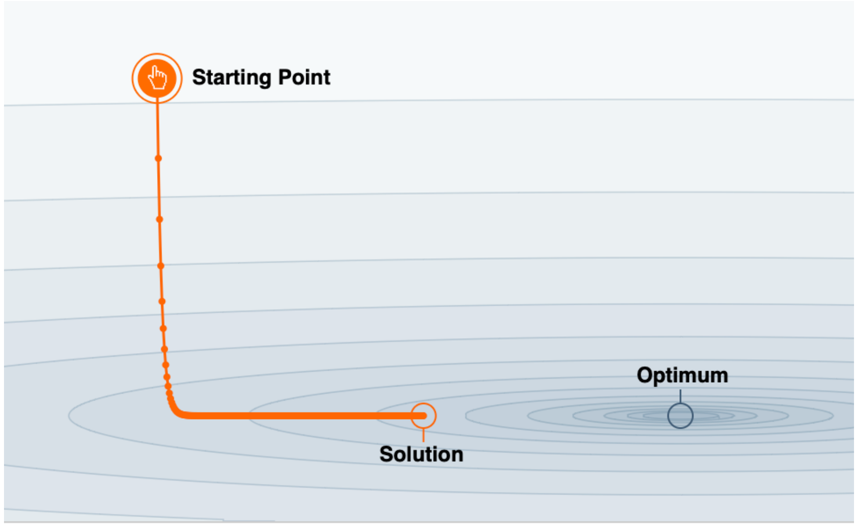
Loss landscapes may present irregular surfaces with different slopes



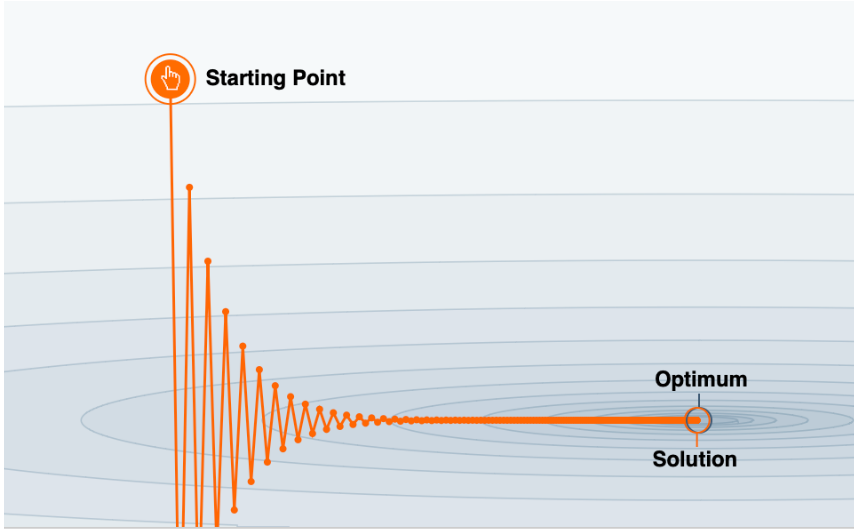
Source of image: https://www.cs.umd.edu/~tomg/projects/landscapes/

A learning rate value may be appropriate for some directions but inappropriate for others

Small learning rate



Bigger learning rate



Idea

Using an adaptive learning rate: if the gradient is larger in one direction, reduce proportionally the learning rate in this direction

Source of images: http://fa.bianp.net/teaching/2018/eecs227at/gradient_descent.html

Regular gradient descent

On iteration k: $(\forall \theta_i)$

$$\theta_i \leftarrow \theta_i - \alpha\, \partial\theta_i$$

The learning rate is constant

RMSprop

On iteration k: $(\forall \theta_i)$

1. $s_i \leftarrow \beta s_i + (1 - \beta)(\partial\theta_i)^2$

2. $\theta_i \leftarrow \theta_i - \frac{\alpha}{\sqrt{s_i}}\, \partial\theta_i$

The learning rate is divided by a value to be inversely proportional to the last gradients

This effect is called “adaptive learning rate”

Hinton, G. et al. (2012). RMSprop: Divide the gradient by a running average of its recent magnitude. Coursera lecture: https://homl.info/58

Regular gradient descent

On iteration k: $(\forall \theta_i)$

$$\theta_i \leftarrow \theta_i - \underbrace{\alpha \, \partial\theta_i}_{\text{Size of step}}$$

RMSprop

On iteration k: $(\forall \theta_i)$

s_i accumulates the square of the gradients (using exponential weighted average)

1. $s_i \leftarrow \beta s_i + (1 - \beta)(\partial\theta_i)^2$

s_i is larger when $\partial\theta_i$ is larger

(i.e., when the cost function has steeper slope in this dimension)

2. $\theta_i \leftarrow \theta_i - \underbrace{\frac{\alpha}{\sqrt{s_i}}}_{\text{Size of step is smaller when } s_i \text{ is larger}} \partial\theta_i$

RMSprop

RMSprop: Root Mean Square Propagation

On iteration k:

1. $s_i \leftarrow \beta s_i + (1 - \beta)(\partial\theta_i)^2 \quad \forall s_i$
2. $\theta_i \leftarrow \theta_i - \frac{\alpha}{\sqrt{s_i + \varepsilon}} \partial\theta_i \quad \forall \theta_i$

$\beta = 0.9$

$\varepsilon = 10^{-7}$

Alternative formulation:

On iteration k:

1. $s \leftarrow \beta\, s + (1 - \beta)\partial\theta \odot \partial\theta$
2. $\theta \leftarrow \theta - \alpha \cdot \partial\theta \oslash \sqrt{s + \varepsilon}$

$\odot$ element-wise multiplication

$\oslash$ element−wise division

How is RMSprop programmed in TensorFlow/Keras?

```
model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9),
              metrics=["categorical_accuracy"])
```

$$\left.\begin{aligned} \beta &= 0.9 \\ \varepsilon &= 10^{-7} \end{aligned}\right\} \text{Default values}$$

Algorithms

- Gradient descent
- Momentum
- RMSprop
- **Adam**

Adam combines momentum and RMSprop

Adam: Adaptive Moment Estimation

On iteration k:

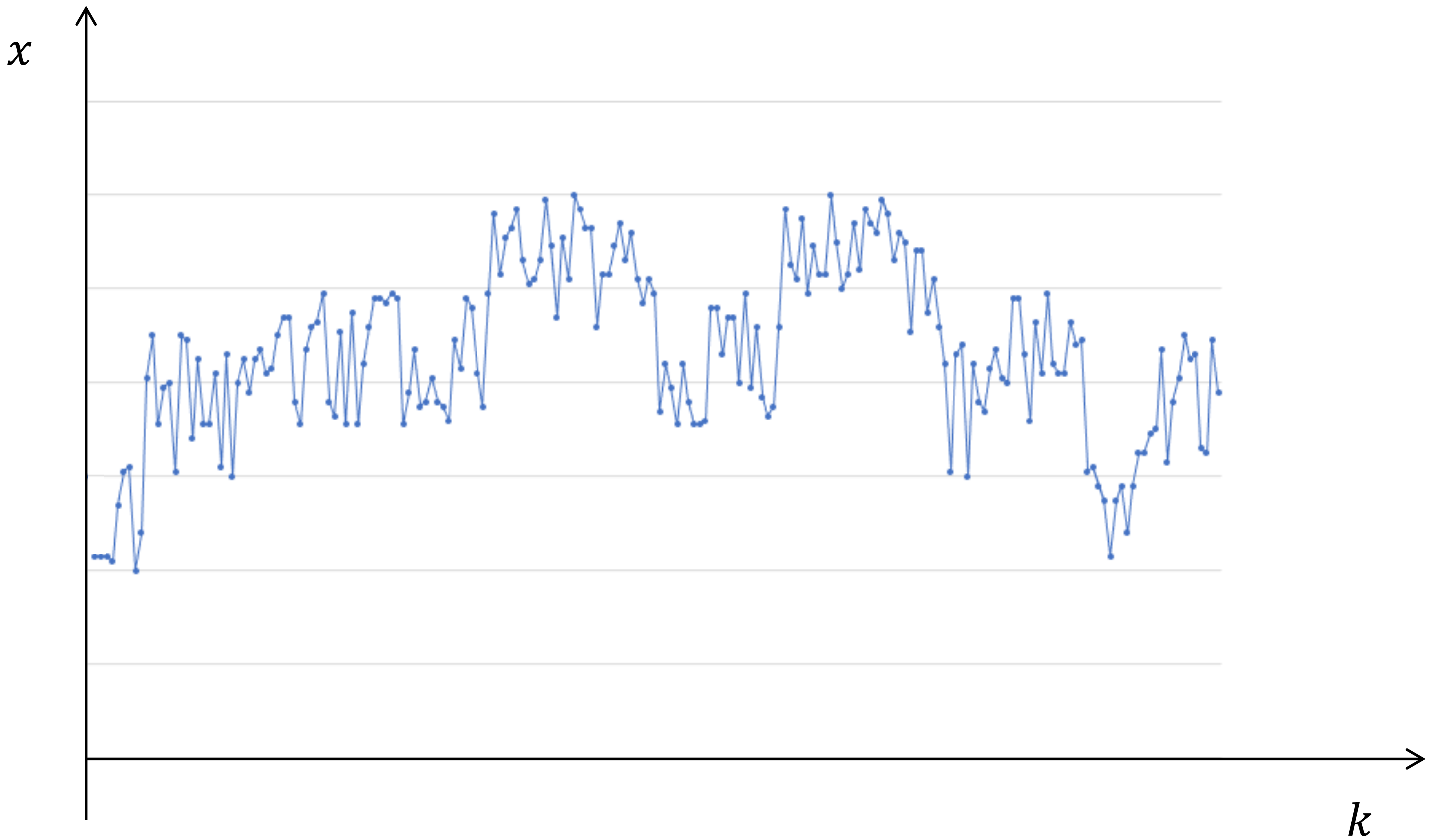
1. $m \leftarrow \beta_1 m + (1 - \beta_1)\, \partial\theta$
2. $s \leftarrow \beta_2 s + (1 - \beta_2)\, \partial\theta \odot \partial\theta$
3. $m' \leftarrow \frac{m}{1-(\beta_1)^k} \qquad s' \leftarrow \frac{s}{1-(\beta_2)^k}$
4. $\theta \leftarrow \theta - \alpha\, m' \oslash \sqrt{s' + \varepsilon}$

Momentum
(exponential weighted average of $\partial\theta$)

RMSProp
(exponential weighted average of $\partial\theta \odot \partial\theta$)

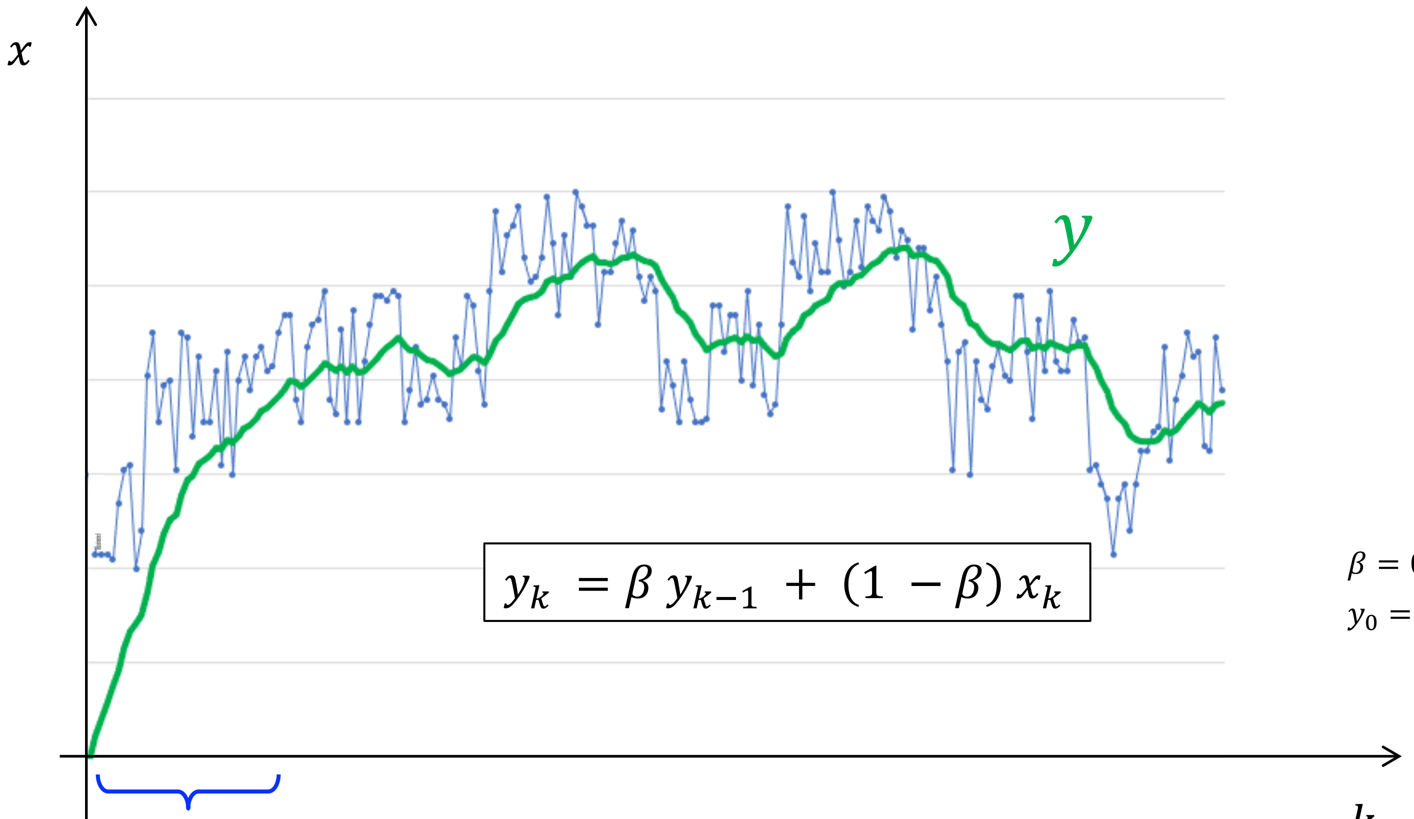
Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Example: A sequence of quantitative values



$k = 1, 2, 3, \ldots$

Exponential weighted average



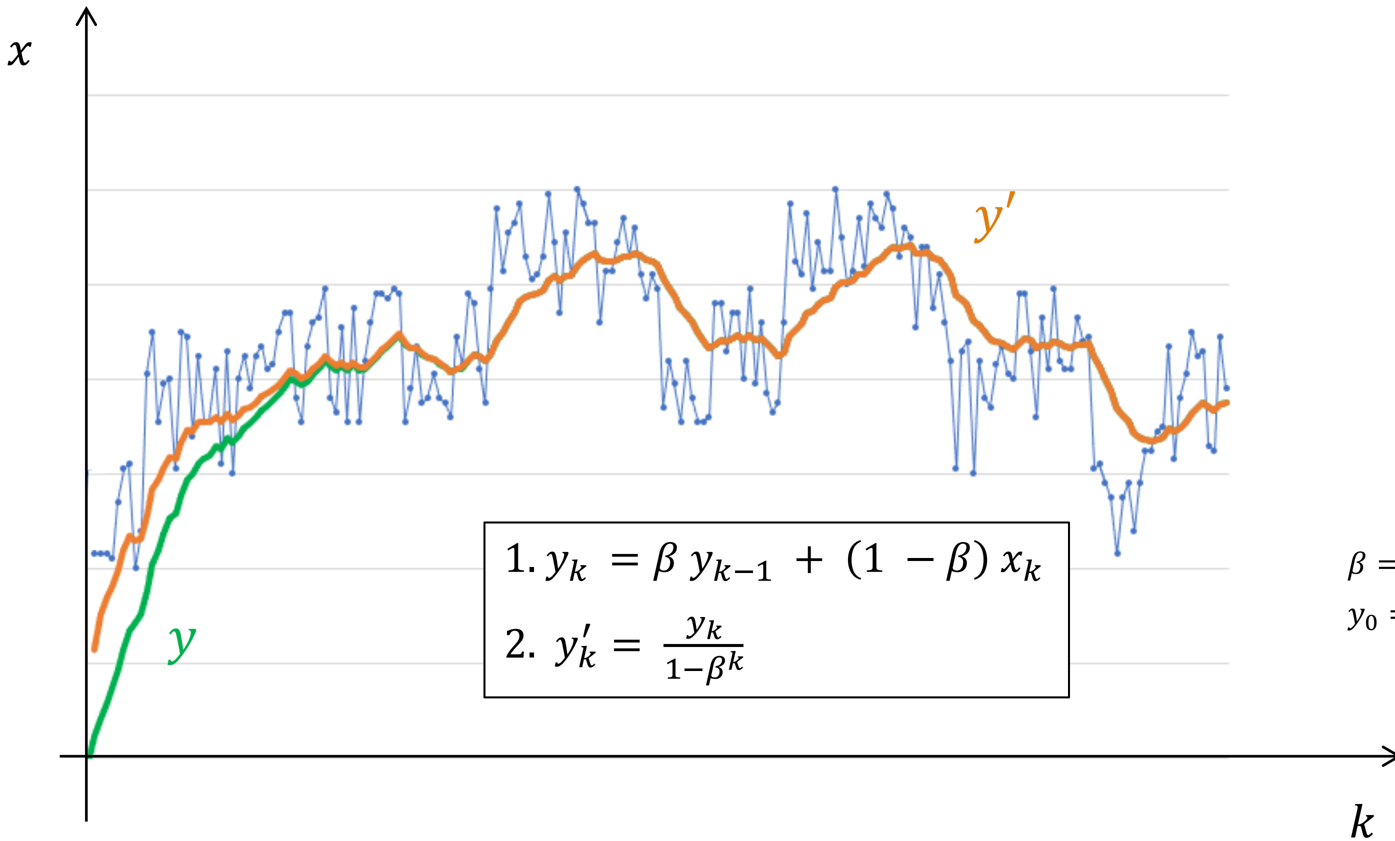
$$y_k = \beta\, y_{k-1} + (1 - \beta)\, x_k$$

$\beta = 0.9$

$y_0 = 0$

Problem: y is biased to zero
at the beginning of the sequence

Correction to avoid values biased to zero at the beginning of the sequence



1. $y_k = \beta\, y_{k-1} + (1-\beta)\, x_k$

2. $y'_k = \frac{y_k}{1-\beta^k}$

$\beta = 0.9$

$y_0 = 0$

Adam combines momentum and RMSprop

Adam: Adaptive Moment Estimation

On iteration k:

1. $m \leftarrow \beta_1 m + (1 - \beta_1)\, \partial\theta$ ← Momentum (exponential weighted average of $\partial\theta$)
2. $s \leftarrow \beta_2 s + (1 - \beta_2)\, \partial\theta \odot \partial\theta$ ← RMSProp (exponential weighted average of $\partial\theta \odot \partial\theta$)
3. $m' \leftarrow \frac{m}{1-(\beta_1)^k}$ $\quad s' \leftarrow \frac{s}{1-(\beta_2)^k}$ ← Bias correction of exponential weighted averages
4. $\theta \leftarrow \theta - \alpha\, m' \oslash \sqrt{s' + \varepsilon}$ ← Integrated expression

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

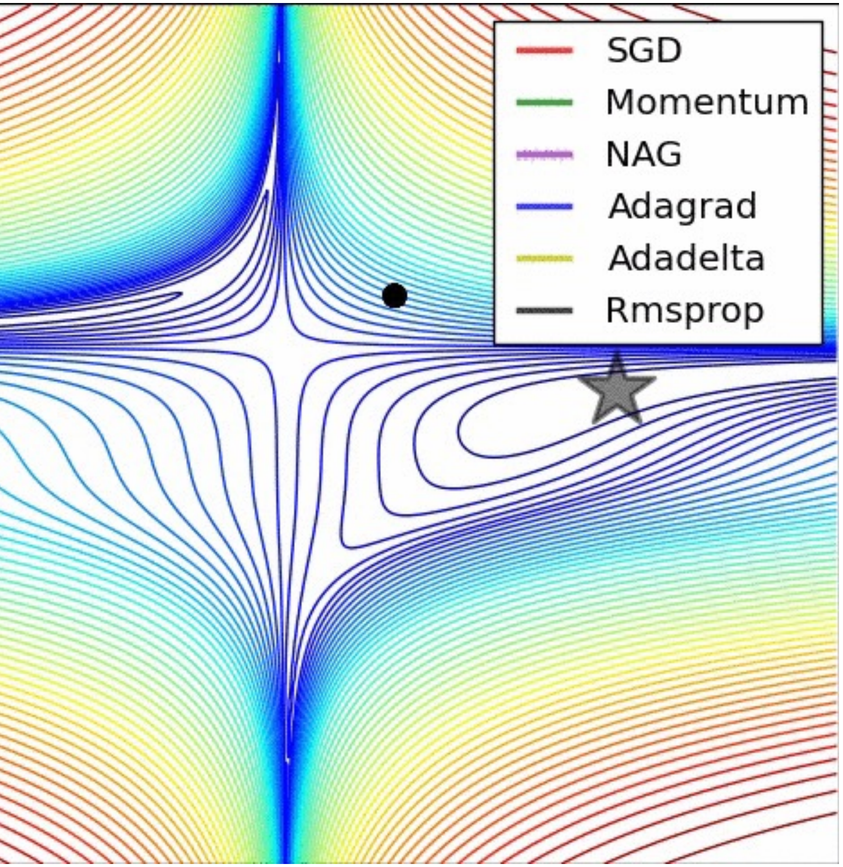
How is Adam programmed in TensorFlow/Keras?

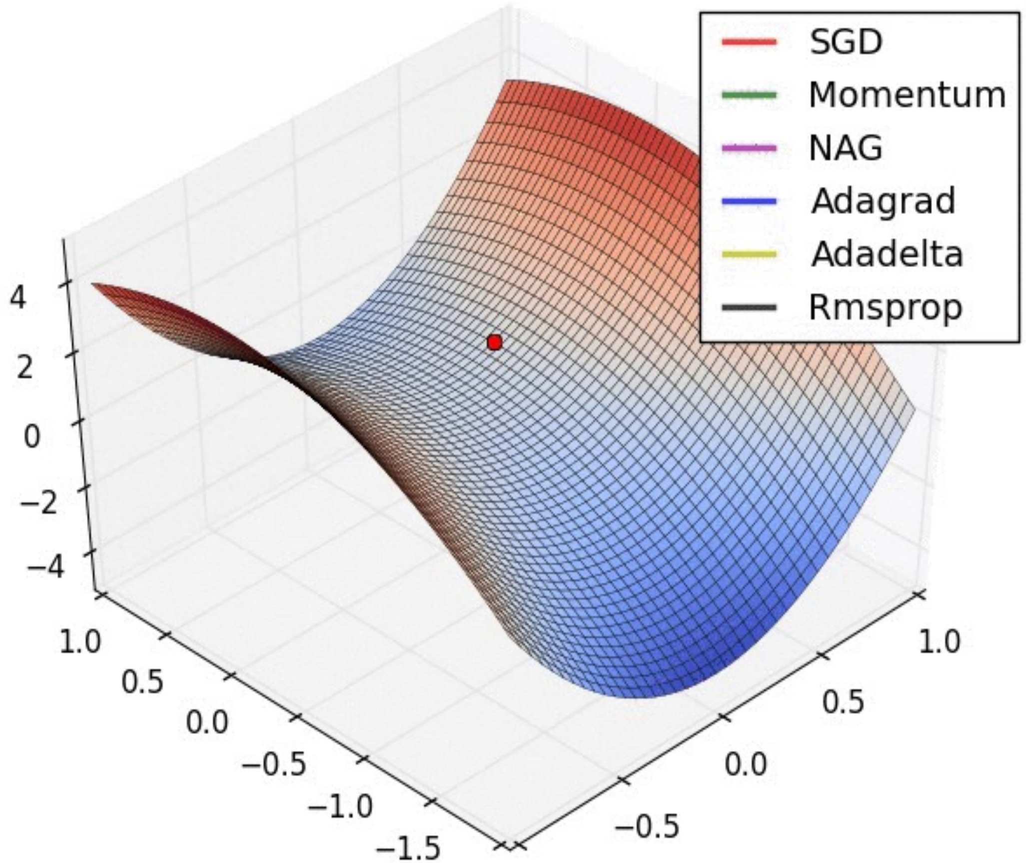
```
tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)
```

$$\left.\begin{aligned} \beta_1 &= 0.9 \\ \beta_2 &= 0.999 \\ \varepsilon &= 10^{-7} \end{aligned}\right\} \text{Default values}$$

Summary

- Gradient descent
  - Step is only proportional to the gradient (constant learning rate)
- Learning rate decay
  - Step decreases (learning rate decreases as the training progresses)
- Momentum
  - Step accumulates past gradients (weighted average)
- RMSprop
  - Step with adaptive learning rate (inversely proportional to past gradients)
- Adam
  - Step as a combination Momentum and RMSprop





Images credit: Alec Radford

Practical aspects

- Adam is one of the preferred algorithm by many developers
  - Adam usually converges faster than other algorithms
  - Adam usually requires less tuning of the learning rate (adaptive learning rate)
- There is not clear common agreement
  - Adam can find solutions that generalize worse than SGD on some datasets [Wilson et al., 2017]
- Other preferred algorithms:
  - SGD with momentum
  - Nadam [Dozat, 2016]

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. Advances in neural information processing systems, 30.

Dozat, T. (2016). Incorporating Nesterov Momentum into Adam. Workshop track - ICLR 2016.

Lecture slides of the master course “Deep Learning”.
© 2025 Martin Molina

This work is licensed under Creative Commons license CC BY-SA 4.0:
https://creativecommons.org/licenses/by-sa/4.0/legalcode

Suggested work citation:

Molina, M. (2025): “Training Methods for Deep Neural Networks: Optimization Algorithms”. Master course (lecture slides). Department of Artificial Intelligence. Universidad Politécnica de Madrid.