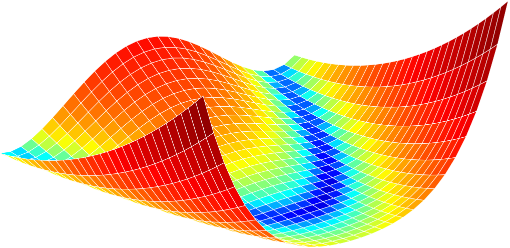
Gradient-based Optimization

A short introduction to optimization in Deep Learning



Christian S. Perone
christian.perone@gmail.com

Agenda

Who Am I

- **Christian S. Perone**
- Machine Learning Engineer / Research
- BSc Computer Science (Brazil/Universidade de Passo Fundo)
- MSc Deep Learning Biomed. Eng. (Canada/Polytechnique Montreal/UdeM)
- Blog

  http://blog.christianperone.com
- Open-source projects

  https://github.com/perone
- Twitter @tarantulae

Section I

Introduction

Motivation

Mathematical optimization is the core of Machine Learning, without it we wouldn't be able to find the needle in the haystack of the parameter space.

Motivation

Mathematical optimization is the core of Machine Learning, without it we wouldn't be able to find the needle in the haystack of the parameter space.

- It materializes in Machine Learning by minimizing an **objective function** such as a divergence or any function that penalizes for mistakes of the model;

Motivation

Mathematical optimization is the core of Machine Learning, without it we wouldn't be able to find the needle in the haystack of the parameter space.

- It materializes in Machine Learning by minimizing an **objective function** such as a divergence or any function that penalizes for mistakes of the model;
- We will talk here about **local methods** that are characterized by the search of an optimal value within a neighboring set of parameter space;

Motivation

Mathematical optimization is the core of Machine Learning, without it we wouldn't be able to find the needle in the haystack of the parameter space.

- It materializes in Machine Learning by minimizing an **objective function** such as a divergence or any function that penalizes for mistakes of the model;
- We will talk here about **local methods** that are characterized by the search of an optimal value within a neighboring set of parameter space;
- We have a huge variety of methods that were recently developed, therefore **this talk is by far from being a comprehensive collection**. I will focus on **intuition and understanding**, instead of throwing algorithms.

Empirical Risk Minimization (ERM)

- On a supervised setting, we want to find a function or a model $f_\theta(\cdot)$ that describes the relationship between a random feature vector $\boldsymbol{x}$ and the label target vector $\boldsymbol{y}$. We assume a joint distribution $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$;

Empirical Risk Minimization (ERM)

- On a supervised setting, we want to find a function or a model $f_\theta(\cdot)$ that describes the relationship between a random feature vector $\boldsymbol{x}$ and the label target vector $\boldsymbol{y}$. We assume a joint distribution $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$;
- We start by defining a loss function L, evaluated as $L(f_\theta(x), y)$ that gives us a penalization for the difference between predictions $f_\theta(x)$ and the true label y;

Empirical Risk Minimization (ERM)

- On a supervised setting, we want to find a function or a model $f_\theta(\cdot)$ that describes the relationship between a random feature vector $\boldsymbol{x}$ and the label target vector $\boldsymbol{y}$. We assume a joint distribution $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$;
- We start by defining a loss function L, evaluated as $L(f_\theta(x), y)$ that gives us a penalization for the difference between predictions $f_\theta(x)$ and the true label y;
- Now, taking the expectation of the loss we have our risk R:

Definition: Risk

$$R(f) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}}[\underbrace{L(f_\theta(\boldsymbol{x}), \boldsymbol{y})}_{\text{Loss}}] = \int L(f_\theta(\boldsymbol{x}), \boldsymbol{y})\, dp_{\text{data}}(\boldsymbol{x}, \boldsymbol{y}),$$

that we want to minimize.

Empirical Risk Minimization (ERM)

- However, we don't know $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$, we only have access to a sample training set $\mathcal{D} = (x_i, y_i) \sim p_{\text{data}}$;

Empirical Risk Minimization (ERM)

- However, we don't know $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$, we only have access to a sample training set $\mathcal{D} = (x_i, y_i) \sim p_{\text{data}}$;
- Therefore, we can approximate the risk with the *empirical risk*:

Definition: Empirical Risk

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x_i), y_i)$$

Empirical Risk Minimization (ERM)

- However, we don't know $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$, we only have access to a sample training set $\mathcal{D} = (x_i, y_i) \sim p_{\text{data}}$;
- Therefore, we can approximate the risk with the *empirical risk*:

Definition: Empirical Risk

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x_i), y_i)$$

- The Empirical Risk Minimization (ERM) principle says that our learning algorithm should minimize the empirical risk;

Empirical Risk Minimization (ERM)

- However, we don't know $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$, we only have access to a sample training set $\mathcal{D} = (x_i, y_i) \sim p_{\text{data}}$;
- Therefore, we can approximate the risk with the *empirical risk*:

Definition: Empirical Risk

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x_i), y_i)$$

- The Empirical Risk Minimization (ERM) principle says that our learning algorithm should minimize the empirical risk;
- The MLE (Maximum Likelihood Estimation) can be posed as a special case of ERM where the loss function is the negative log-likelihood.

Maximum Likelihood Estimation (MLE)

Under the ERM framework we can describe the MLE cost function $J(\cdot)$ as:

$$J(\theta) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}} \underbrace{-\log p_\theta(\boldsymbol{y} \mid \boldsymbol{x})}_{\text{log-likelihood}}$$

where we define the cost as the expectation under the empirical distribution $\hat{p}_{\text{data}}$, as we only have access to a sample training set $\mathcal{D} = (x_i, y_i) \sim p_{\text{data}}$.

Maximum Likelihood Estimation (MLE)

Under the ERM framework we can describe the MLE cost function $J(\cdot)$ as:

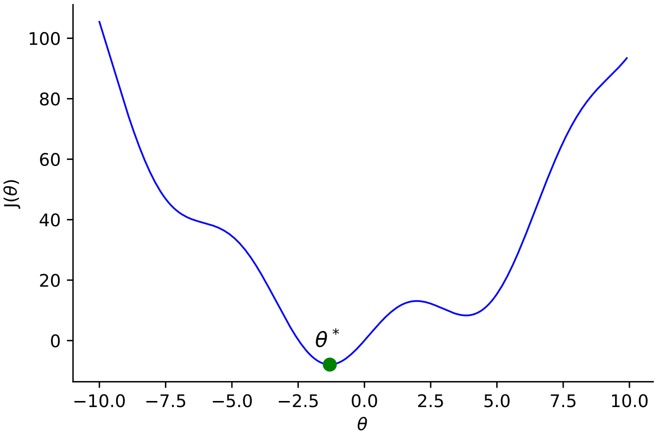
$$J(\theta) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}} \underbrace{-\log p_\theta(\boldsymbol{y} \mid \boldsymbol{x})}_{\text{log-likelihood}}$$

where we define the cost as the expectation under the empirical distribution $\hat{p}_{\text{data}}$, as we only have access to a sample training set $\mathcal{D} = (x_i, y_i) \sim p_{\text{data}}$.

- We might be interested in let's say predicting a statistic of the distribution, such as the mean of $\boldsymbol{y}$ using the predictor $f_\theta(\boldsymbol{x})$
- Our interest here in terms of optimization is:

$$\theta^* = \arg\min_\theta J(\theta), \text{where } \theta \in \mathbb{R}^n$$

The global optimum



Taylor approximation

Let's talk about a powerful calculus tool called *Taylor approximation*:

- Taylor approximation is based on the Taylor theorem[1]:

$$h(\theta) = \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}} + \underbrace{\frac{1}{2}\nabla^2 f(\theta_0)(\theta - \theta_0)^2}_{\text{second-order}},$$

where we want an approximation of the function at the point θ_0;

[1] Taylor's theorem gives an approximation of a k-times differentiable function around a given point by a polynomial of degree k. We're using only up to second-order here.

Taylor approximation

Let's talk about a powerful calculus tool called *Taylor approximation*:

- Taylor approximation is based on the Taylor theorem[1]:

$$h(\theta) = \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}} + \underbrace{\frac{1}{2}\nabla^2 f(\theta_0)(\theta - \theta_0)^2}_{\text{second-order}},$$

  where we want an approximation of the function at the point θ_0;
- This theorem is very powerful as it allows us to approximate any differentiable (and twice differentiable) function;

[1] Taylor's theorem gives an approximation of a k-times differentiable function around a given point by a polynomial of degree k. We're using only up to second-order here.

Taylor approximation

Let's talk about a powerful calculus tool called *Taylor approximation*:

- Taylor approximation is based on the Taylor theorem[1]:

$$h(\theta) = \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}} + \underbrace{\frac{1}{2}\nabla^2 f(\theta_0)(\theta - \theta_0)^2}_{\text{second-order}},$$

  where we want an approximation of the function at the point θ_0;
- This theorem is very powerful as it allows us to approximate any differentiable (and twice differentiable) function;
- The $\nabla^2 f(\cdot)$ is also called the Hessian, or $\mathbf{H}_f$. We will talk more about it later;

[1] Taylor's theorem gives an approximation of a k-times differentiable function around a given point by a polynomial of degree k. We're using only up to second-order here.

Taylor approximation

Let's talk about a powerful calculus tool called *Taylor approximation*:

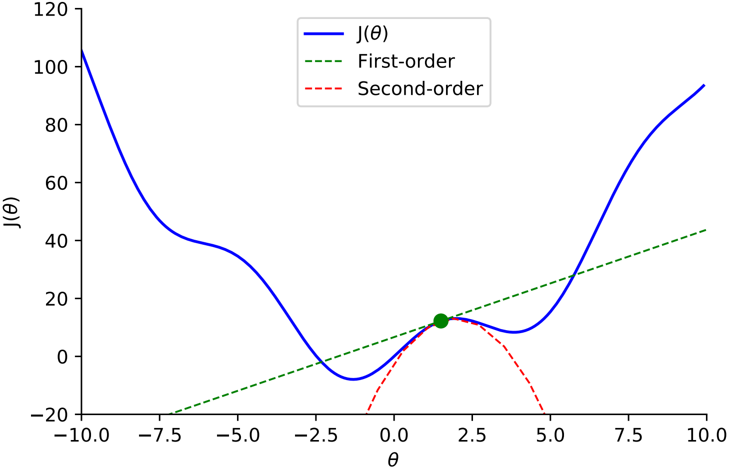
- Taylor approximation is based on the Taylor theorem[1]:

$$h(\theta) = \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}} + \underbrace{\frac{1}{2}\nabla^2 f(\theta_0)(\theta - \theta_0)^2}_{\text{second-order}},$$

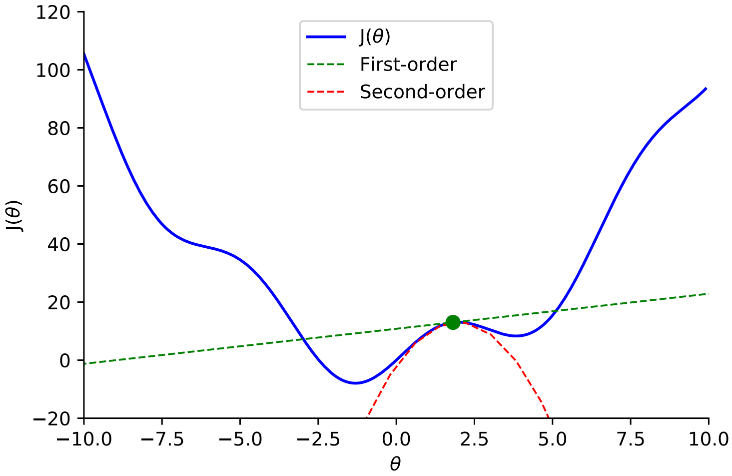
  where we want an approximation of the function at the point θ_0;
- This theorem is very powerful as it allows us to approximate any differentiable (and twice differentiable) function;
- The $\nabla^2 f(\cdot)$ is also called the Hessian, or $\mathbf{H}_f$. We will talk more about it later;
- We will understand the deep connection of this approximation with Gradient Descent.

[1] Taylor's theorem gives an approximation of a k-times differentiable function around a given point by a polynomial of degree k. We're using only up to second-order here.

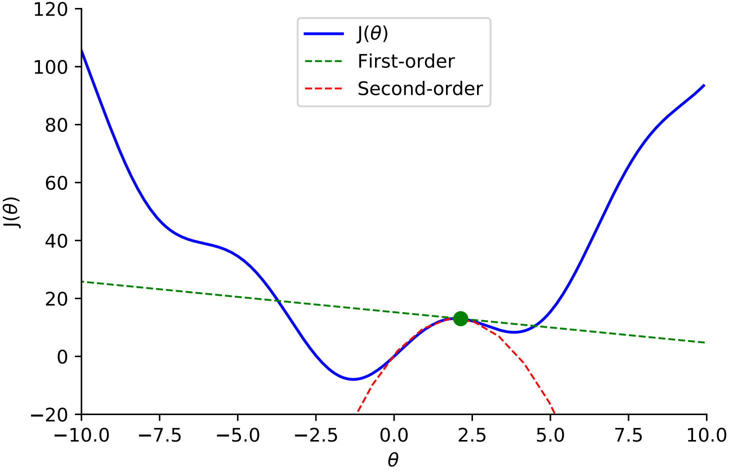
Taylor approximation



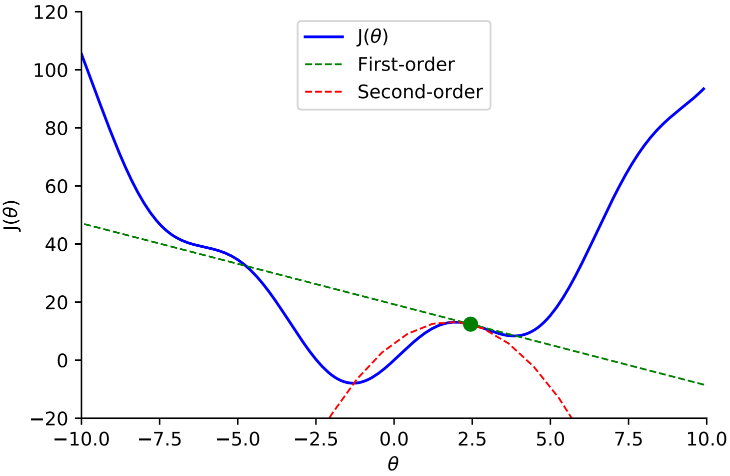
Taylor approximation



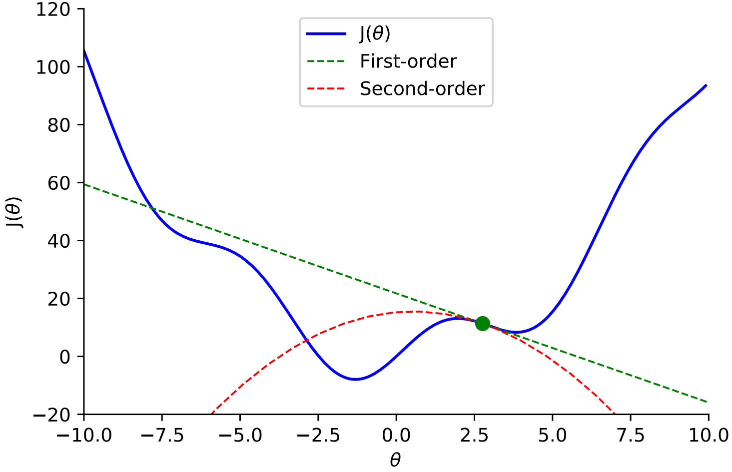
Taylor approximation



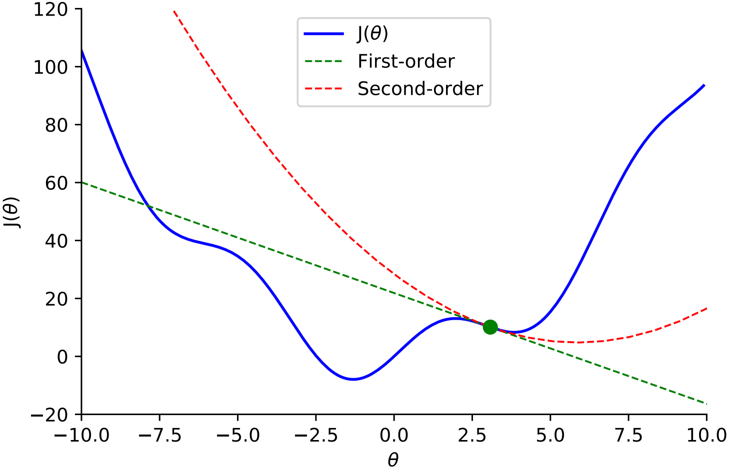
Taylor approximation



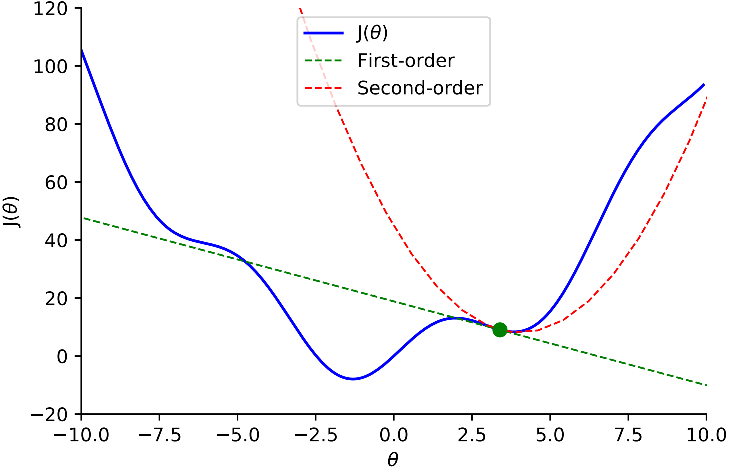
Taylor approximation



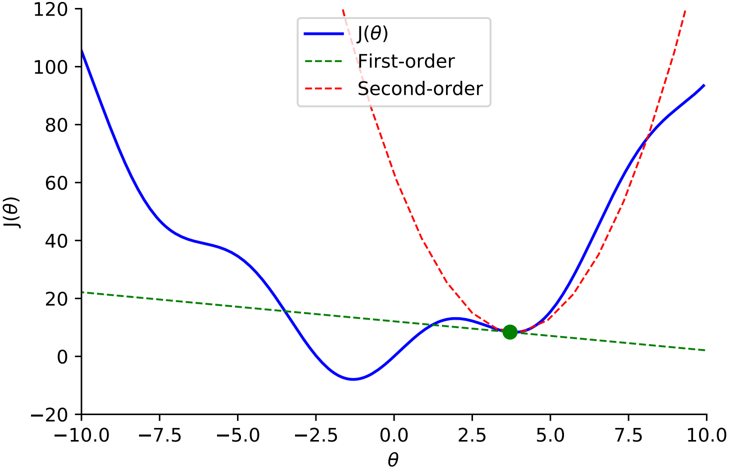
Taylor approximation



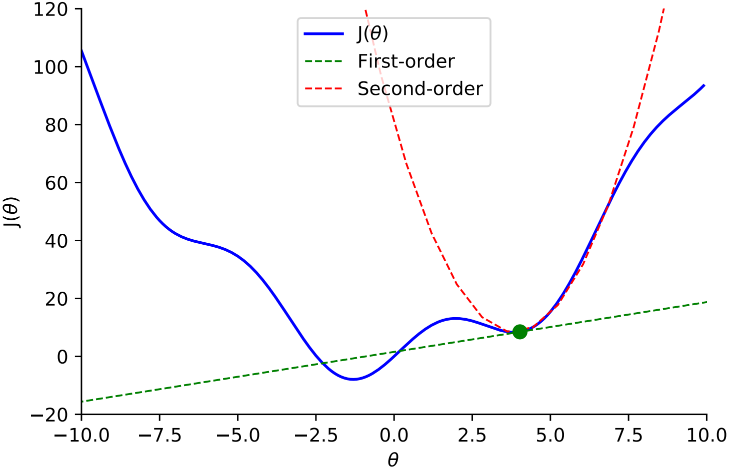
Taylor approximation



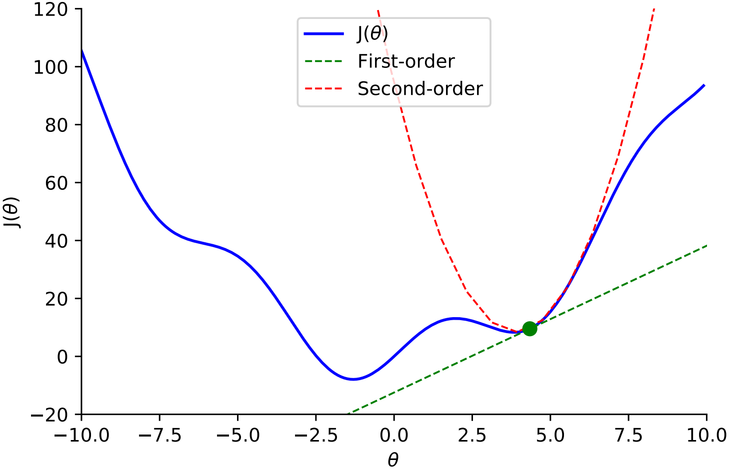
Taylor approximation



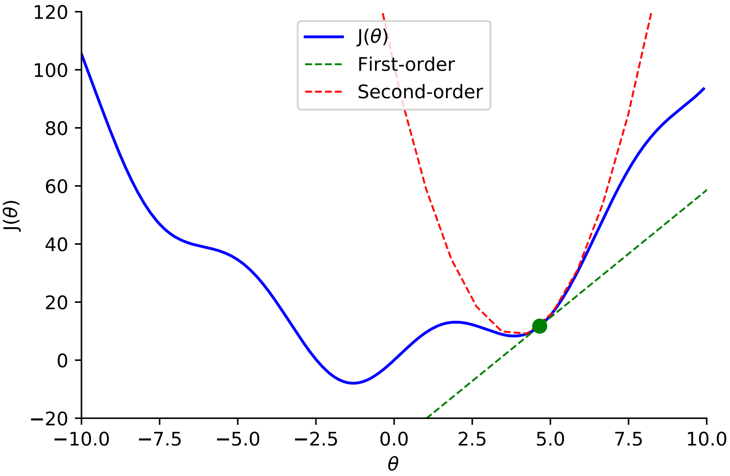
Taylor approximation



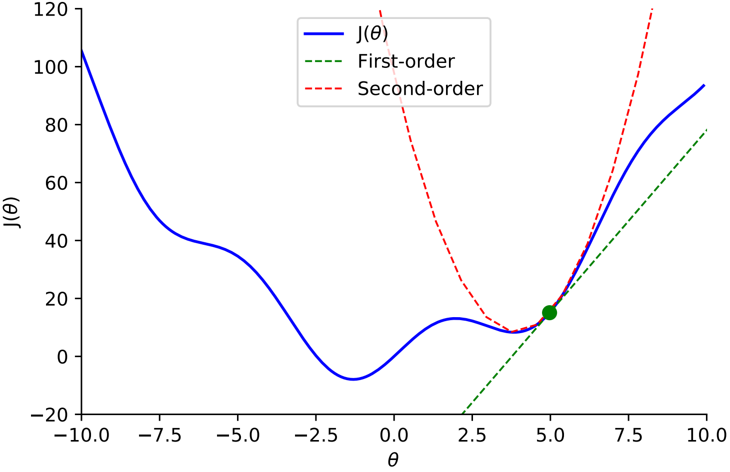
Taylor approximation



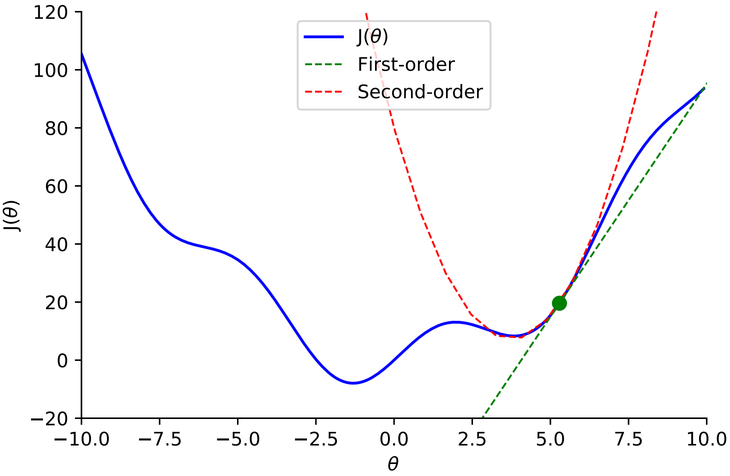
Taylor approximation



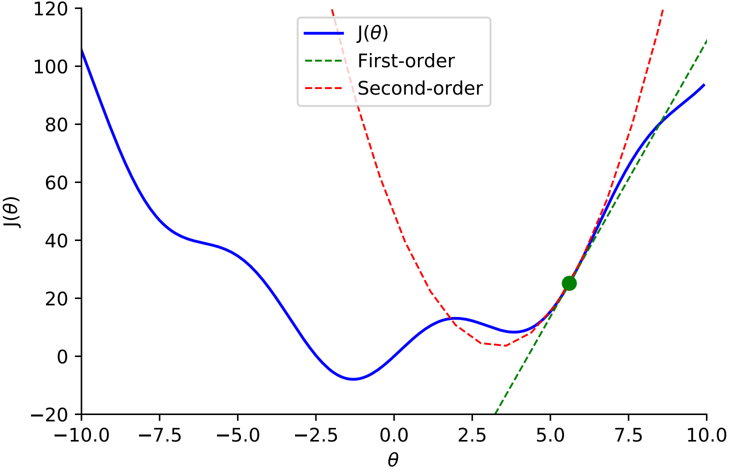
Taylor approximation



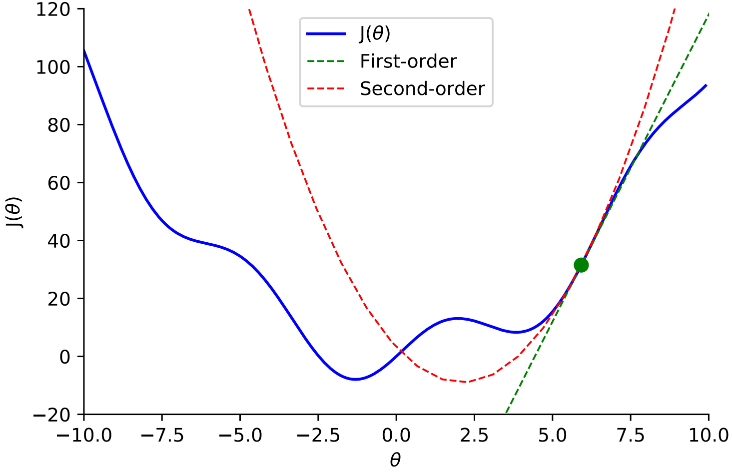
Taylor approximation



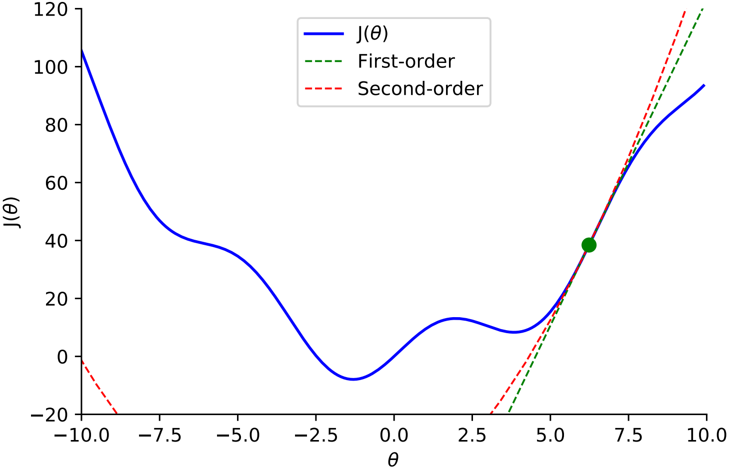
Taylor approximation



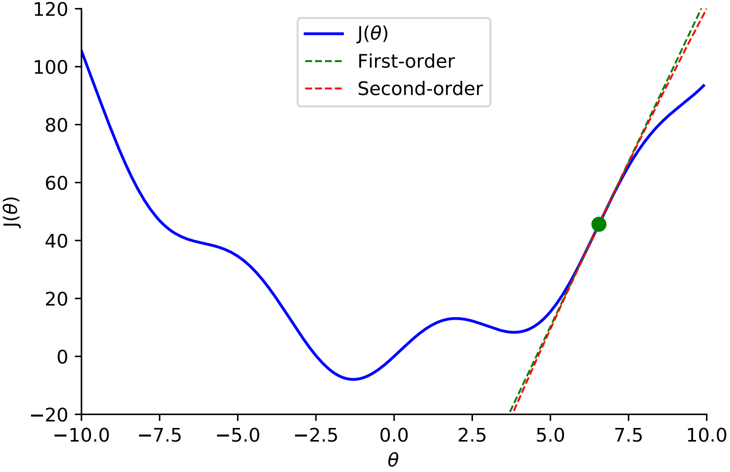
Taylor approximation



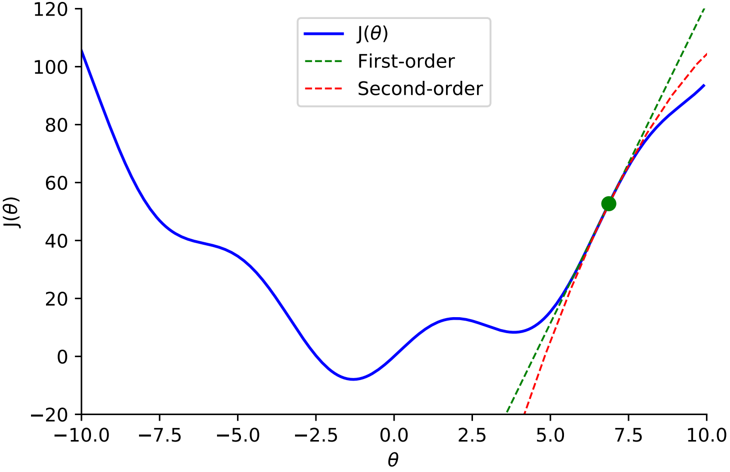
Taylor approximation



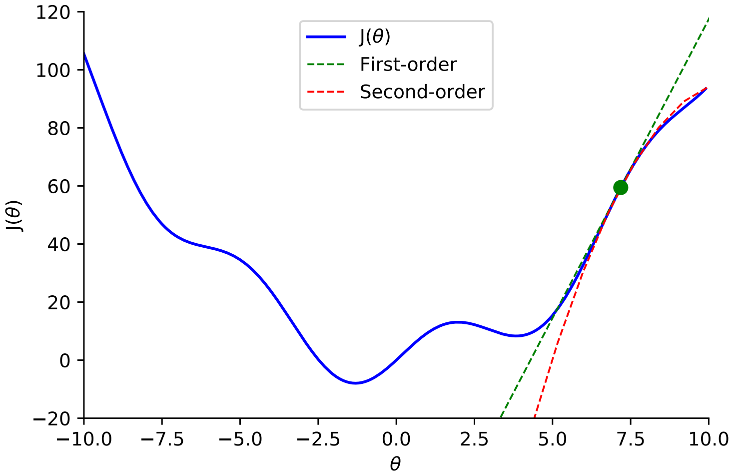
Taylor approximation



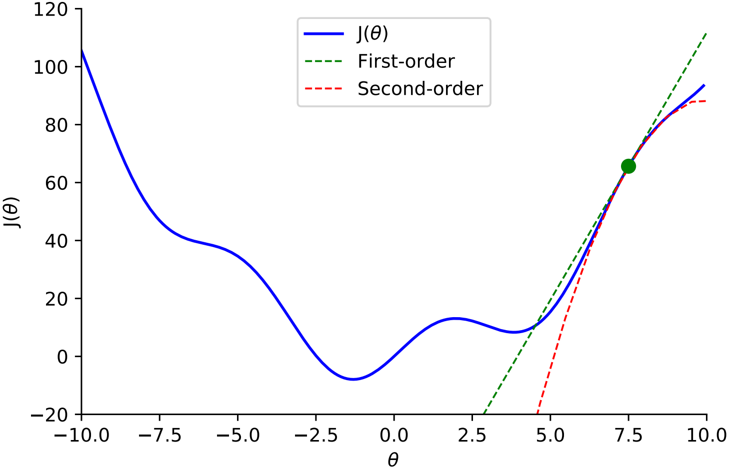
Taylor approximation



Taylor approximation



Taylor approximation



Taylor approximation in Jax

```
from jax import grad

def taylor_first_order(θ, θ0):
    return f(θ0) + grad(f)(θ0)*(θ - θ0)

def taylor_second_order(θ, θ0):
    d1 = taylor_first_order(θ, θ0)
    d2 = 1./2. * grad(grad(f))(θ0) * (θ - a)**2
    return d1 + d2
```

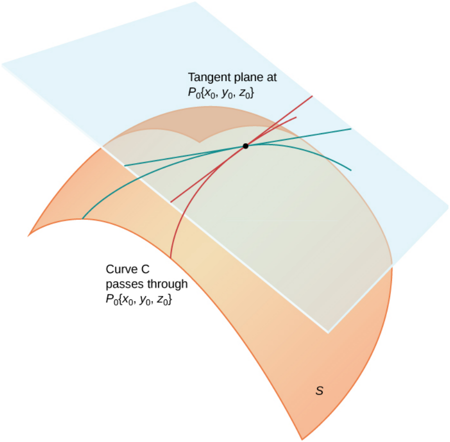
Do not use greek symbols on your Python code, your colleagues will curse you.

Taylor approximation in Jax

```
from jax import grad

def taylor_first_order(θ, θ0):
    return f(θ0) + grad(f)(θ0)*(θ - θ0)

def taylor_second_order(θ, θ0):
    d1 = taylor_first_order(θ, θ0)
    d2 = 1./2. * grad(grad(f))(θ0) * (θ - a)**2
    return d1 + d2


>>> taylor_first_order(6.01, 6.0)
33.421864
>>> taylor_second_order(6.01, 6.0)
33.422104
>>> taylor_first_order(6.5, 6.0)
44.0067
>>> taylor_second_order(6.5, 6.0)
44.60597
```

Do not use greek symbols on your Python code, your colleagues will curse you.

Linear approximation plane



Source: Tangent Planes and Linear Approximations. Calculus Volume 3. Rice University. 2020. Creative Commons Attribution 4.0 International License.

Local approximation and second-order

- Let's now think about that second-order term:

$$h(\theta) = \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}} + \underbrace{\frac{1}{2}\nabla^2 f(\theta_0)(\theta - \theta_0)^2}_{\text{second-order}},$$

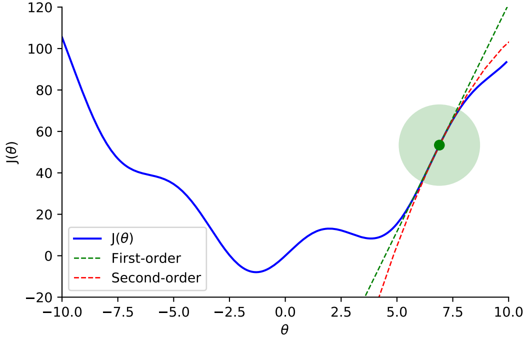
- If we do a small step from θ_0, what happens with the second-term ?

Local approximation and second-order

- Let's now think about that second-order term:

$$h(\theta) = \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}} + \underbrace{\frac{1}{2}\nabla^2 f(\theta_0)(\theta - \theta_0)^2}_{\text{second-order}},$$

- If we do a small step from θ_0, what happens with the second-term ?



The steepest descent

- Even if $f(\cdot)$ is very complex, **locally** it is simple, and we can use a simple function to approximate it, a linear function:

$$h(\theta) \approx \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}}$$

- This is also called *linearization*;

The steepest descent

- Even if $f(\cdot)$ is very complex, **locally** it is simple, and we can use a simple function to approximate it, a linear function:

$$h(\theta) \approx \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}}$$

- This is also called *linearization*;
- It is already apparent what we need now. How can we guarantee, locally, that we can always minimize the function (reduce the loss) ?

The steepest descent

- Even if $f(\cdot)$ is very complex, **locally** it is simple, and we can use a simple function to approximate it, a linear function:

$$h(\theta) \approx \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}}$$

- This is also called *linearization*;
- It is already apparent what we need now. How can we guarantee, locally, that we can always minimize the function (reduce the loss) ?
- We can just follow the slope (negative) of the approximation that is given by $-\nabla f(\theta_0)$;

The steepest descent

- Even if $f(\cdot)$ is very complex, **locally** it is simple, and we can use a simple function to approximate it, a linear function:

$$h(\theta) \approx \underbrace{f(\theta_0) + \nabla f(\theta_0)(\theta - \theta_0)}_{\text{first-order}}$$

- This is also called *linearization*;
- It is already apparent what we need now. How can we guarantee, locally, that we can always minimize the function (reduce the loss) ?
- We can just follow the slope (negative) of the approximation that is given by $-\nabla f(\theta_0)$;
- No twice differentiability requirement, less computational resources;

Section II

Gradient Descent

Gradient Descent

Algorithm The general gradient descent algorithm.

Input: initial weights $\theta^{(0)}$, iterations T, learning rate η
Output: final weights $\theta^{(T)}$

1. **for** $t = 0$ **to** $T - 1$
2. compute $\nabla L(\theta^{(t)})$
3. $\theta^{(t+1)} := \theta^{(t)} - \eta \nabla L(\theta^{(t)})$
4. **return** $\theta^{(T)}$

Gradient Descent

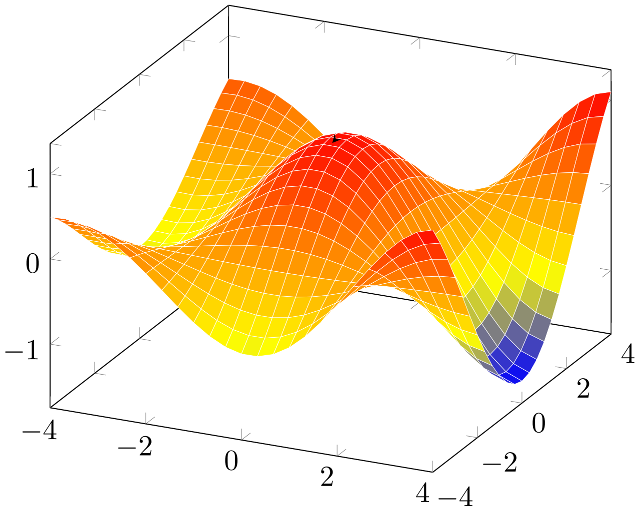
Algorithm The general gradient descent algorithm.

Input: initial weights $\theta^{(0)}$, iterations T, learning rate η
Output: final weights $\theta^{(T)}$
1. **for** $t = 0$ **to** $T - 1$
2. compute $\nabla L(\theta^{(t)})$
3. $\theta^{(t+1)} := \theta^{(t)} - \eta \nabla L(\theta^{(t)})$
4. **return** $\theta^{(T)}$

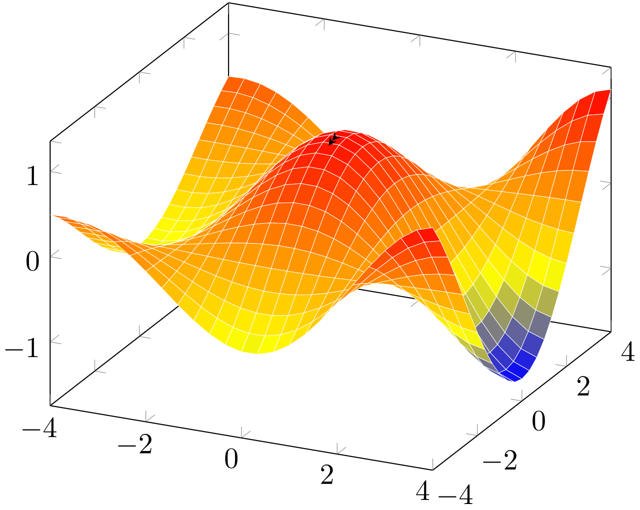
The important part here is the iterative rule:

$$\theta^{(t+1)} = \theta^{(t)} - \underbrace{\eta \nabla L(\theta^{(t)})}_{\text{How much we move}}$$

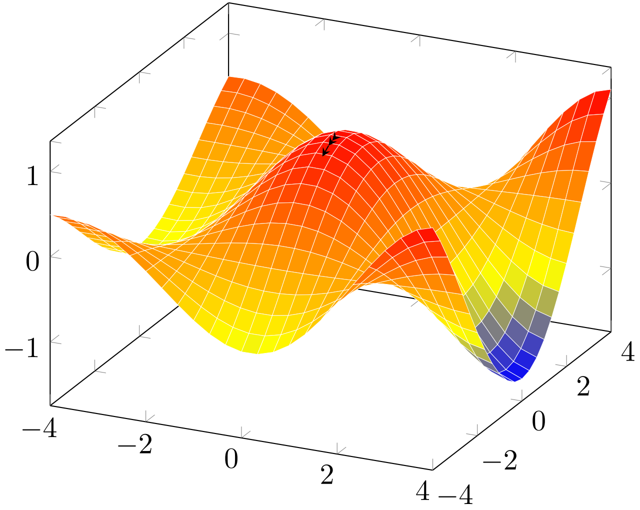
Gradient Descent - Loss surface



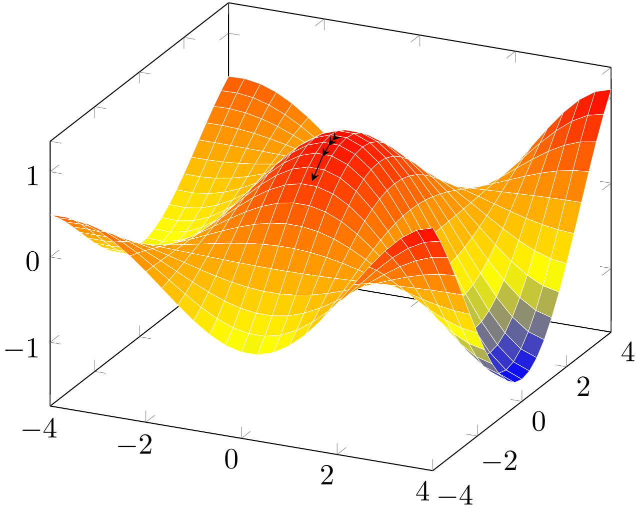
Gradient Descent - Loss surface



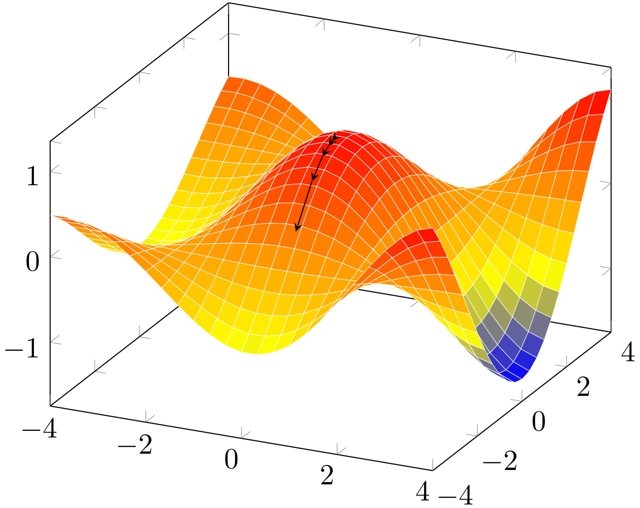
Gradient Descent - Loss surface



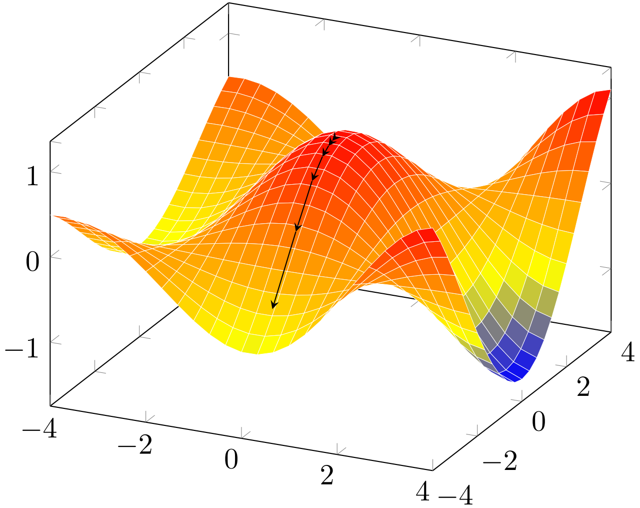
Gradient Descent - Loss surface



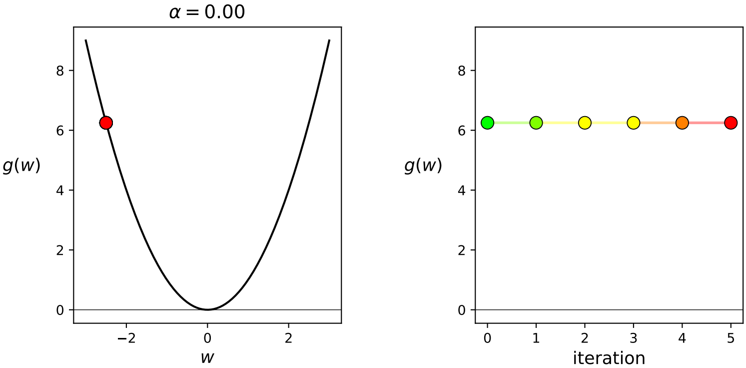
Gradient Descent - Loss surface



Gradient Descent - Loss surface

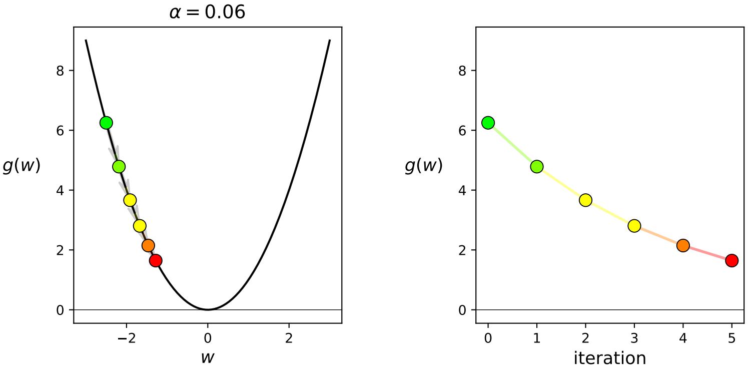


Learning Rate



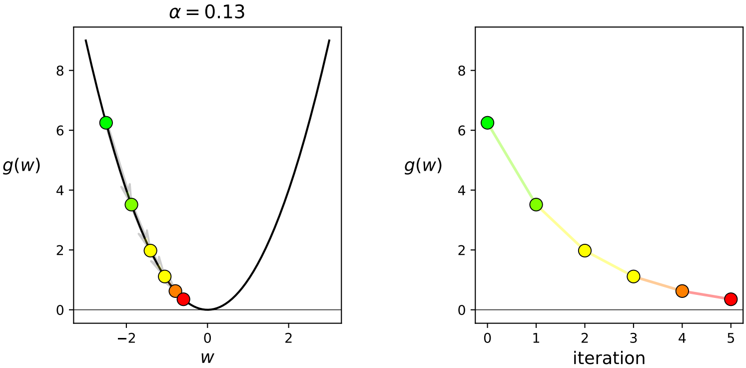
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



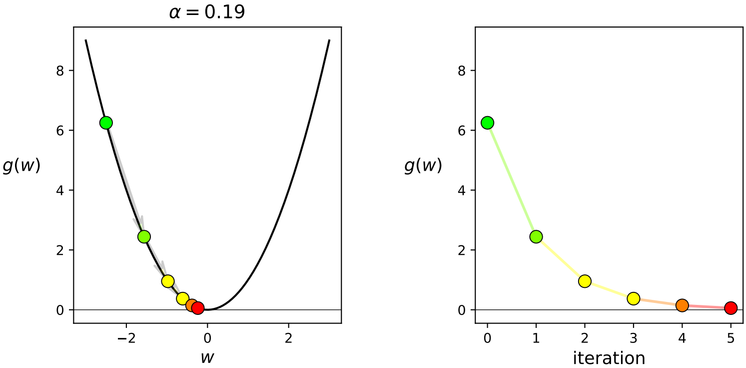
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate

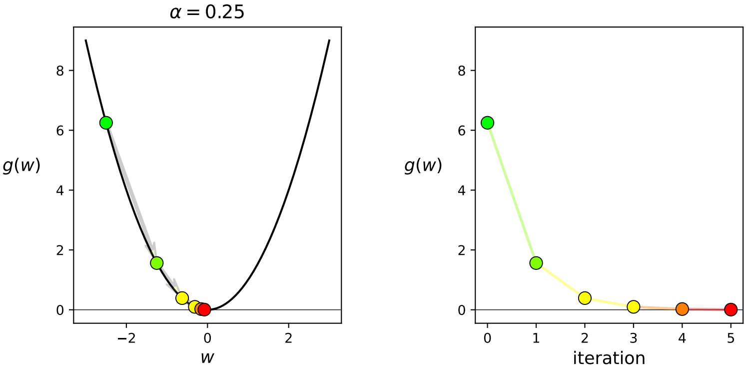


Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate

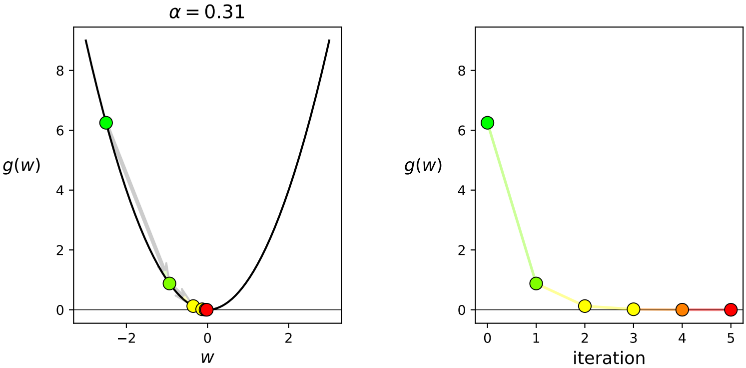


Learning Rate



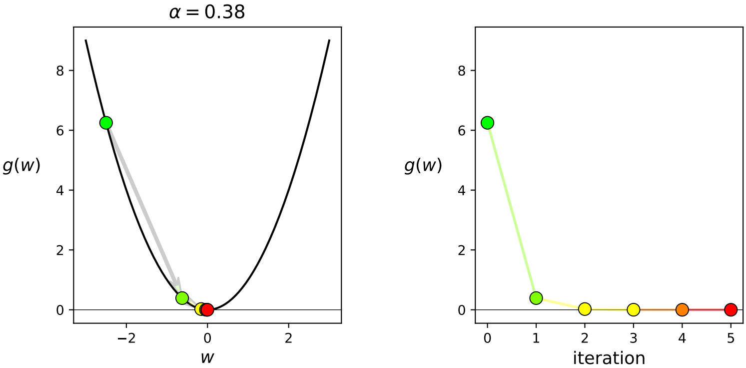
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



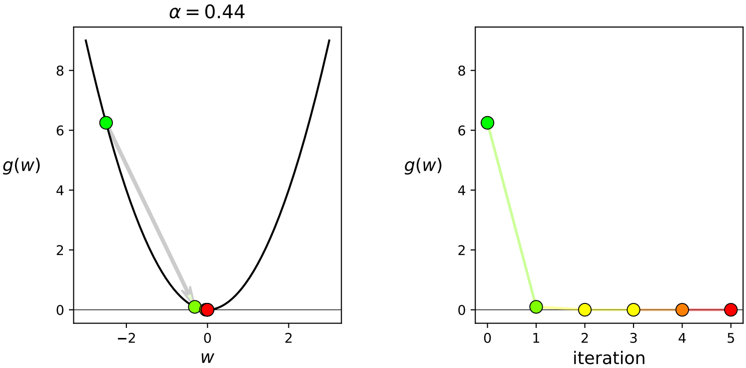
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



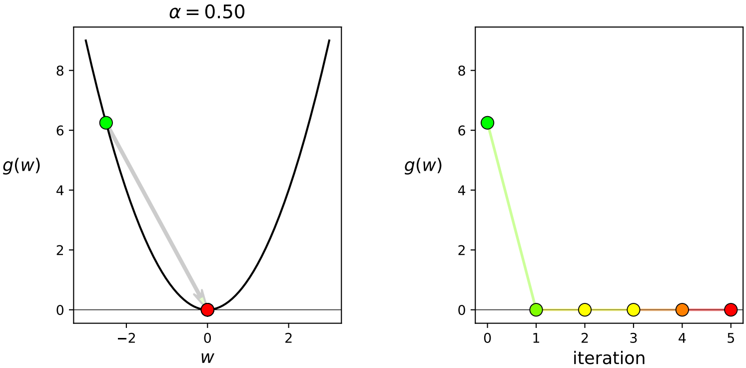
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



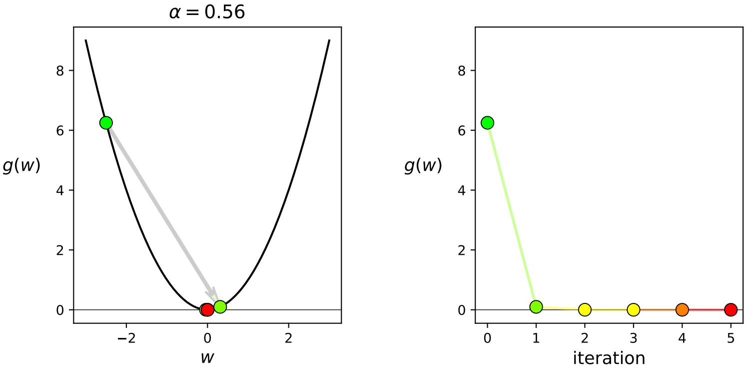
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



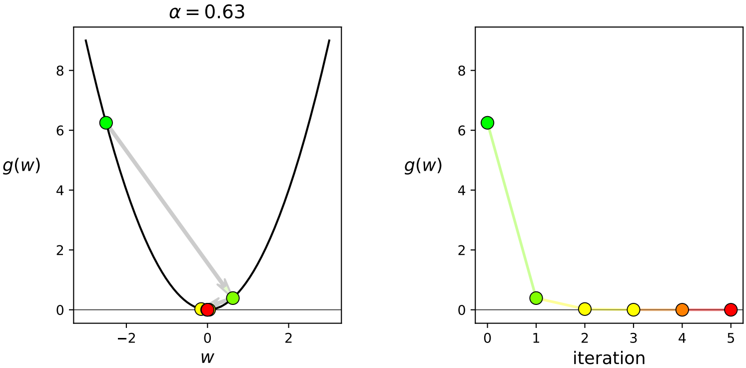
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



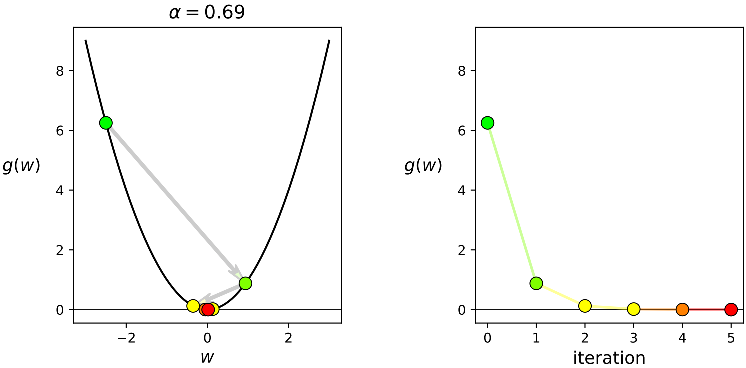
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



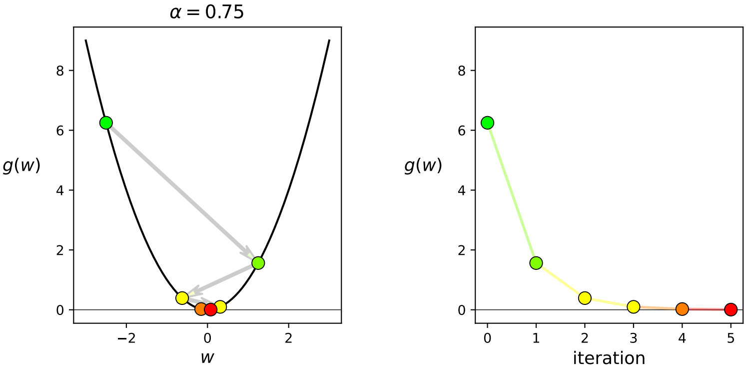
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



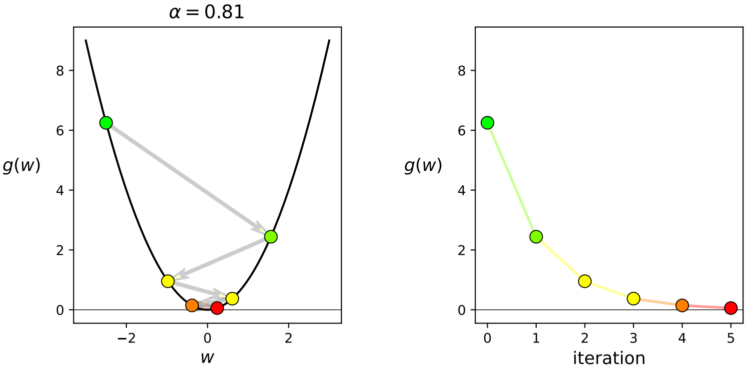
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



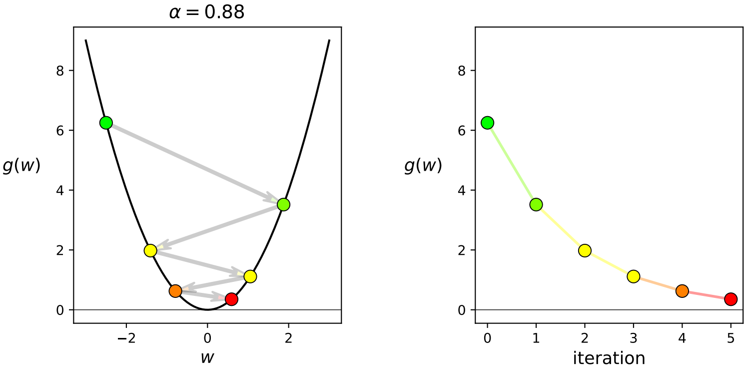
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



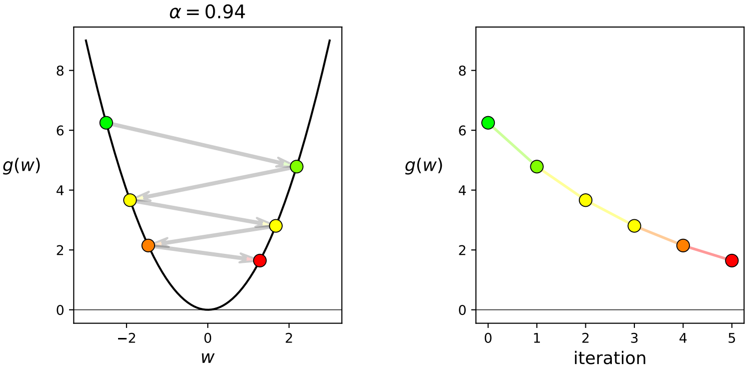
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



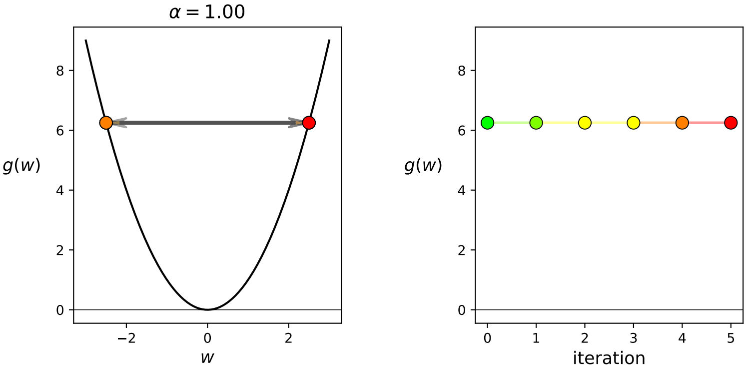
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



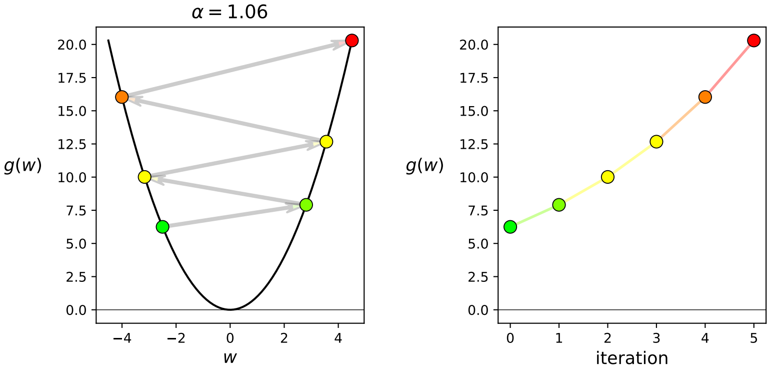
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



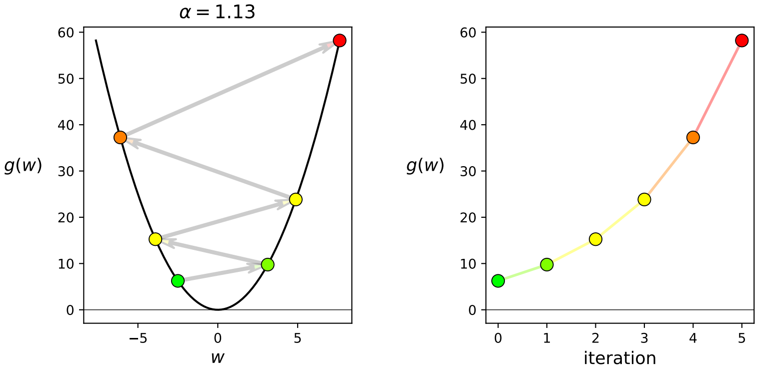
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



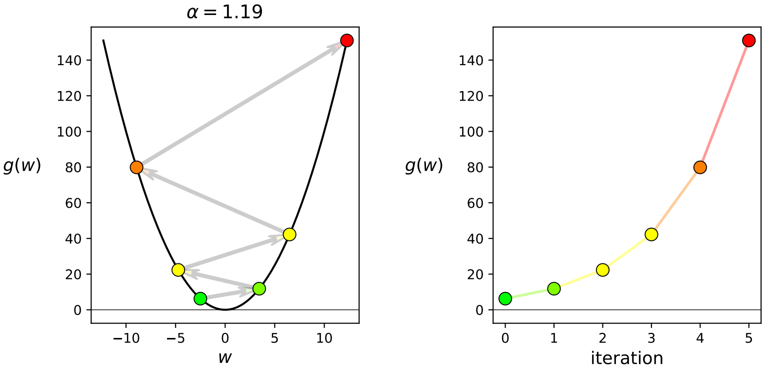
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



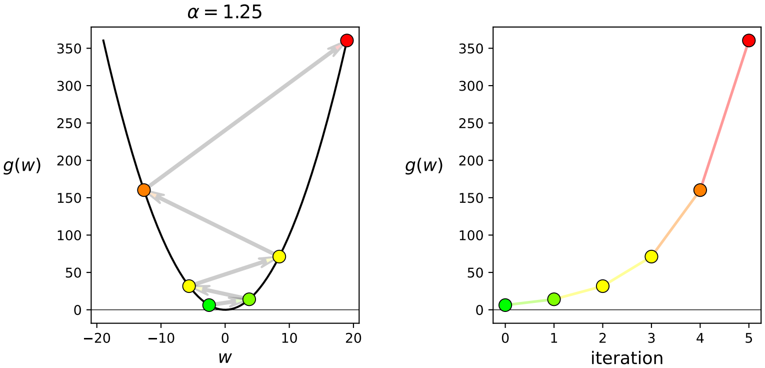
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



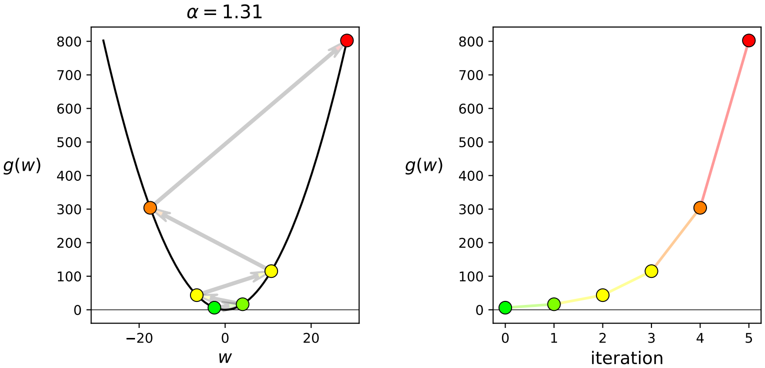
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



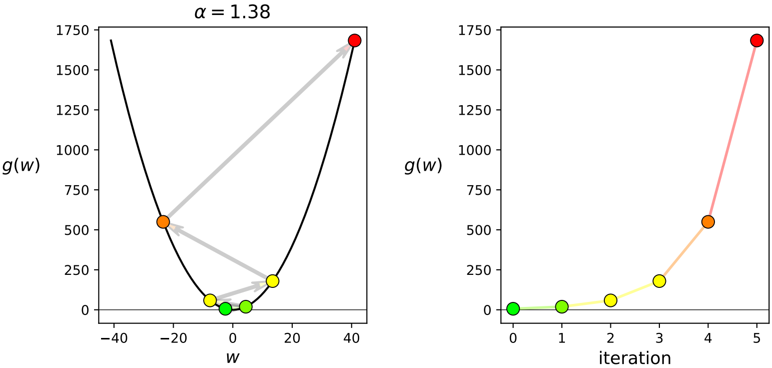
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



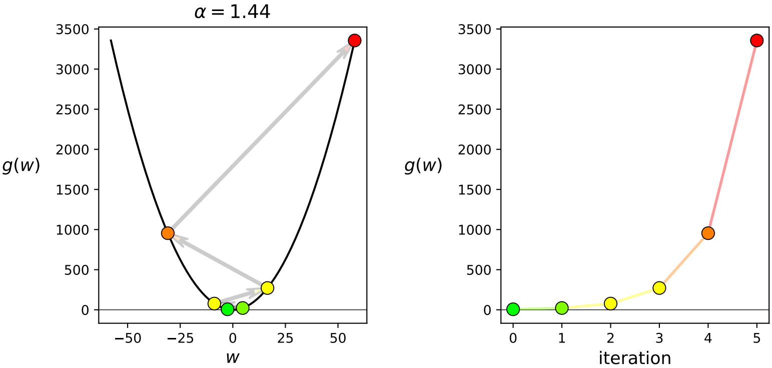
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



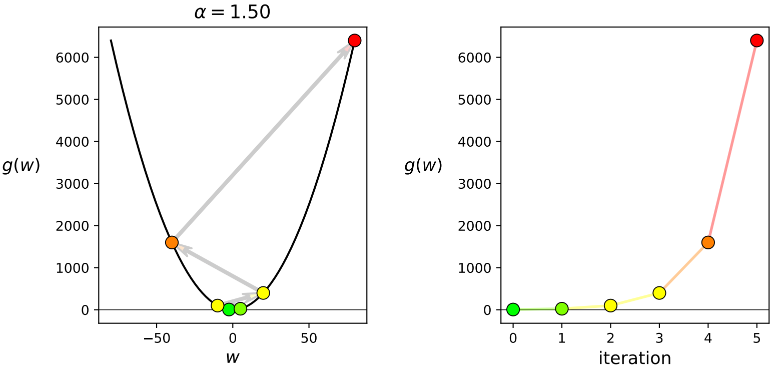
Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

Learning Rate



Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

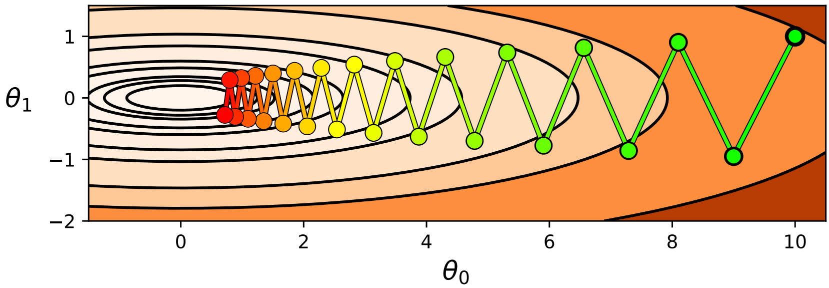
Learning Rate



Source: Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License. Used with permission from the authors.

High curvatures

Gradient descent can suffer on some pathological curvatures and cause a lot of oscillations:



Source: Code adapted from Machine Learning Refined. Jeremy Watt and Reza Borhani. 2020. Creative Commons Attribution 4.0 International License.

Momentum

Momentum is a method to damp out oscillations:
Vanilla gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(\theta^{(t)})$$

Momentum

Momentum is a method to damp out oscillations:
Vanilla gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(\theta^{(t)})$$

Momentum:

$$V^{(t+1)} = \underbrace{\beta}_{\text{Constant}} V^{(t)} + \nabla L(\theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{V^{(t+1)}}_{\text{Momentum buffer}}$$

Momentum

Momentum is a method to damp out oscillations:
Vanilla gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(\theta^{(t)})$$

Momentum:

$$V^{(t+1)} = \underbrace{\beta}_{\text{Constant}} V^{(t)} + \nabla L(\theta^{(t)})$$
$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{V^{(t+1)}}_{\text{Momentum buffer}}$$

- Momentum works by acceleration and smoothing, it makes the trajectories to take more time to react to changes in the loss landscape;

Momentum

Momentum is a method to damp out oscillations:
Vanilla gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(\theta^{(t)})$$

Momentum:

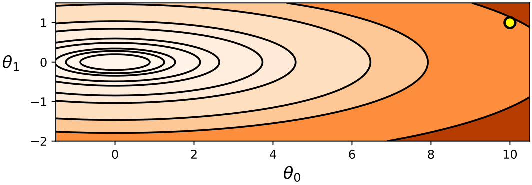
$$V^{(t+1)} = \underbrace{\beta}_{\text{Constant}} V^{(t)} + \nabla L(\theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{V^{(t+1)}}_{\text{Momentum buffer}}$$

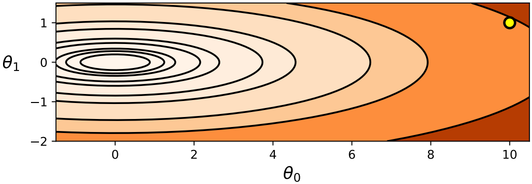
- Momentum works by acceleration and smoothing, it makes the trajectories to take more time to react to changes in the loss landscape;
- Note that with $\beta = 0$ we recover vanilla Gradient descent;

Momentum

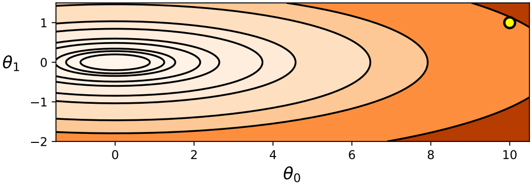
$$\beta = 0.0$$



$$\beta = 0.1$$



$$\beta = 0.7$$



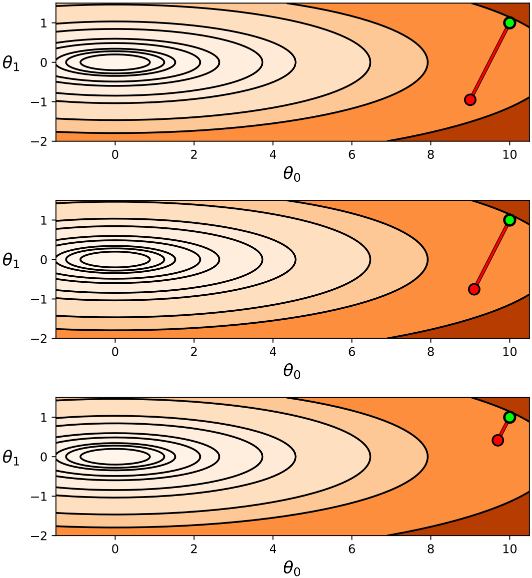
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



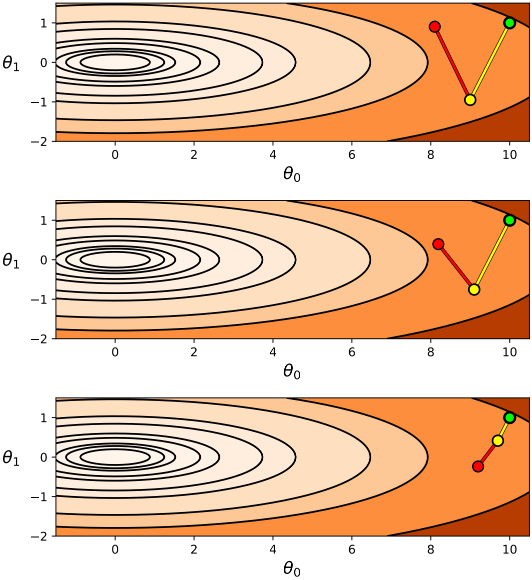
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



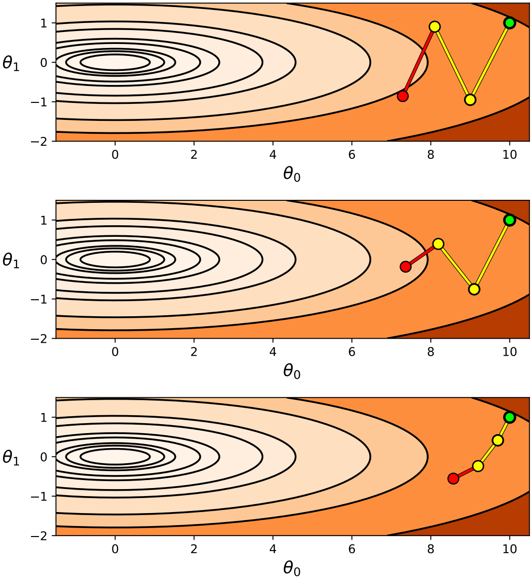
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



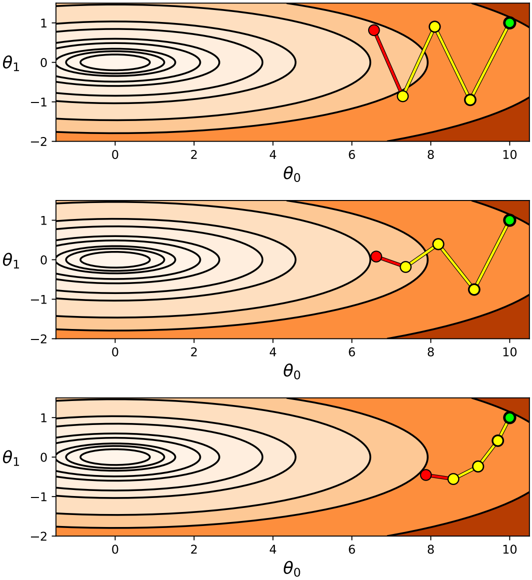
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

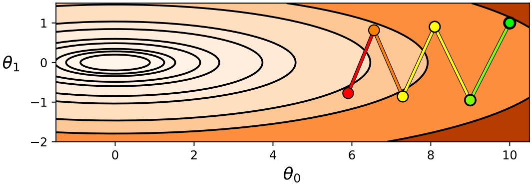
$\beta = 0.7$



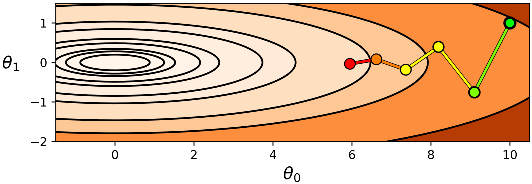
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

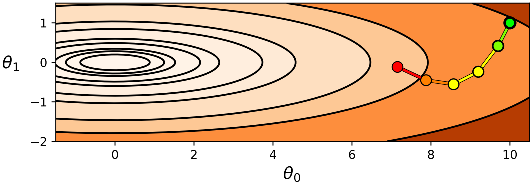
$\beta = 0.0$



$\beta = 0.1$



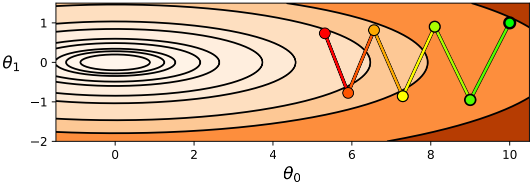
$\beta = 0.7$



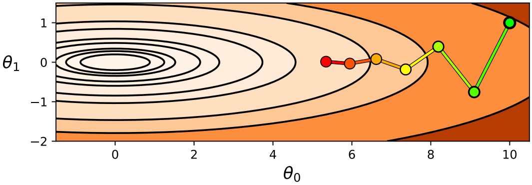
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

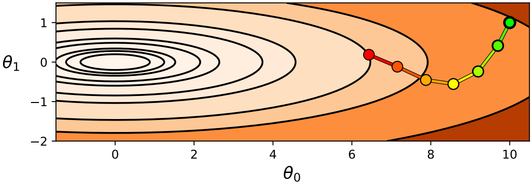
$\beta = 0.0$



$\beta = 0.1$



$\beta = 0.7$



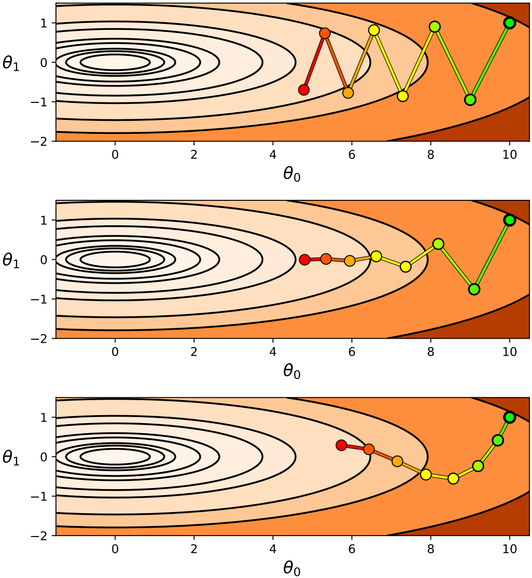
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

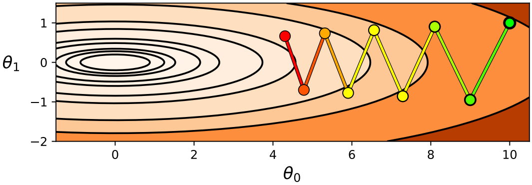
$\beta = 0.7$



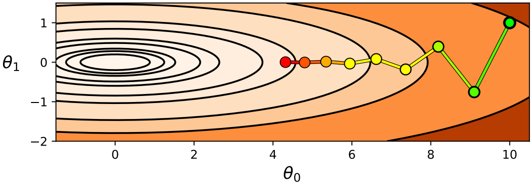
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

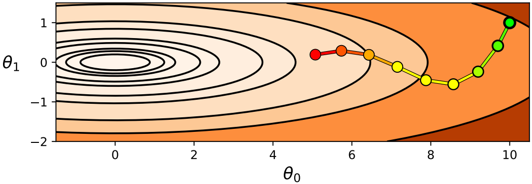
$\beta = 0.0$



$\beta = 0.1$



$\beta = 0.7$



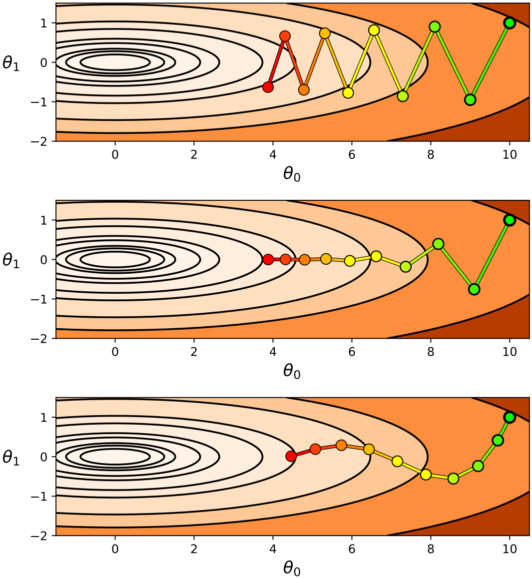
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



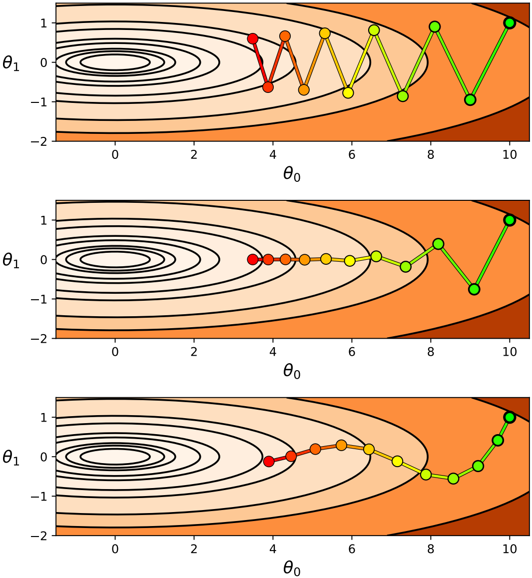
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



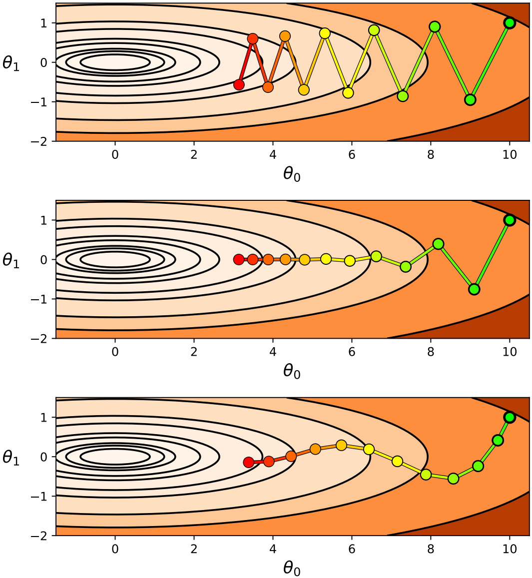
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



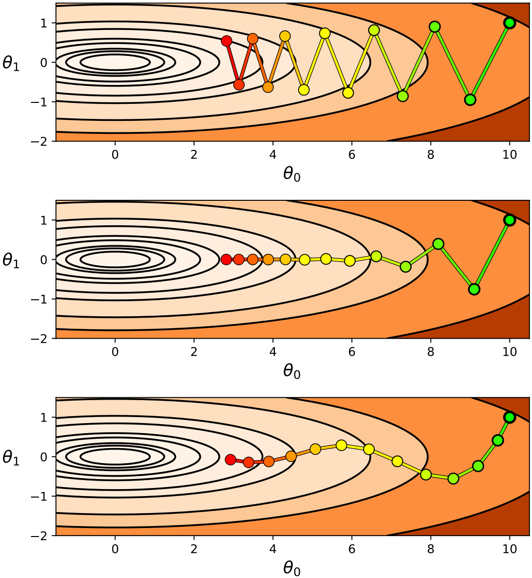
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

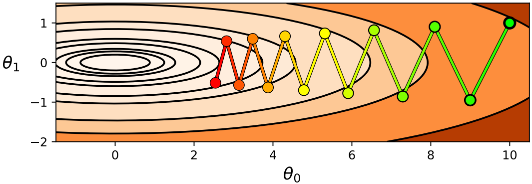
$\beta = 0.7$



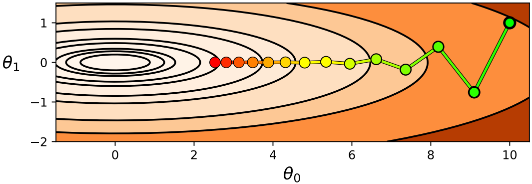
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

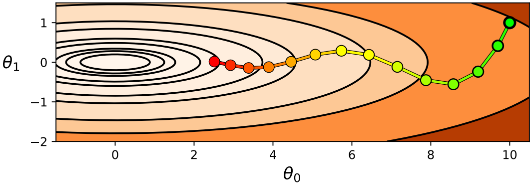
$\beta = 0.0$



$\beta = 0.1$



$\beta = 0.7$



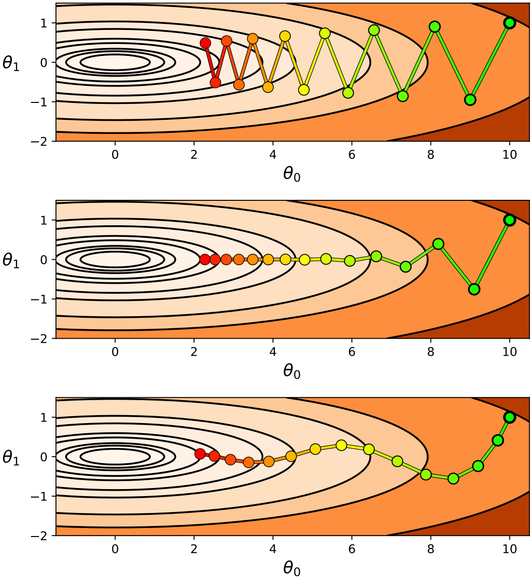
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



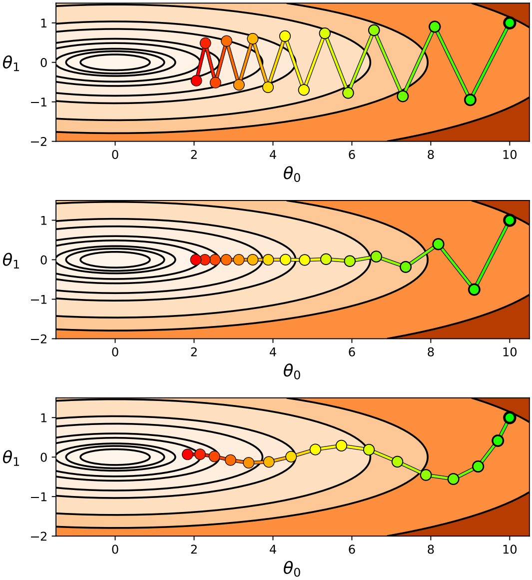
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



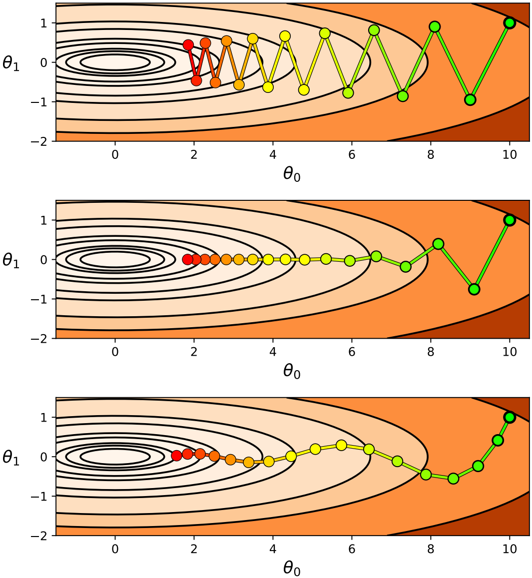
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

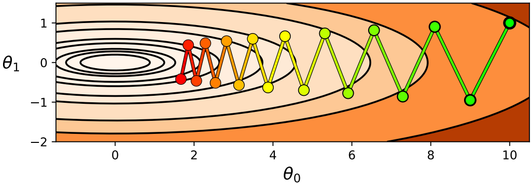
$\beta = 0.7$



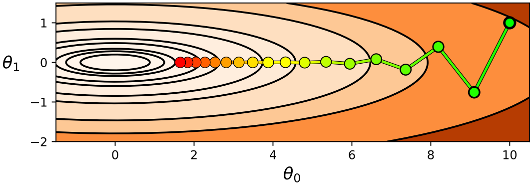
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

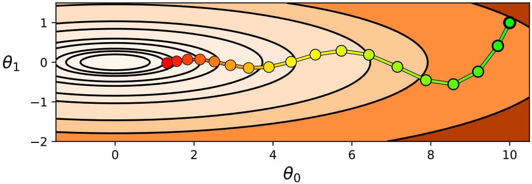
$\beta = 0.0$



$\beta = 0.1$



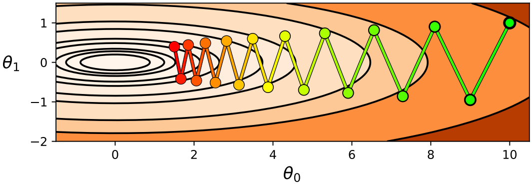
$\beta = 0.7$



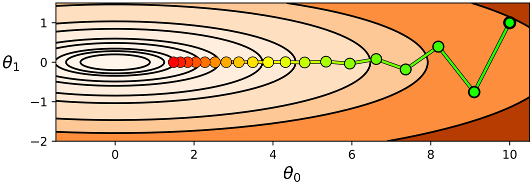
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

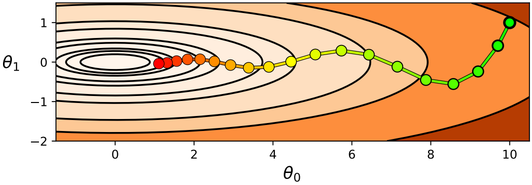
$\beta = 0.0$



$\beta = 0.1$



$\beta = 0.7$



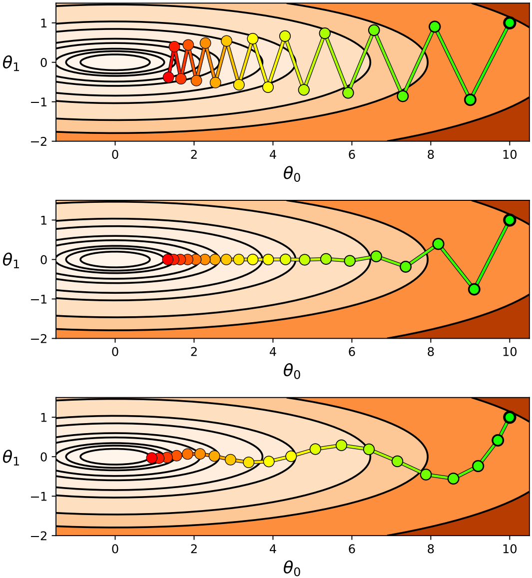
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



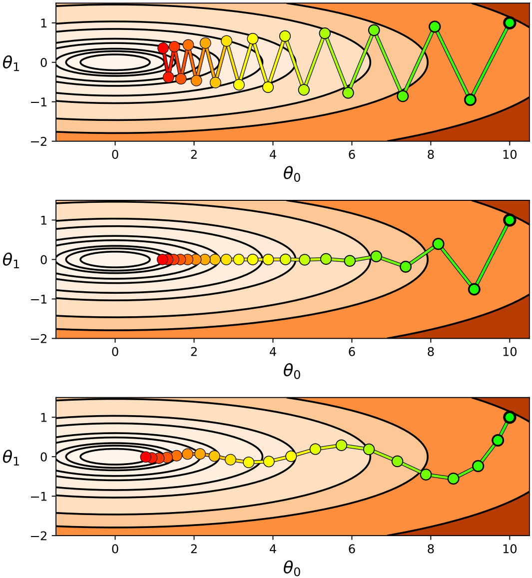
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

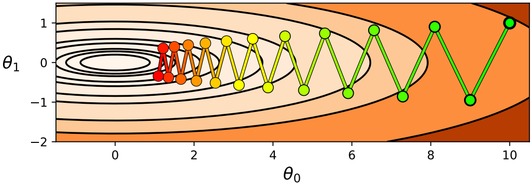
$\beta = 0.7$



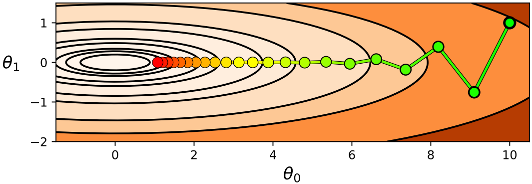
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

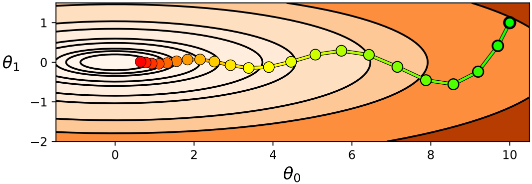
$\beta = 0.0$



$\beta = 0.1$

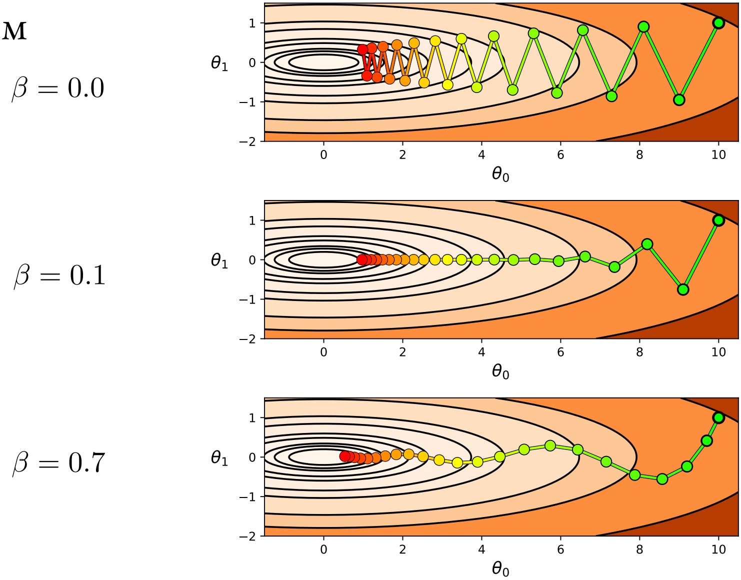


$\beta = 0.7$



Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum



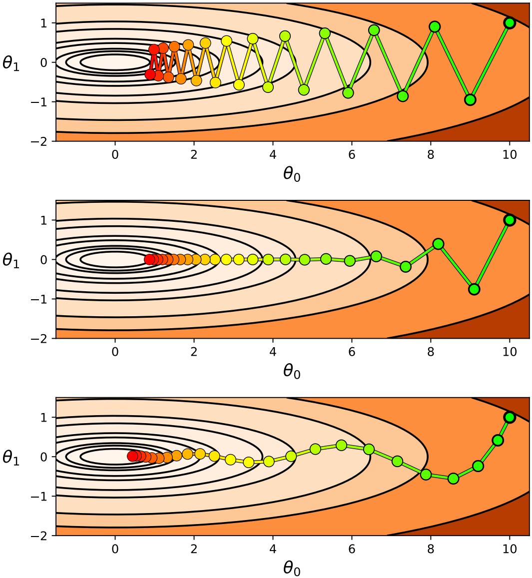
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

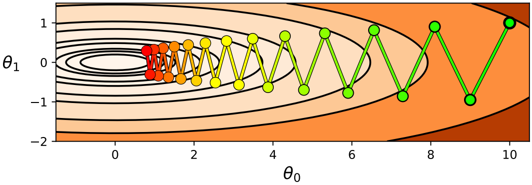
$\beta = 0.7$



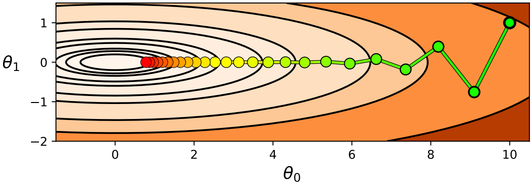
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

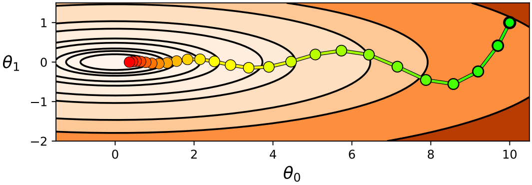
$\beta = 0.0$



$\beta = 0.1$



$\beta = 0.7$



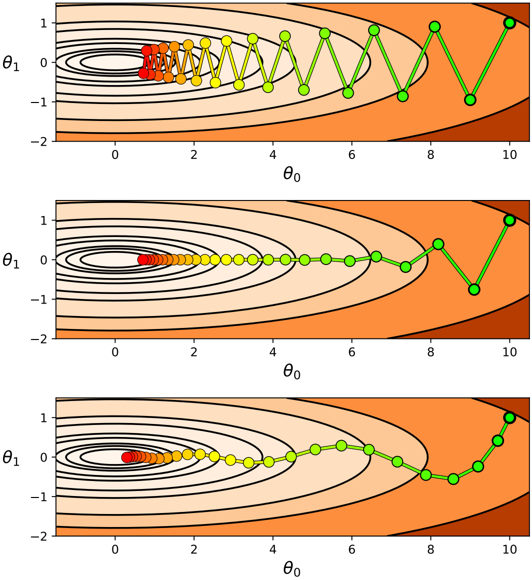
Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

$\beta = 0.0$

$\beta = 0.1$

$\beta = 0.7$



Source: Code adapted from Machine Learning Refined. Jeremy Watt et al. 2020.

Momentum

Pause for a quick demo from Lili Jiang, from:

https://github.com/lilipads/gradient_descent_viz

Stochastic Gradient Descent (SGD)

It turns out that we don't quite need to compute the gradients $\nabla L(\theta)$ over the whole dataset at every iteration of Gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\nabla L_i(\theta^{(t)})}_{\text{Individual samples}}$$

where we do random sampling (or not, we can stratify too, in practice it can lead to better results) of individual samples i at every step.

[3] Robbins and Monro, *"A Stochastic Approximation Method"*, 1951

Stochastic Gradient Descent (SGD)

It turns out that we don't quite need to compute the gradients $\nabla L(\theta)$ over the whole dataset at every iteration of Gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\nabla L_i(\theta^{(t)})}_{\text{Individual samples}}$$

where we do random sampling (or not, we can stratify too, in practice it can lead to better results) of individual samples i at every step.

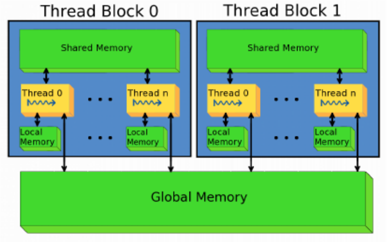
- Much more efficient (don't have to compute gradient for entire dataset);
- Noise (can be beneficial);
- Lots of redundancy on real datasets;
- Highly correlation at early steps (similar gradients SGD vs GD);

SGD can be traced back to 1950s work on the Robbins–Monro algorithm [3].

[3] Robbins and Monro, *"A Stochastic Approximation Method"*, 1951

Graphics Processing Unit (GPUs)

Most of the operations in Machine Learning ends up being lowered to GEMM (*General Matrix Multiplication*) and MAC (*Multiply–accumulate operation*) operations.
To leverage these massively parallel engines, we need to provide enough data to take advantage of the parallelization potential.



Source: Standard GPU memory hierarchy. By Giacomo Parigi.

Mini-batch SGD

That's why using mini-batches instead of individual samples on SGD is a perfect marriage of having better gradient estimates together with improved parallelization:

$$\tilde{\nabla} L(\theta^{(t)}) = \underbrace{\frac{1}{|B|}}_{\text{Batch size}} \sum_{i \in B} \nabla L_i(\theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\tilde{\nabla} L(\theta^{(t)})}_{\text{Estimated gradients}}$$

Mini-batch SGD

That's why using mini-batches instead of individual samples on SGD is a perfect marriage of having better gradient estimates together with improved parallelization:

$$\tilde{\nabla} L(\theta^{(t)}) = \underbrace{\frac{1}{|B|}}_{\text{Batch size}} \sum_{i \in B} \nabla L_i(\theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\tilde{\nabla} L(\theta^{(t)})}_{\text{Estimated gradients}}$$

If we do random sampling, then:

$$\underbrace{\mathbb{E}[\tilde{\nabla} L(\theta^{(t)})] = \nabla L(\theta)}_{\text{Unbiased estimate}}$$

Section III

Adaptation and Preconditioning

Adaptive Moment Estimation (Adam)

There are many adaptive methods, we will focus on one of the most frequently used in Deep Learning, the *Adaptive Moment Estimation* [4], also called **Adam**.

[4] Kingma and Ba, *"Adam: a Method for Stochastic Optimization"*, 2015

Adaptive Moment Estimation (Adam)

There are many adaptive methods, we will focus on one of the most frequently used in Deep Learning, the *Adaptive Moment Estimation* [4], also called **Adam**.

- Single learning rate for all parameters of the network doesn't seem to be enough to cope with the growing complexity of Deep Learning architectures;

[4] Kingma and Ba, *"Adam: a Method for Stochastic Optimization"*, 2015

Adaptive Moment Estimation (Adam)

There are many adaptive methods, we will focus on one of the most frequently used in Deep Learning, the *Adaptive Moment Estimation* [4], also called **Adam**.

- Single learning rate for all parameters of the network doesn't seem to be enough to cope with the growing complexity of Deep Learning architectures;
- What we can do is often bounded by what we can optimize, therefore better optimization techniques that explores structure are paramount;

[4] Kingma and Ba, *"Adam: a Method for Stochastic Optimization"*, 2015

Adaptive Moment Estimation (Adam)

There are many adaptive methods, we will focus on one of the most frequently used in Deep Learning, the *Adaptive Moment Estimation* [4], also called **Adam**.

- Single learning rate for all parameters of the network doesn't seem to be enough to cope with the growing complexity of Deep Learning architectures;
- What we can do is often bounded by what we can optimize, therefore better optimization techniques that explores structure are paramount;
- Most of the adaptive methods adapt to some kind of structure or curvature of the optimization landscape;

[4] Kingma and Ba, *"Adam: a Method for Stochastic Optimization"*, 2015

Adaptive Moment Estimation (Adam)

There are many adaptive methods, we will focus on one of the most frequently used in Deep Learning, the *Adaptive Moment Estimation* [4], also called **Adam**.

- Single learning rate for all parameters of the network doesn't seem to be enough to cope with the growing complexity of Deep Learning architectures;
- What we can do is often bounded by what we can optimize, therefore better optimization techniques that explores structure are paramount;
- Most of the adaptive methods adapt to some kind of structure or curvature of the optimization landscape;
- Many of these algorithms are still not well understood, lots of folklore in the field;

[4] Kingma and Ba, *"Adam: a Method for Stochastic Optimization"*, 2015

Adaptive Moment Estimation (Adam)

There are many adaptive methods, we will focus on one of the most frequently used in Deep Learning, the *Adaptive Moment Estimation* [4], also called **Adam**.

- Single learning rate for all parameters of the network doesn't seem to be enough to cope with the growing complexity of Deep Learning architectures;
- What we can do is often bounded by what we can optimize, therefore better optimization techniques that explores structure are paramount;
- Most of the adaptive methods adapt to some kind of structure or curvature of the optimization landscape;
- Many of these algorithms are still not well understood, lots of folklore in the field;
- Will try to focus on building intuition from the original algorithm.

[4] Kingma and Ba, *"Adam: a Method for Stochastic Optimization"*, 2015

Adaptive Moment Estimation (Adam)

Algorithm $g_t^2 = g_t \odot g_t$. Good defaults: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. β_1^t and β_2^t are β_1 and β_2 to the power t.

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector, α: Stepsize
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Adaptive Moment Estimation (Adam)

Lots of things going on here, let's focus on how moments are being computed and neglect bias correction and initialization:

$$
\begin{aligned}
g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1}) \\
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2
\end{aligned}
$$

And the parameter updates:

$$
\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}
$$

Adaptive Moment Estimation (Adam)

Lots of things going on here, let's focus on how moments are being computed and neglect bias correction and initialization:

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$$
$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

And the parameter updates:

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}$$

- Do you recognize m_t ?
- What happens when the uncentered variance grows ?

The good, the bad, and the Hessian

- The convergence rate of Gradient descent is deeply connected to the curvature of the landscape it is trying to optimize;

The good, the bad, and the Hessian

- The convergence rate of Gradient descent is deeply connected to the curvature of the landscape it is trying to optimize;
- The Hessian matrix $\mathbf{H}_f$ carries information about the curvature, therefore we usually use it understand problems or even make them better conditioned;

The good, the bad, and the Hessian

- The convergence rate of Gradient descent is deeply connected to the curvature of the landscape it is trying to optimize;
- The Hessian matrix $\mathbf{H}_f$ carries information about the curvature, therefore we usually use it understand problems or even make them better conditioned;
- The $\mathbf{H}_f$ is often very costly to compute for real-life problems, therefore much of the work rely on approximating it or computing information about it without having to materialize the entire matrix;

Hessian

The $\mathbf{H}_f$ is a square matrix of 2nd-order partial derivatives. Let's compute the $\mathbf{H}_f$ of $f(x,y) = x^2y + xy^3$, starting with first-order:

$$\frac{\partial f}{\partial x} = 2xy + y^3 \quad , \quad \frac{\partial f}{\partial y} = x^2 + 3xy^2$$

Note that the $\mathbf{H}_f$ can be constant and not depend on variables or depend only on some of them. We will see this case later.

Hessian

The $\mathbf{H}_f$ is a square matrix of 2nd-order partial derivatives. Let's compute the $\mathbf{H}_f$ of $f(x, y) = x^2y + xy^3$, starting with first-order:

$$\frac{\partial f}{\partial x} = 2xy + y^3 \quad , \quad \frac{\partial f}{\partial y} = x^2 + 3xy^2$$

Second order

$$\frac{\partial^2 f}{\partial x^2} = 2y \quad , \quad \frac{\partial^2 f}{\partial y \partial x} 2x + 3y^2 \quad , \quad \frac{\partial^2 f}{\partial x \partial y} = 2x + 3y^2 \quad , \quad \frac{\partial^2 f}{\partial y^2} = 6xy$$

Note that the $\mathbf{H}_f$ can be constant and not depend on variables or depend only on some of them. We will see this case later.

Hessian

The $\mathbf{H}_f$ is a square matrix of 2nd-order partial derivatives. Let's compute the $\mathbf{H}_f$ of $f(x,y) = x^2y + xy^3$, starting with first-order:

$$\frac{\partial f}{\partial x} = 2xy + y^3 \quad , \quad \frac{\partial f}{\partial y} = x^2 + 3xy^2$$

Second order

$$\frac{\partial^2 f}{\partial x^2} = 2y \quad , \quad \frac{\partial^2 f}{\partial y \partial x} 2x + 3y^2 \quad , \quad \frac{\partial^2 f}{\partial x \partial y} = 2x + 3y^2 \quad , \quad \frac{\partial^2 f}{\partial y^2} = 6xy$$

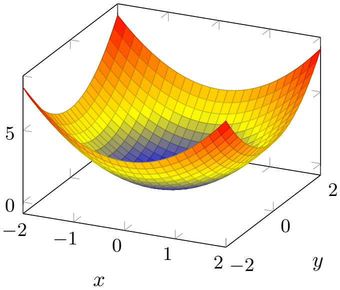
Hessian

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial y \partial x} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2y & 2x + 3y^2 \\ 2x + 3y^2 & 6xy \end{bmatrix}$$

Note that the $\mathbf{H}_f$ can be constant and not depend on variables or depend only on some of them. We will see this case later.

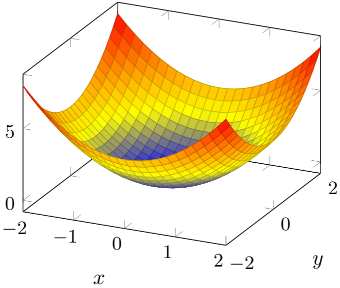
Hessian Eigenvalues

All positive eigenvalues
(positive definite)

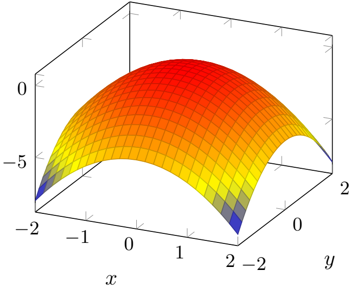


Hessian Eigenvalues

All positive eigenvalues
(positive definite)



All negative eigenvalues
(negative definite)



Condition Number

The **Condition number**, also defined as κ, is the ratio of maximum and minimum eigenvalues (λ_{max} and λ_{min}) of the Hessian $\mathbf{H}_f$:

$$\kappa = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$$

Condition Number

The **Condition number**, also defined as κ, is the ratio of maximum and minimum eigenvalues ($\lambda_{\max}$ and $\lambda_{\min}$) of the Hessian $\mathbf{H}_f$:

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$

- When κ is high we say that the problem is ill-conditioned;

Condition Number

The **Condition number**, also defined as κ, is the ratio of maximum and minimum eigenvalues ($\lambda_{\max}$ and $\lambda_{\min}$) of the Hessian $\mathbf{H}_f$:

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$

- When κ is high we say that the problem is ill-conditioned;
- Steepest descent convergence rate is *slow* for ill-conditioned problems;

Condition Number

The **Condition number**, also defined as κ, is the ratio of maximum and minimum eigenvalues (λ_{max} and λ_{min}) of the Hessian $\mathbf{H}_f$:

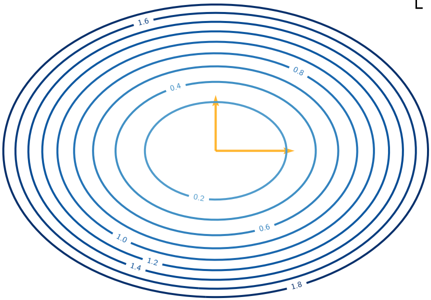
$$\kappa = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$$

- When κ is high we say that the problem is ill-conditioned;
- Steepest descent convergence rate is *slow* for ill-conditioned problems;
- Let's understand it on a quadratic problem to gain intuition.

Condition Number

$$f(\theta) = \frac{1.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

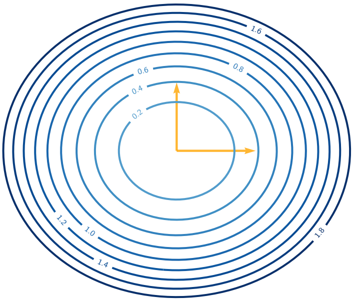


$$\kappa = 2.00\ (\lambda_{max} = 2.0, \lambda_{min} = 1.0)$$

Condition Number

$$f(\theta) = \frac{1.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 1.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

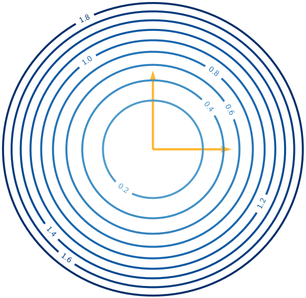


$$\kappa = 1.33 \; (\lambda_{max} = 2.0, \lambda_{min} = 1.5)$$

Condition Number

$$f(\theta) = \frac{2.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 2.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

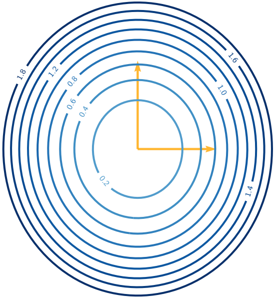


$$\kappa = 1.00 \ (\lambda_{max} = 2.0, \lambda_{min} = 2.0)$$

Condition Number

$$f(\theta) = \frac{2.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 2.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

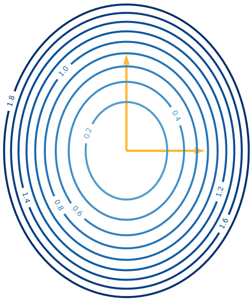


$$\kappa = 1.25 \; (\lambda_{max} = 2.5, \lambda_{min} = 2.0)$$

Condition Number

$$f(\theta) = \frac{3.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 3.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

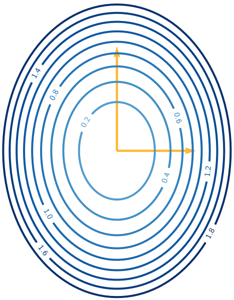


$$\kappa = 1.50 \; (\lambda_{max} = 3.0, \lambda_{min} = 2.0)$$

Condition Number

$$f(\theta) = \frac{3.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 3.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

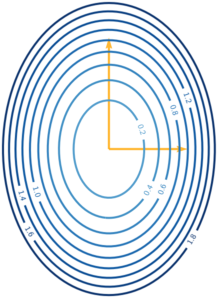


$$\kappa = 1.75 \ (\lambda_{max} = 3.5, \lambda_{min} = 2.0)$$

Condition Number

$$f(\theta) = \frac{4.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 4.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

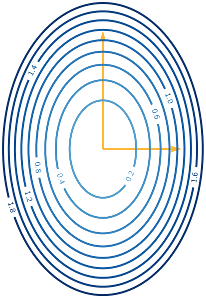


$$\kappa = 2.00 \ (\lambda_{max} = 4.0, \lambda_{min} = 2.0)$$

Condition Number

$$f(\theta) = \frac{4.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 4.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

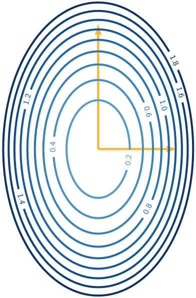


$$\kappa = 2.25 \; (\lambda_{max} = 4.5, \lambda_{min} = 2.0)$$

Condition Number

$$f(\theta) = \frac{5.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 5.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

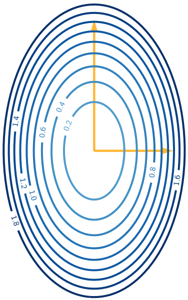


$$\kappa = 2.50 \; (\lambda_{max} = 5.0, \lambda_{min} = 2.0)$$

Condition Number

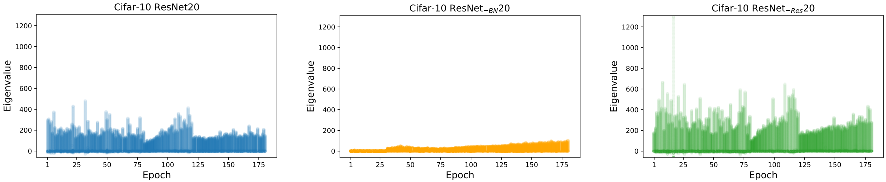
$$f(\theta) = \frac{5.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 5.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$



$$\kappa = 2.75 \ (\lambda_{max} = 5.5, \lambda_{min} = 2.0)$$

Hessian eigenvalue spectral density (ESD)



Source: Yao, Z., Gholami, A., Keutzer, K., & Mahoney, M. W. (2019, December 15). PYHESSIAN: Neural networks through the lens of the hessian.

ResNet with depth 20 trained on Cifar-10. ResNet$_{_BN}$ is the ResNet without Batch Normalization and the ResNet$_{_Res}$ is without the residual connections. In [6], they also show that the distribution seem to composed of two parts: the bulk around zero, and the edges scattered away from zero.

[6] Sagun, Leon Bottou, and LeCun, *"Eigenvalues of the Hessian in Deep Learning: Singularity and Beyond"*, 2016

Preconditioning

From Adam's original paper:

(…) Like natural gradient descent (NGD)[7], Adam employs a **preconditioner** that adapts to the **geometry of the data**, since $\widehat{v}_t$ is an approximation to the **diagonal of the Fisher information matrix**[8]; (…)

- Preconditioning can be viewed as a change in the geometry;

[7] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[8] Pascanu and Bengio, *"Revisiting Natural Gradient for Deep Networks"*, 2013

Preconditioning

From Adam's original paper:

(…) Like natural gradient descent (NGD) [7], Adam employs a **preconditioner** that adapts to the **geometry of the data**, since $\widehat{v}_t$ is an approximation to the **diagonal of the Fisher information matrix** [8]; (…)

- Preconditioning can be viewed as a change in the geometry;
- It can help with poorly conditioned problems;

[7] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998
[8] Pascanu and Bengio, *"Revisiting Natural Gradient for Deep Networks"*, 2013

Preconditioning

From Adam's original paper:

(…) Like natural gradient descent (NGD) [7], Adam employs a **preconditioner** that adapts to the **geometry of the data**, since $\widehat{v}_t$ is an approximation to the **diagonal of the Fisher information matrix** [8]; (…)

- Preconditioning can be viewed as a change in the geometry;
- It can help with poorly conditioned problems;
- We will talk about the Fisher Information Matrix (FIM) later;

[7] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[8] Pascanu and Bengio, *"Revisiting Natural Gradient for Deep Networks"*, 2013

Preconditioning

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(\theta^{(t)})$$

Preconditioning

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

Preconditioning

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{P}_{\text{Preconditioner}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

Preconditioning

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{I}_{\text{Identity}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

Preconditioning

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Identity}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

Preconditioning

$$\theta^{(t+1)} = \theta^{(t)} - \underbrace{\mathbf{H}_L^{-1}}_{\text{Hessian}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

- Can be interpreted as an iterative minimization of the quadratic approximation, we're using a 2nd-order term here, remember the Taylor approximation ?

The superscript t was omitted from the $\mathbf{H}_L^{-1}$ for clarity.

Preconditioning

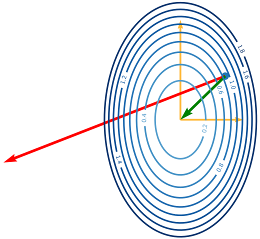
$$\theta^{(t+1)} = \theta^{(t)} - \underbrace{(\mathbf{H}_L + \lambda I)^{-1}}_{\text{Damped Hessian}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

The superscript t was omitted from the $\mathbf{H}_L$ for clarity.

Preconditioning

$$f(\theta) = \frac{5.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 5.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

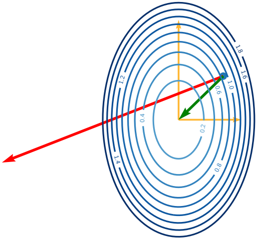


$\kappa = 2.50\ (\lambda_{max} = 5.0, \lambda_{min} = 2.0)$

Preconditioning

$$f(\theta) = \frac{5.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 5.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$



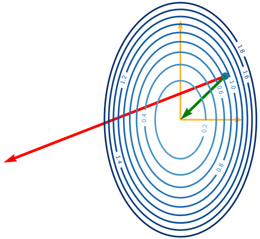
$\kappa = 2.50 \; (\lambda_{max} = 5.0, \lambda_{min} = 2.0)$

Let's think about what the preconditioner is doing in this situation, we have a point $\theta \in \mathbb{R}^2$ at $\theta = (0.5, 0.5)$ and we have that:

Preconditioning

$f(\theta) = \frac{5.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$

$\mathbf{H}_f \begin{bmatrix} 5.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$



$\kappa = 2.50 \; (\lambda_{max} = 5.0, \lambda_{min} = 2.0)$

Let's think about what the preconditioner is doing in this situation, we have a point $\theta \in \mathbb{R}^2$ at $\theta = (0.5, 0.5)$ and we have that:

$$\nabla f(\theta) = (2.5, 1.0)$$

$$\mathbf{H}_f = \begin{bmatrix} 5.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

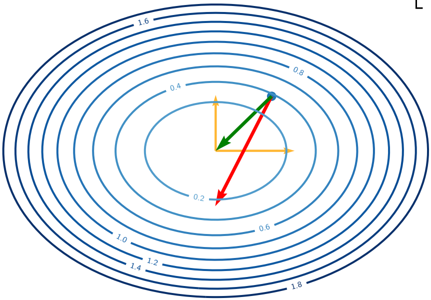
$$\theta - \mathbf{H}_L^{-1} \nabla f(\theta) = (0., 0.)$$

$$\theta - \nabla f(\theta) = (-2., -0.5)$$

Hessian as preconditioner

$$f(\theta) = \frac{1.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

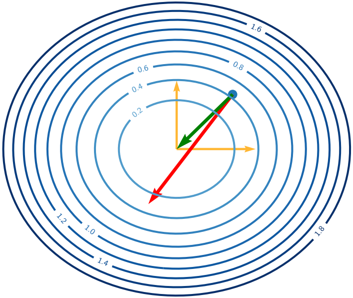


$$\kappa = 2.00\ (\lambda_{max} = 2.0, \lambda_{min} = 1.0)$$

Hessian as preconditioner

$$f(\theta) = \frac{1.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 1.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

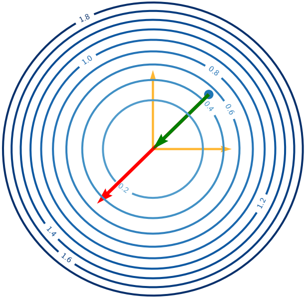


$$\kappa = 1.33\ (\lambda_{max} = 2.0, \lambda_{min} = 1.5)$$

Hessian as preconditioner

$$f(\theta) = \frac{2.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 2.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

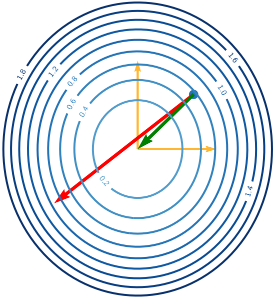


$$\kappa = 1.00 \ (\lambda_{max} = 2.0, \lambda_{min} = 2.0)$$

Hessian as preconditioner

$$f(\theta) = \frac{2.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 2.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

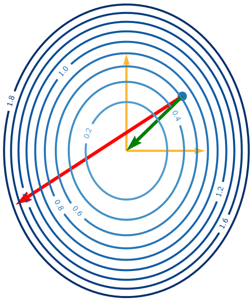


$$\kappa = 1.25 \; (\lambda_{max} = 2.5, \lambda_{min} = 2.0)$$

Hessian as preconditioner

$$f(\theta) = \frac{3.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 3.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

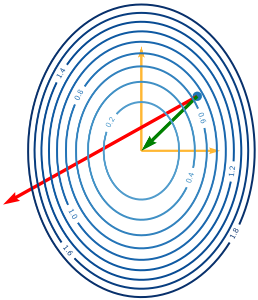


$$\kappa = 1.50 \ (\lambda_{max} = 3.0, \lambda_{min} = 2.0)$$

Hessian as preconditioner

$$f(\theta) = \frac{3.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 3.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

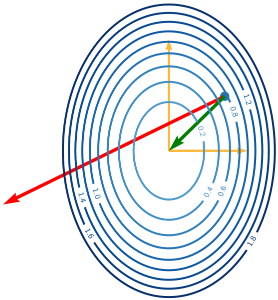


$$\kappa = 1.75 \ (\lambda_{max} = 3.5, \lambda_{min} = 2.0)$$

Hessian as preconditioner

$$f(\theta) = \frac{4.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 4.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

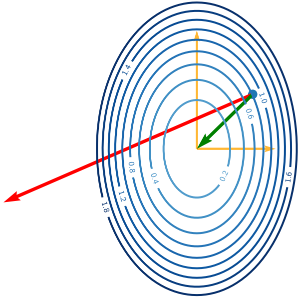


$$\kappa = 2.00 \; (\lambda_{max} = 4.0, \lambda_{min} = 2.0)$$

Hessian as preconditioner

$$f(\theta) = \frac{4.5}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 4.5 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

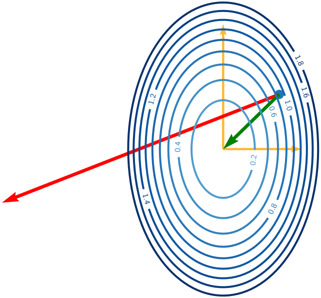


$$\kappa = 2.25 \; (\lambda_{max} = 4.5, \lambda_{min} = 2.0)$$

Hessian as preconditioner

$$f(\theta) = \frac{5.0}{2.0}\theta_1 + \frac{2.0}{2.0}\theta_2$$

$$\mathbf{H}_f \begin{bmatrix} 5.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$



$$\kappa = 2.50 \ (\lambda_{max} = 5.0, \lambda_{min} = 2.0)$$

Difficulties of the Hessian preconditioning

- Using the Hessian as preconditioner is the basis of the Newton's method;

[11] Dauphin et al., *"Identifying and attacking the saddle point problem in high-dimensional non-convex optimization"*, 2014

[12] Yao et al., *PYHESSIAN: Neural networks through the lens of the hessian*, 2019

[13] Martens, *Deep learning via Hessian-free optimization*, 2010

Difficulties of the Hessian preconditioning

- Using the Hessian as preconditioner is the basis of the Newton's method;
- Invariant to affine transformations;

[11] Dauphin et al., *"Identifying and attacking the saddle point problem in high-dimensional non-convex optimization"*, 2014

[12] Yao et al., *PYHESSIAN: Neural networks through the lens of the hessian*, 2019

[13] Martens, *Deep learning via Hessian-free optimization*, 2010

Difficulties of the Hessian preconditioning

- Using the Hessian as preconditioner is the basis of the Newton's method;
- Invariant to affine transformations;
- However, a model with 23 million parameters (i.e. ResNet-50), what is the space complexity to store the $\mathbf{H}_f$ and the computational complexity to invert it ?

[11] Dauphin et al., *"Identifying and attacking the saddle point problem in high-dimensional non-convex optimization"*, 2014

[12] Yao et al., *PYHESSIAN: Neural networks through the lens of the hessian*, 2019

[13] Martens, *Deep learning via Hessian-free optimization*, 2010

Difficulties of the Hessian preconditioning

- Using the Hessian as preconditioner is the basis of the Newton's method;
- Invariant to affine transformations;
- However, a model with 23 million parameters (i.e. ResNet-50), what is the space complexity to store the $\mathbf{H}_f$ and the computational complexity to invert it ?
- Difficult on non-convex problems, not always invertible, attracted by saddle points [11];

[11] Dauphin et al., *"Identifying and attacking the saddle point problem in high-dimensional non-convex optimization"*, 2014

[12] Yao et al., *PYHESSIAN: Neural networks through the lens of the hessian*, 2019

[13] Martens, *Deep learning via Hessian-free optimization*, 2010

Difficulties of the Hessian preconditioning

- Using the Hessian as preconditioner is the basis of the Newton's method;
- Invariant to affine transformations;
- However, a model with 23 million parameters (i.e. ResNet-50), what is the space complexity to store the $\mathbf{H}_f$ and the computational complexity to invert it ?
- Difficult on non-convex problems, not always invertible, attracted by saddle points [11];
- Among other reasons, you now understand all the efforts into Hessian approximations [12], alternative curvature matrices and hessian-free optimization [13].

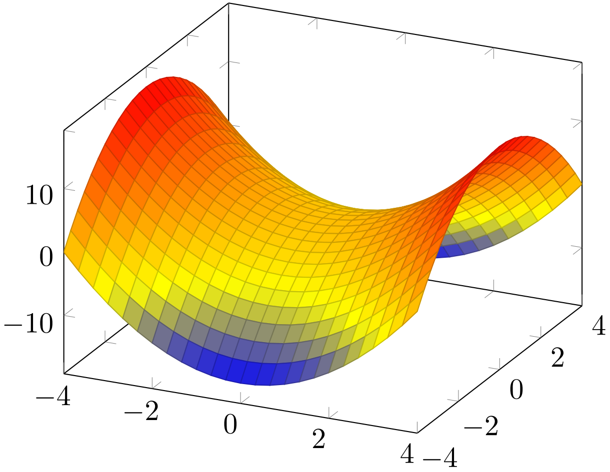
---

[11] Dauphin et al., *"Identifying and attacking the saddle point problem in high-dimensional non-convex optimization"*, 2014

[12] Yao et al., *PYHESSIAN: Neural networks through the lens of the hessian*, 2019

[13] Martens, *Deep learning via Hessian-free optimization*, 2010

Saddle points



Fisher Information Matrix (FIM)

Going back to the Adam's article:

From Adam's original paper:

(...) Like natural gradient descent (NGD) [14], Adam employs a **preconditioner** that adapts to the **geometry of the data**, since $\widehat{v}_t$ is an approximation to the **diagonal of the Fisher information matrix** [15]; (...)

- We now know what a preconditioner means;
- The missing ingredient now is the **Fisher Information Matrix** (also known as FIM).

[14] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[15] Pascanu and Bengio, *"Revisiting Natural Gradient for Deep Networks"*, 2013

Fisher Information Matrix (FIM)

The Fisher Information Matrix is the covariance of the score function (gradients of the log-likelihood function) with expectation over the *model's predictive distribution* (pay attention to this detail).

Definition: Fisher Information Matrix

$$\mathbf{F}_\theta = \mathop{\mathbb{E}}_{\substack{y \sim p_\theta(y|x) \\ x \sim p_{\text{data}}}} \left[\nabla_\theta \log p_\theta(y|x) \, \nabla_\theta \log p_\theta(y|x)^\intercal \right]$$

Where $\mathbf{F}_\theta \in \mathbb{R}^{n \times n}$.

Fisher Information Matrix (FIM)

The Fisher Information Matrix is the covariance of the score function (gradients of the log-likelihood function) with expectation over the *model's predictive distribution* (pay attention to this detail).

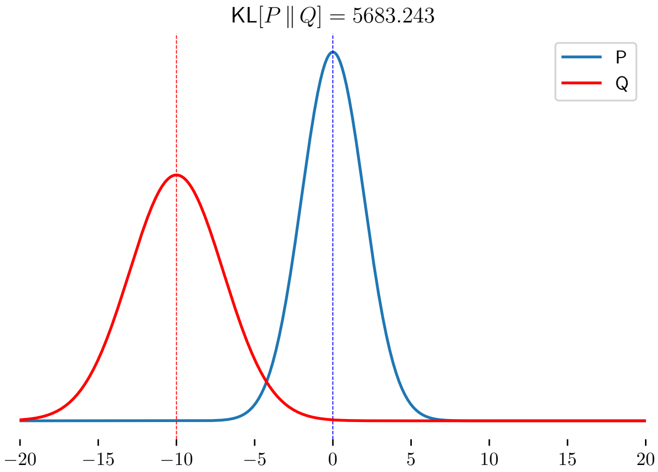
Definition: Fisher Information Matrix

$$\mathbf{F}_\theta = \mathop{\mathbb{E}}_{\substack{y \sim p_\theta(y|x) \\ x \sim p_{\text{data}}}} \left[\nabla_\theta \log p_\theta(y|x) \, \nabla_\theta \log p_\theta(y|x)^\intercal \right]$$

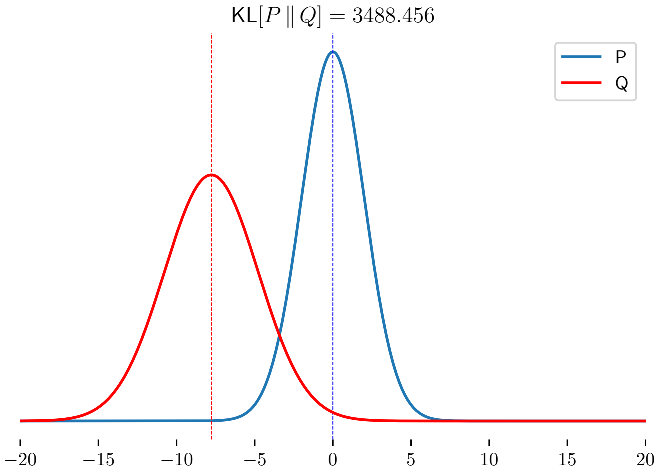
Where $\mathbf{F}_\theta \in \mathbb{R}^{n \times n}$. We often approximate it using input samples (y is still from model's predictive distribution), as we don't have access to p_{data}:

$$\mathbf{F}_\theta = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(y|x_i) \, \nabla_\theta \log p_\theta(y|x_i)^\intercal$$

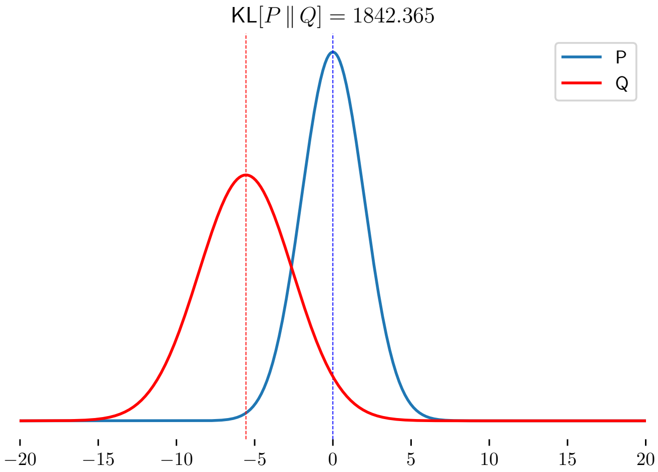
Kullback-Leibler divergence



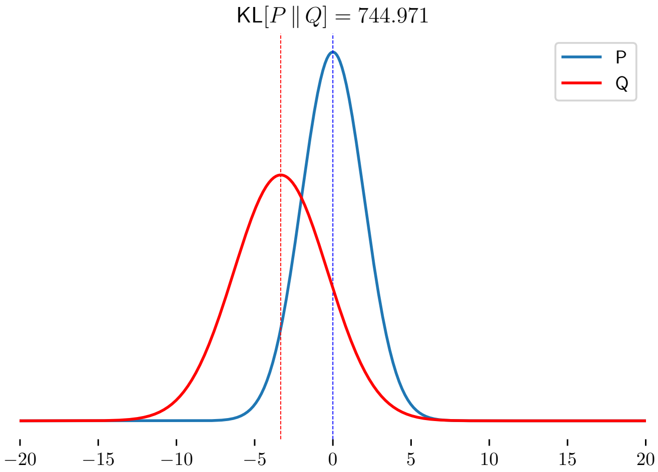
Kullback-Leibler divergence



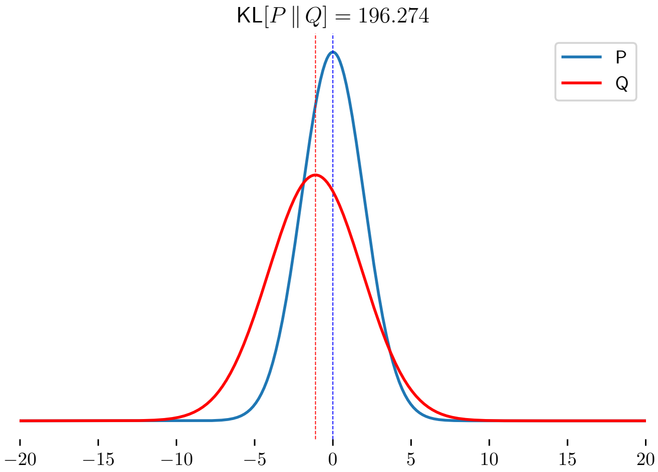
Kullback-Leibler divergence



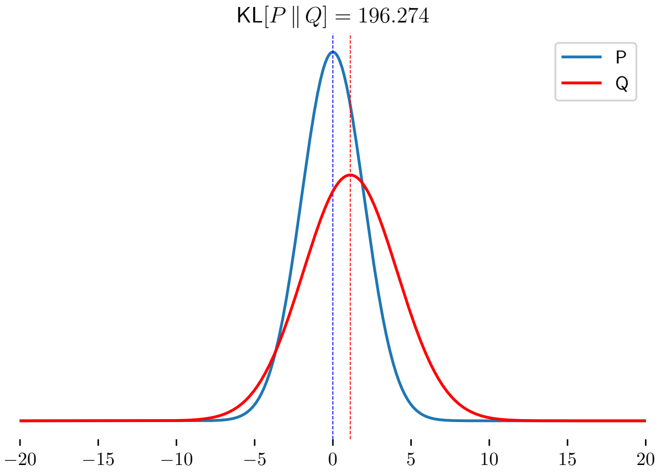
Kullback-Leibler divergence



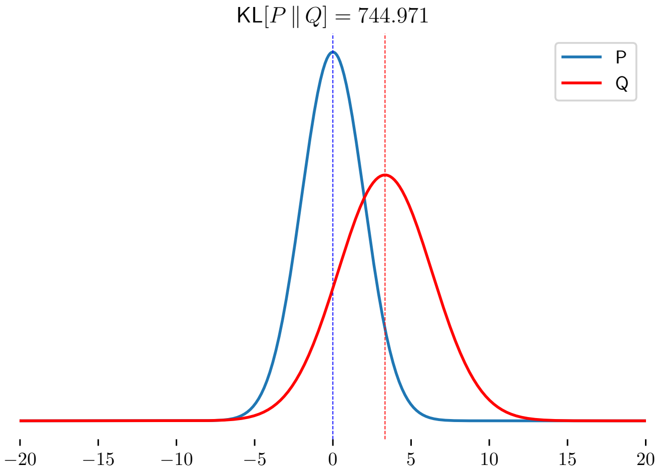
Kullback-Leibler divergence



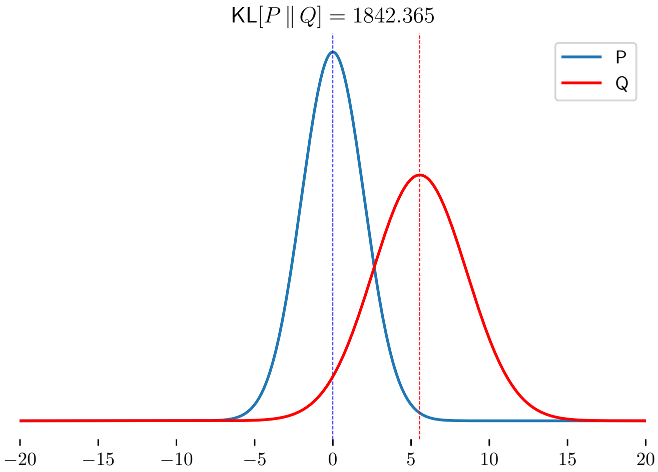
Kullback-Leibler divergence



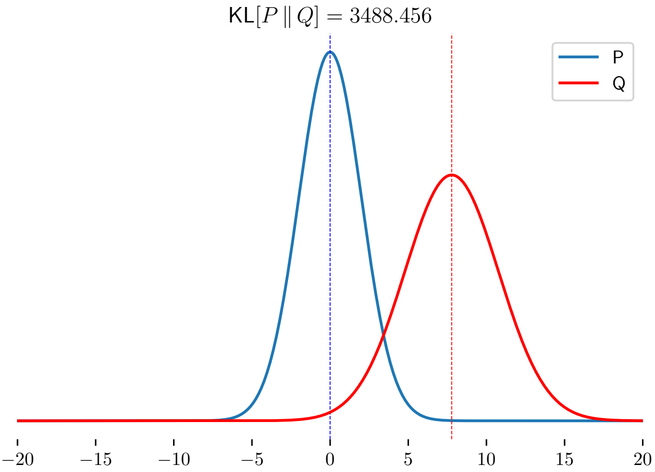
Kullback-Leibler divergence



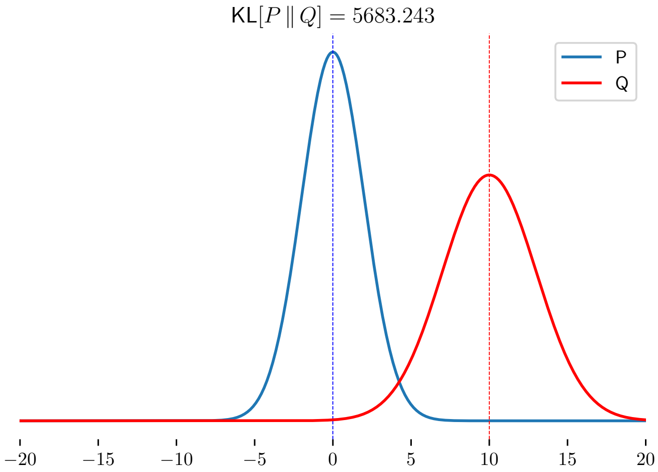
Kullback-Leibler divergence



Kullback-Leibler divergence



Kullback-Leibler divergence



Fisher Information Matrix (FIM)

- We can parametrize the same distribution family on many different ways;
- Moving in the parameter space using the Euclidean distance as a metric makes us tied to the particular parametrization;

[16] For a full derivation please refer to: Ratliff, N. (2013). Information Geometry and Natural Gradients.

[17] Martens, *"New insights and perspectives on the natural gradient method"*, 2014

Fisher Information Matrix (FIM)

- We can parametrize the same distribution family on many different ways;
- Moving in the parameter space using the Euclidean distance as a metric makes us tied to the particular parametrization;
- One interesting way is to move on the distribution space, to which the KL divergence makes more sense;

[16] For a full derivation please refer to: Ratliff, N. (2013). Information Geometry and Natural Gradients.

[17] Martens, *"New insights and perspectives on the natural gradient method"*, 2014

Fisher Information Matrix (FIM)

- We can parametrize the same distribution family on many different ways;
- Moving in the parameter space using the Euclidean distance as a metric makes us tied to the particular parametrization;
- One interesting way is to move on the distribution space, to which the KL divergence makes more sense;
- It turns out that the second-order Taylor approximation to the KL divergence is the Fisher Information Matrix[16];

[16] For a full derivation please refer to: Ratliff, N. (2013). Information Geometry and Natural Gradients.

[17] Martens, *"New insights and perspectives on the natural gradient method"*, 2014

Fisher Information Matrix (FIM)

- We can parametrize the same distribution family on many different ways;
- Moving in the parameter space using the Euclidean distance as a metric makes us tied to the particular parametrization;
- One interesting way is to move on the distribution space, to which the KL divergence makes more sense;
- It turns out that the second-order Taylor approximation to the KL divergence is the Fisher Information Matrix[16];
- We won't be talking here, but the Fisher has a strong connection to the Hessian and the Generalized Gauss-Newton (GGN), please refer to [17] if you are interested.

[16] For a full derivation please refer to: Ratliff, N. (2013). Information Geometry and Natural Gradients.

[17] Martens, *"New insights and perspectives on the natural gradient method"*, 2014

Section IV

Natural Gradient

Natural Gradient

When we do a preconditioning on Gradient descent using the Fisher, we have the Natural Gradient Descent [18]:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\mathbf{F}_{\theta}^{-1}}_{\text{FIM}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

[18] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[19] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

When we do a preconditioning on Gradient descent using the Fisher, we have the Natural Gradient Descent [18]:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\mathbf{F}_{\theta}^{-1}}_{\text{FIM}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

- It converges much faster than ordinary Gradient descent;

[18] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[19] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

When we do a preconditioning on Gradient descent using the Fisher, we have the Natural Gradient Descent [18]:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\mathbf{F}_\theta^{-1}}_{\text{FIM}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

- It converges much faster than ordinary Gradient descent;
- It moves on the distribution space manifold, invariant with respect to all differentiable and invertible transformations [19];

---

[18] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[19] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

When we do a preconditioning on Gradient descent using the Fisher, we have the Natural Gradient Descent [18]:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\mathbf{F}_{\theta}^{-1}}_{\text{FIM}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

- It converges much faster than ordinary Gradient descent;
- It moves on the distribution space manifold, invariant with respect to all differentiable and invertible transformations [19];
- Given that the FIM is the result of an outer-product, it is always PSD (positive semidefinite matrix);

[18] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[19] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

When we do a preconditioning on Gradient descent using the Fisher, we have the Natural Gradient Descent [18]:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \underbrace{\mathbf{F}_\theta^{-1}}_{\text{FIM}} \underbrace{\nabla L(\theta^{(t)})}_{\text{Gradients}}$$

- It converges much faster than ordinary Gradient descent;
- It moves on the distribution space manifold, invariant with respect to all differentiable and invertible transformations [19];
- Given that the FIM is the result of an outer-product, it is always PSD (positive semidefinite matrix);
- It is still a $n \times n$ matrix, that needs to be inverted;

[18] Amari, *"Natural Gradient Works Efficiently in Learning"*, 1998

[19] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

The natural gradient is connected to *information geometry* [20].

- In a **Euclidean space**, the shortest path between two points is always the straight line;

[20] Amari, *"Information geometry and its applications"*, 2016
[21] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

The natural gradient is connected to *information geometry* [20].

- In a **Euclidean space**, the shortest path between two points is always the straight line;
- In a **Riemannian space**, the shortest path between two points (minimal geodesic) can have a curvature and sometimes there is more than one between two points;

[20] Amari, *"Information geometry and its applications"*, 2016

[21] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

The natural gradient is connected to *information geometry* [20].

- In a **Euclidean space**, the shortest path between two points is always the straight line;
- In a **Riemannian space**, the shortest path between two points (minimal geodesic) can have a curvature and sometimes there is more than one between two points;
- The **metric tensor** represents this curvature and can be different at different points;

[20] Amari, *"Information geometry and its applications"*, 2016

[21] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

The natural gradient is connected to *information geometry* [20].

- In a **Euclidean space**, the shortest path between two points is always the straight line;
- In a **Riemannian space**, the shortest path between two points (minimal geodesic) can have a curvature and sometimes there is more than one between two points;
- The **metric tensor** represents this curvature and can be different at different points;
- With the natural gradient, we are moving in this Riemannian manifold using the Fisher as the metric tensor;

[20] Amari, *"Information geometry and its applications"*, 2016

[21] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

Natural Gradient

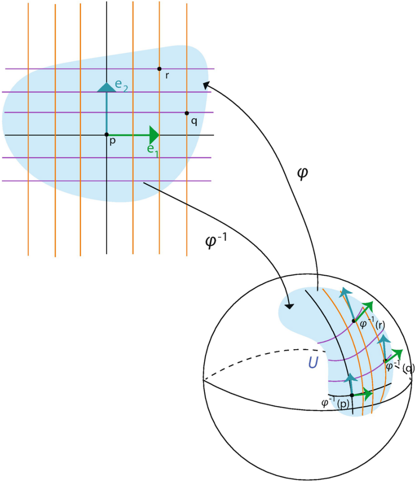
The natural gradient is connected to *information geometry* [20].

- In a **Euclidean space**, the shortest path between two points is always the straight line;
- In a **Riemannian space**, the shortest path between two points (minimal geodesic) can have a curvature and sometimes there is more than one between two points;
- The **metric tensor** represents this curvature and can be different at different points;
- With the natural gradient, we are moving in this Riemannian manifold using the Fisher as the metric tensor;
- Parameters move more quickly along directions that have a small impact on the decision function, and more cautiously along directions that have a large impact [21];

[20] Amari, *"Information geometry and its applications"*, 2016

[21] Léon Bottou, Curtis, and Nocedal, *Optimization methods for large-scale machine learning*, 2018

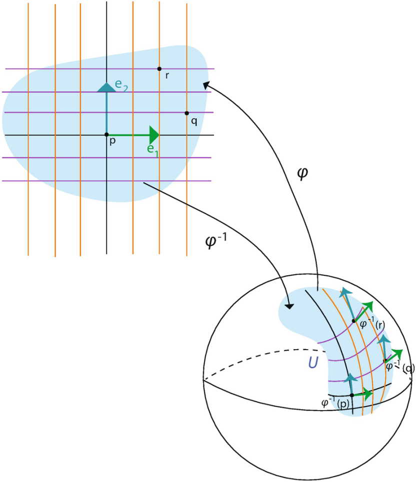
Riemannian manifold



Source: Gallier J., (2020) Advanced Geometric Methods in Computer Science. CIS 610, Spring 2018.

- A manifold is a collection of points, where locally (but not globally), is Euclidean;

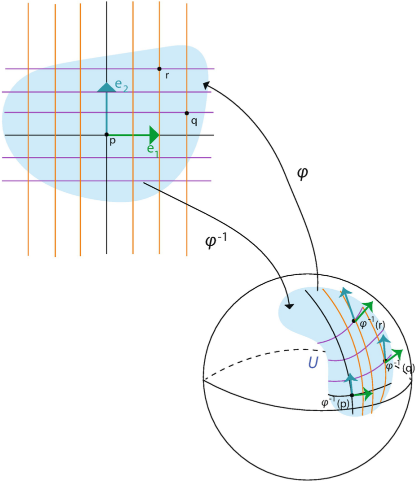
Riemannian manifold



Source: Gallier J., (2020) Advanced Geometric Methods in Computer Science. CIS 610, Spring 2018.

- A manifold is a collection of points, where locally (but not globally), is Euclidean;
- A metric induces an inner product on the tangent space at each point on the manifold;

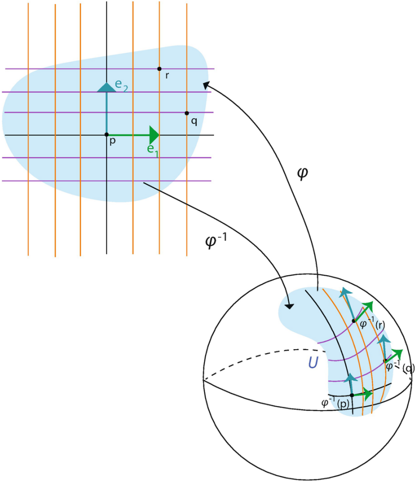
Riemannian manifold



Source: Gallier J., (2020) Advanced Geometric Methods in Computer Science. CIS 610, Spring 2018.

- A manifold is a collection of points, where locally (but not globally), is Euclidean;
- A metric induces an inner product on the tangent space at each point on the manifold;
- The metric on the statistical manifold is *unique*, it is an *intrinsic* geometry;

Riemannian manifold



Source: Gallier J., (2020) Advanced Geometric Methods in Computer Science. CIS 610, Spring 2018.

- A manifold is a collection of points, where locally (but not globally), is Euclidean;
- A metric induces an inner product on the tangent space at each point on the manifold;
- The metric on the statistical manifold is *unique*, it is an *intrinsic* geometry;
- In Euclidean space we don't care because the metric is constant everywhere;

Empirical Fisher

There is a lot of confusion[22] about the Fisher Information Matrix [23].

- In some scenarios you will see people sampling $y \sim p_{\text{data}}$ too instead of sampling from the model's predictive distribution $y \sim p_\theta(y|x)$;

[22] I blame evil people who omit expectation qualifiers about where y is coming from.

[23] Kunstner, Balles, and Hennig, *"Limitations of the empirical fisher approximation for natural gradient descent"*, 2019

Empirical Fisher

There is a lot of confusion[22] about the Fisher Information Matrix [23].

- In some scenarios you will see people sampling $y \sim p_{\text{data}}$ too instead of sampling from the model's predictive distribution $y \sim p_\theta(y|x)$;
- This is called the Empirical Fisher, Empirical FIM or just EF:

$$\widetilde{\mathbf{F}}_\theta = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(y_i|x_i)\, \nabla_\theta \log p_\theta(y_i|x_i)^\intercal$$

[22] I blame evil people who omit expectation qualifiers about where y is coming from.

[23] Kunstner, Balles, and Hennig, *"Limitations of the empirical fisher approximation for natural gradient descent"*, 2019

Empirical Fisher

There is a lot of confusion[22] about the Fisher Information Matrix [23].

- In some scenarios you will see people sampling $y \sim p_{\text{data}}$ too instead of sampling from the model's predictive distribution $y \sim p_\theta(y|x)$;
- This is called the Empirical Fisher, Empirical FIM or just EF:

$$\widetilde{\mathbf{F}}_\theta = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(y_i|x_i) \, \nabla_\theta \log p_\theta(y_i|x_i)^\intercal$$

- It turns out that Adam is using the Empirical Fisher, and to make things more confusing it is using the square root of it.

[22] I blame evil people who omit expectation qualifiers about where y is coming from.

[23] Kunstner, Balles, and Hennig, *"Limitations of the empirical fisher approximation for natural gradient descent"*, 2019

Adam and the Natural Gradient Descent

Original Adam paper [24] claims that Adam is an approximation to the natural gradient descent (diagonal of the FIM):

$$
\begin{aligned}
g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1}) \\
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \\
\theta_t &\leftarrow \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}
\end{aligned}
$$

[24] Kingma and Ba, *“Adam: a Method for Stochastic Optimization”*, 2015

[25] Staib et al., *“Escaping saddle points with adaptive gradient methods”*, 2019

Adam and the Natural Gradient Descent

Original Adam paper [24] claims that Adam is an approximation to the natural gradient descent (diagonal of the FIM):

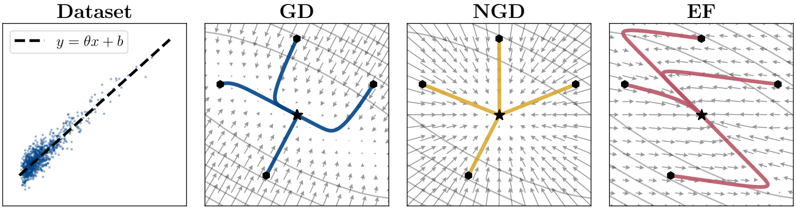
$$
\begin{aligned}
g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1}) \\
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \\
\theta_t &\leftarrow \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}
\end{aligned}
$$

However, the approximation is only valid near optimality (why ?). The exponent is also different, since Adam is taking square root, it doesn't change direction of the descent (only stepsize) [25].

[24] Kingma and Ba, *"Adam: a Method for Stochastic Optimization"*, 2015

[25] Staib et al., *"Escaping saddle points with adaptive gradient methods"*, 2019

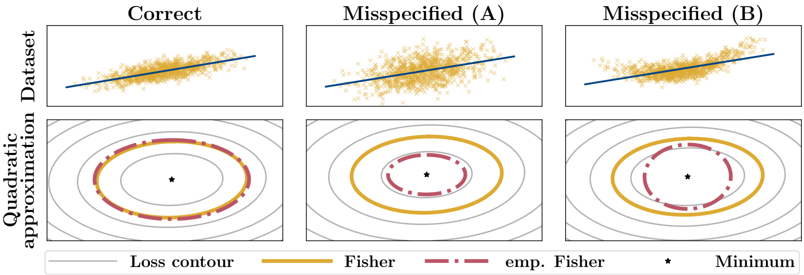
Empirical Fisher



Source: Kunstner, F., Balles, L., & Hennig, P. Limitations of the Empirical Fisher Approximation for Natural Gradient Descent. 2019. https://arxiv.org/abs/1905.12558.

- Vector fields of the gradients conditioned using the FIM vs using the EF are very different;
- Are they close to each other close to the minima ?

Empirical Fisher



Source: Kunstner, F., Balles, L., & Hennig, P. Limitations of the Empirical Fisher Approximation for Natural Gradient Descent. 2019. https://arxiv.org/abs/1905.12558.

- EF is a good approximation of the Fisher at the minimum if model is well-specified. Otherwise, even at the minimum and with a large amount of samples, it can be a very poor approximation [26];
- Is EF just the non-central gradient covariance matrix, working as variance reduction instead of curvature adaptation ?

[26] Kunstner, Balles, and Hennig, *"Limitations of the empirical fisher approximation for natural gradient descent"*, 2019

The epsilon that might not be an epsilon

Many implementations use the epsilon to avoid division by zero:

$$
\begin{aligned}
g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1}) \\
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \\
\theta_t &\leftarrow \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}
\end{aligned}
$$

[27] Choi et al., *On empirical comparisons of optimizers for deep learning*, 2019

The epsilon that might not be an epsilon

Many implementations use the epsilon to avoid division by zero:

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$$
$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$$
$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$$
$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}$$

However, remember about the damping mechanism ? The ϵ can be seen as setting a trust region radius [27].

[27] Choi et al., *On empirical comparisons of optimizers for deep learning*, 2019

Fisher is a big Fisher

Computing the inverse of the diagonal Fisher is easy, but computing the inverse of the "full" Fisher $\mathbf{F}^{-1}$ and the natural gradient $\mathbf{F}_\theta^{-1}\nabla L(\theta^{(t)})$, on networks with millions of parameters, is just intractable.

[28] Martens and Grosse, *"Optimizing neural networks with Kronecker-factored approximate curvature"*, 2015

Fisher is a big Fisher

Computing the inverse of the diagonal Fisher is easy, but computing the inverse of the "full" Fisher $\mathbf{F}^{-1}$ and the natural gradient $\mathbf{F}_\theta^{-1}\nabla L(\theta^{(t)})$, on networks with millions of parameters, is just intractable.

- What about other structural approximations ? We don't want to lose all of the off-diagonal structure;

[28] Martens and Grosse, *"Optimizing neural networks with Kronecker-factored approximate curvature"*, 2015

Fisher is a big Fisher

Computing the inverse of the diagonal Fisher is easy, but computing the inverse of the "full" Fisher $\mathbf{F}^{-1}$ and the natural gradient $\mathbf{F}_\theta^{-1}\nabla L(\theta^{(t)})$, on networks with millions of parameters, is just intractable.

- What about other structural approximations ? We don't want to lose all of the off-diagonal structure;
- However, there are certain goals that we should be ideally try to achieve: memory (remember we have $\mathbf{F} \in \mathbb{R}^{n\times n}$) and computation (we want to have an efficient $\mathbf{F}^{-1}$);

[28] Martens and Grosse, *"Optimizing neural networks with Kronecker-factored approximate curvature"*, 2015

Fisher is a big Fisher

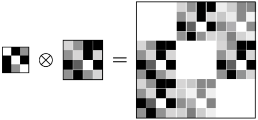
Computing the inverse of the diagonal Fisher is easy, but computing the inverse of the “full” Fisher $\mathbf{F}^{-1}$ and the natural gradient $\mathbf{F}_{\theta}^{-1}\nabla L(\theta^{(t)})$, on networks with millions of parameters, is just intractable.

- What about other structural approximations ? We don’t want to lose all of the off-diagonal structure;
- However, there are certain goals that we should be ideally try to achieve: memory (remember we have $\mathbf{F} \in \mathbb{R}^{n \times n}$) and computation (we want to have an efficient $\mathbf{F}^{-1}$);
- That is what Kronecker-Factored Approximate Curvature (K-FAC) [28] proposes, an structured approximation to natural gradient descent;

[28] Martens and Grosse, *“Optimizing neural networks with Kronecker-factored approximate curvature”*, 2015

Kronecker product

$$\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} [\mathbf{A}]_{1,1}\mathbf{B} & \cdots & [\mathbf{A}]_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ [\mathbf{A}]_{m,1}\mathbf{B} & \cdots & [\mathbf{A}]_{m,n}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{ma \times nb}$$

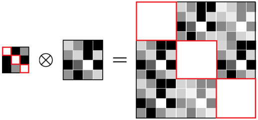


$\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{a \times b}$: Kronecker factors

Source: Kazuki Osawa. Introducing k-fac: A second-order optimization method for large-scale deep learning, 2018.

Kronecker product

$$\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} [\mathbf{A}]_{1,1}\mathbf{B} & \cdots & [\mathbf{A}]_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ [\mathbf{A}]_{m,1}\mathbf{B} & \cdots & [\mathbf{A}]_{m,n}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{ma \times nb}$$

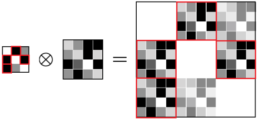


$\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{a \times b}$: Kronecker factors

Source: Kazuki Osawa. Introducing k-fac: A second-order optimization method for large-scale deep learning, 2018.

Kronecker product

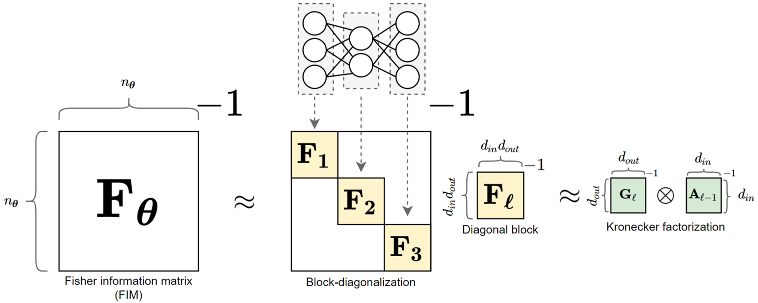
$$\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} [\mathbf{A}]_{1,1}\mathbf{B} & \cdots & [\mathbf{A}]_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ [\mathbf{A}]_{m,1}\mathbf{B} & \cdots & [\mathbf{A}]_{m,n}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{ma \times nb}$$



$\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{a \times b}$: Kronecker factors

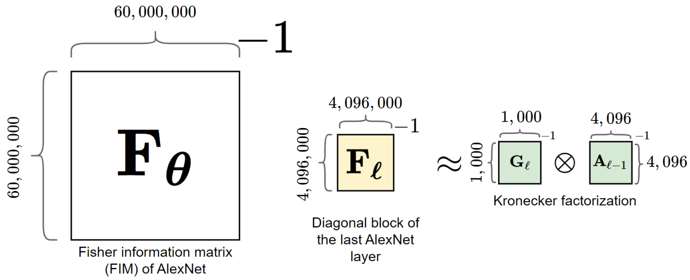
Source: Kazuki Osawa. Introducing k-fac: A second-order optimization method for large-scale deep learning, 2018.

Fisher approximation



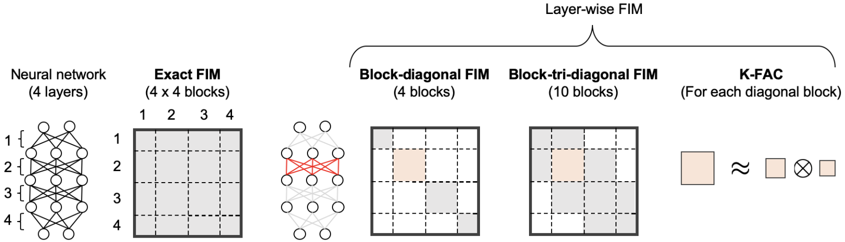
Source: Kazuki Osawa. Introducing k-fac: A second-order optimization method for large-scale deep learning, 2018.

Fisher approximation



Source: Kazuki Osawa. Introducing k-fac: A second-order optimization method for large-scale deep learning, 2018.

Fisher approximation



Source: Osawa, K. et al. Understanding Approximate Fisher Information for Fast Convergence of Natural Gradient Descent in Wide Neural Networks, 2020.

Kronecker Inversion

Kronecker product has a very interesting and critical property:

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$$

This means that the inverse of the product is the same as the product of the inverse of the operands. And this gives us a critical performance speed-up because we just need to invert small factor matrices.

Backpack in PyTorch

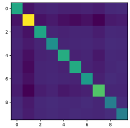
If you want to play with K-FAC on PyTorch, you can try using Backpack [29]:

```
from torch import nn
from backpack import backpack, extend
from backpack.extensions import KFAC
from backpack.utils.examples import load_one_batch_mnist
from backpack.utils import kroneckers

X, y = load_one_batch_mnist(batch_size=512)

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 10)
)

lossfunc = nn.CrossEntropyLoss()

model = extend(model)
lossfunc = extend(lossfunc)

loss = lossfunc(model(X), y)

with backpack(KFAC(mc_samples=1)):
    loss.backward()
```

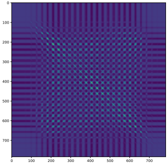
```
named_params = dict(model.named_parameters())
layer_weights = named_params["1.weight"]
# layer_weights.grad = [10, 784]

kfac_f1, kfac_f2 = layer_weights.kfac
# kfac_f1 = [10, 10]
# kfac_f2 = [784, 784]

mat = kroneckers.two_kfacs_to_mat(kfac_f1,
                                  kfac_f2)
# mat = [7840, 7840]
```

[29] Dangel, Kunstner, and Hennig, *"BackPACK: Packing more into backprop"*, 2019

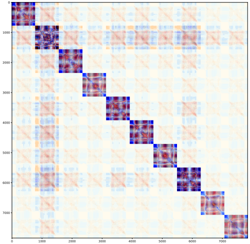
Kronecker Matrices



$\otimes$



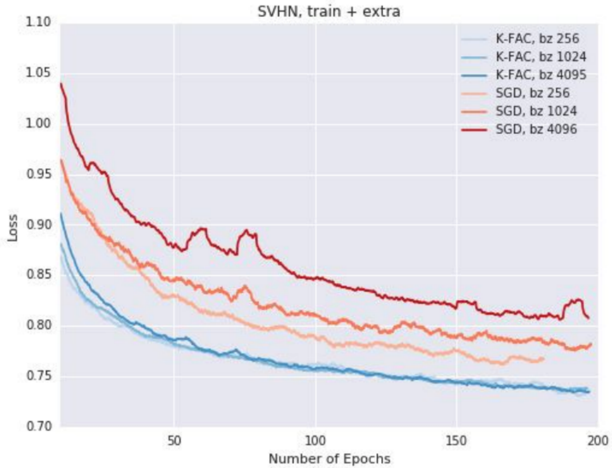
$=$



$\mathbf{A} \in \mathbb{R}^{10\times 10}$ $\quad$ $\mathbf{B} \in \mathbb{R}^{784\times 784}$ $\quad$ $\widetilde{\mathbf{G}}(\theta) \in \mathbb{R}^{7840\times 7840}$

Note that the colormap of the $\widetilde{\mathbf{G}}(\theta)$ was changed for visualization purposes.

Some empirical results

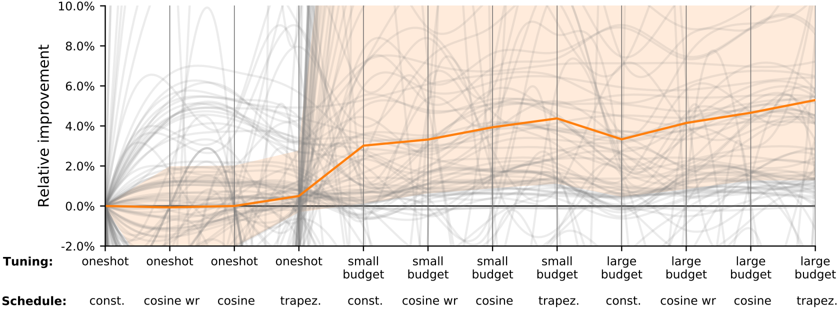


Source: Johnson, M. et al. K-FAC and Natural Gradients, 2017. https://supercomputersfordl2017.github.io/Presentations/K-FAC.pdf .

Section V

Thoughts

Benchmarking optimizers



Source: Schmidt, R. M., Schneider, F., & Hennig, P. (2020). Descending through a Crowded Valley – Benchmarking Deep Learning Optimizers.

Lines in gray (—, smoothed by cubic splines for visual guidance only) show the relative improvement for a certain tuning and schedule (compared to the *one-shot* tuning without schedule) for all 14 optimizers on all eight test problems. The median over all lines is plotted in orange (—) with the shaded area indicating the area between the 25th and 75th percentile. [30]

[30] Schmidt, Schneider, and Hennig, *"Descending through a Crowded Valley – Benchmarking Deep Learning Optimizers"*, 2020

To think

- Do we really need normalization techniques (i.e. Batch Normalization) if we can come up with optimization methods that embed invariant properties ?

To think

- Do we really need normalization techniques (i.e. Batch Normalization) if we can come up with optimization methods that embed invariant properties ?
- What are the other difficult problems we can optimize with better optimization algorithms ?

To think

- Do we really need normalization techniques (i.e. Batch Normalization) if we can come up with optimization methods that embed invariant properties ?
- What are the other difficult problems we can optimize with better optimization algorithms ?
- What other approximations can we achieve ?

To think

- Do we really need normalization techniques (i.e. Batch Normalization) if we can come up with optimization methods that embed invariant properties ?
- What are the other difficult problems we can optimize with better optimization algorithms ?
- What other approximations can we achieve ?
- What is empirical Fisher actually doing ?

To think

- Do we really need normalization techniques (i.e. Batch Normalization) if we can come up with optimization methods that embed invariant properties ?
- What are the other difficult problems we can optimize with better optimization algorithms ?
- What other approximations can we achieve ?
- What is empirical Fisher actually doing ?
- What are the barriers to the use of second-order or approximately second-order methods ? Are we going to see more software support ?

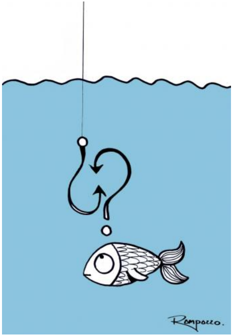
To think

- Do we really need normalization techniques (i.e. Batch Normalization) if we can come up with optimization methods that embed invariant properties ?
- What are the other difficult problems we can optimize with better optimization algorithms ?
- What other approximations can we achieve ?
- What is empirical Fisher actually doing ?
- What are the barriers to the use of second-order or approximately second-order methods ? Are we going to see more software support ?
- Are we driving towards more hyper-parameters or more robust methods ?

To think

- Do we really need normalization techniques (i.e. Batch Normalization) if we can come up with optimization methods that embed invariant properties ?
- What are the other difficult problems we can optimize with better optimization algorithms ?
- What other approximations can we achieve ?
- What is empirical Fisher actually doing ?
- What are the barriers to the use of second-order or approximately second-order methods ? Are we going to see more software support ?
- Are we driving towards more hyper-parameters or more robust methods ?
- What are properties of the different solutions that different optimization methods can achieve ?

Q&A



References I

Amari, Shun Ichi (Feb. 1998). "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2, pp. 251–276. ISSN: 08997667. DOI: 10.1162/089976698300017746. URL: https://www.mitpressjournals.org/doi/abs/10.1162/089976698300017746.

— (2016). "Information geometry and its applications". In: *Applied Mathematical Sciences (Switzerland)*. Vol. 194. Springer, pp. i–374. DOI: 10.1007/978-4-431-55978-8.

Bengio, Yoshua (2012). "Practical recommendations for gradient-based training of deep architectures". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7700 LECTU, pp. 437–478. ISSN: 03029743. DOI: 10.1007/978-3-642-35289-8-26. arXiv: 1206.5533. URL: https://arxiv.org/pdf/1206.5533.pdf.

Bottou, Léon, Frank E Curtis, and Jorge Nocedal (2018). *Optimization methods for large-scale machine learning*. DOI: 10.1137/16M1080173. arXiv: 1606.04838.

Choi, Dami et al. (Oct. 2019). *On empirical comparisons of optimizers for deep learning*. arXiv: 1910.05446. URL: http://arxiv.org/abs/1910.05446.

Dangel, Felix, Frederik Kunstner, and Philipp Hennig (2019). "BackPACK: Packing more into backprop". In: *ICLR*, pp. 1–22. arXiv: 1912.10985. URL: http://arxiv.org/abs/1912.10985.

References II

Dauphin, Yann N et al. (2014). “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in Neural Information Processing Systems*. Vol. 4. January, pp. 2933–2941. arXiv: 1406.2572.

Johnson, Matt et al. (2017). “K-FAC and Natural Gradients”. In: URL: https://supercomputersfordl2017.github.io/Presentations/K-FAC.pdf.

Karakida, Ryo and Kazuki Osawa (Oct. 2020). “Understanding Approximate Fisher Information for Fast Convergence of Natural Gradient Descent in Wide Neural Networks”. In: arXiv: 2010.00879. URL: http://arxiv.org/abs/2010.00879.

Kingma, Diederik P. and Jimmy Lei Ba (2015). “Adam: a Method for Stochastic Optimization”. In: *International Conference on Learning Representations 2015*, pp. 1–15. ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. arXiv: 1412.6980v9.

Kunstner, Frederik, Lukas Balles, and Philipp Hennig (2019). “Limitations of the empirical fisher approximation for natural gradient descent”. In: *Advances in Neural Information Processing Systems*. Vol. 32. arXiv: 1905.12558.

Lê, Hông Vân (June 2017). “The uniqueness of the Fisher metric as information metric”. In: *Annals of the Institute of Statistical Mathematics* 69.4, pp. 879–896. ISSN: 15729052. DOI: 10.1007/s10463-016-0562-0. arXiv: 1306.1465. URL: http://arxiv.org/abs/1306.1465.

Martens, James (2010). *Deep learning via Hessian-free optimization*. Tech. rep.

References III

- Martens, James (2014). "New insights and perspectives on the natural gradient method". In: *Journal of Machine Learning Research* 21, pp. 1–76. ISSN: 15337928. arXiv: 1412.1193. URL: http://arxiv.org/abs/1412.1193.
- Martens, James and Roger Grosse (2015). "Optimizing neural networks with Kronecker-factored approximate curvature". In: *32nd International Conference on Machine Learning, ICML 2015*. Vol. 3, pp. 2398–2407. ISBN: 9781510810587. arXiv: 1503.05671.
- Pascanu, Razvan and Yoshua Bengio (Jan. 2013). "Revisiting Natural Gradient for Deep Networks". In: *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*. arXiv: 1301.3584. URL: http://arxiv.org/abs/1301.3584.
- Ratliff, Nathan (2013). "Information Geometry and Natural Gradients". In: pp. 1–8. URL: http://www.nathanratliff.com/pedagogy/mathematics-for-intelligent-systems/mathematics%7B%5C_%7Dfor%7B%5C_%7Dintelligent%7B%5C_%7Dsystems%7B%5C_%7Dlecture12%7B%5C_%7Dnotes%7B%5C_%7DI.pdf?attredirects=0.
- Robbins, Herbert and Sutton Monro (Sept. 1951). "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729586. URL: https://projecteuclid.org/euclid.aoms/1177729586.
- Sagun, Levent, Leon Bottou, and Yann LeCun (2016). "Eigenvalues of the Hessian in Deep Learning: Singularity and Beyond". In: arXiv: 1611.07476. URL: http://arxiv.org/abs/1611.07476.

References IV

- Schmidt, Robin M, Frank Schneider, and Philipp Hennig (2020). "Descending through a Crowded Valley – Benchmarking Deep Learning Optimizers". In: arXiv: 2007.01547. URL: https://github.com/SirRob1997/Crowded-Valley---Results%20http://arxiv.org/abs/2007.01547.
- Staib, Matthew et al. (2019). "Escaping saddle points with adaptive gradient methods". In: *36th International Conference on Machine Learning, ICML 2019*. Vol. 2019-June, pp. 10420–10454. ISBN: 9781510886988. arXiv: 1901.09149.
- Thomas, Valentin et al. (2019). "On the interplay between noise and curvature and its effect on optimization and generalization". In: arXiv: 1906.07774. URL: http://arxiv.org/abs/1906.07774.
- Truong, Tuyen Trung et al. (June 2020). "A modification of quasi-Newton's methods helping to avoid saddle points". In: arXiv: 2006.01512. URL: http://arxiv.org/abs/2006.01512.
- Watt, Jeremy, Reza Borhani, and Aggelos Katsaggelos (Jan. 2020). *Machine Learning Refined*. Cambridge University Press. ISBN: 9781108690935. DOI: 10.1017/9781108690935. URL: https://www.cambridge.org/core/product/identifier/9781108690935/type/book.
- Yao, Zhewei et al. (Dec. 2019). *PYHESSIAN: Neural networks through the lens of the hessian*. arXiv: 1912.07145. URL: http://arxiv.org/abs/1912.07145.