Machine Learning and Intelligent Systems

Gradient Descent

Maria A. Zuluaga

October 27, 2023

EURECOM - Data Science Department

Table of contents

Quick Logistic Regression Review

Logistic Regression: Quick recap

- The loss function to minimize for logistic regression is the cross-entropy loss function:

$$E(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log \left(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right)$$

Logistic Regression: Quick recap

- The loss function to minimize for logistic regression is the cross-entropy loss function:

$$E(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log \left(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right)$$

- Due to the non-linearity of the sigmoid function, there is no closed-form solution that allows to obtain $\mathbf{w}$.

Logistic Regression: Quick recap

- The loss function to minimize for logistic regression is the cross-entropy loss function:

$$E(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log \left(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right)$$

- Due to the non-linearity of the sigmoid function, there is no closed-form solution that allows to obtain $\mathbf{w}$.
- Recall:

$$\frac{dE}{d\mathbf{w}} = \sum_{i=1}^{N} \left(\sigma(\mathbf{w}^T \mathbf{x}_i) - y_i\right) \mathbf{x}_i$$

Logistic Regression: Quick recap

- The loss function to minimize for logistic regression is the cross-entropy loss function:

$$E(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log \left(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right)$$

- Due to the non-linearity of the sigmoid function, there is no closed-form solution that allows to obtain $\mathbf{w}$.
- Recall:

$$\frac{dE}{d\mathbf{w}} = \sum_{i=1}^{N} \left(\sigma(\mathbf{w}^T \mathbf{x}_i) - y_i\right) \mathbf{x}_i$$

- We will now see a way to find $\mathbf{w}$ through an iterative algorithm.

Gradient Descent

The Optimization Problem: Formalization

- Given $\boldsymbol{\theta} \in \mathbb{R}^D$ and an **objective, cost energy** function $J : \mathbb{R}^D \longrightarrow \mathbb{R}$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

The Optimization Problem: Formalization

- Given $\boldsymbol{\theta} \in \mathbb{R}^D$ and an **objective, cost energy** function $J : \mathbb{R}^D \longrightarrow \mathbb{R}$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- **In words:** Choose $\boldsymbol{\theta}$ to minimize J

The Optimization Problem: Formalization

- Given $\boldsymbol{\theta} \in \mathbb{R}^D$ and an **objective, cost energy** function $J : \mathbb{R}^D \longrightarrow \mathbb{R}$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- **In words:** Choose $\boldsymbol{\theta}$ to minimize J
- We denote $\boldsymbol{\theta}$ **optimal** if $\forall\, \boldsymbol{\theta}$

$$J(\boldsymbol{\theta}^*) \leq J(\boldsymbol{\theta})$$

The Optimization Problem: Formalization

- Given $\boldsymbol{\theta} \in \mathbb{R}^D$ and an **objective, cost energy** function $J : \mathbb{R}^D \longrightarrow \mathbb{R}$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- **In words:** Choose $\boldsymbol{\theta}$ to minimize J
- We denote $\boldsymbol{\theta}$ **optimal** if $\forall\, \boldsymbol{\theta}$

$$J(\boldsymbol{\theta}^*) \leq J(\boldsymbol{\theta})$$

- $J^* = J(\boldsymbol{\theta}^*)$ is the **optimal value**

The Optimization Problem: An equivalence

- Given $\boldsymbol{\theta} \in \mathbb{R}^D$ and an **utility** function $J : \mathbb{R}^D \longrightarrow \mathbb{R}$:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \; -J(\boldsymbol{\theta})$$

- **In words:** Choose $\boldsymbol{\theta}$ to **maximize** $-J$

- We denote $\boldsymbol{\theta}$ **optimal** if $\forall\, \boldsymbol{\theta}$

$$J(\boldsymbol{\theta}^*) \geq J(\boldsymbol{\theta})$$

The Optimization Problem: An equivalence

- Given $\boldsymbol{\theta} \in \mathbb{R}^D$ and an **utility** function $J : \mathbb{R}^D \longrightarrow \mathbb{R}$:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \; -J(\boldsymbol{\theta})$$

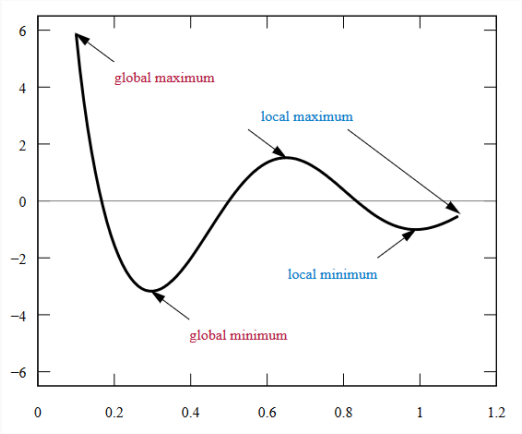
- **In words:** Choose $\boldsymbol{\theta}$ to **maximize** $-J$

- We denote $\boldsymbol{\theta}$ **optimal** if $\forall\, \boldsymbol{\theta}$

$$J(\boldsymbol{\theta}^*) \geq J(\boldsymbol{\theta})$$

- In the remaining we will use minimization but, remember these are equivalent formulations

Local vs Global Minima



Source: Wikipedia

Global minimum:

$\boldsymbol{\theta}^*$ is a global minimum if

$$J(\boldsymbol{\theta}^*) \leq J(\boldsymbol{\theta}) \; \forall \, \boldsymbol{\theta} \in \Theta$$

Local minimum:

$\boldsymbol{\theta}^*$ is a local minimum if there exists some $\varepsilon > 0$ such that:

$$J(\boldsymbol{\theta}^*) \leq J(\boldsymbol{\theta}) \; \forall \, \boldsymbol{\theta} \in \Theta$$

within distance ϵ of $\boldsymbol{\theta}^*$.

Θ : Functions' domain

Convexity

- A **convex** function has only one minimum.

Convexity

- A **convex** function has only one minimum.
- If J is convex, then it is possible to guarantee finding its global optima

Convexity

- A **convex** function has only one minimum.
- If J is convex, then it is possible to guarantee finding its global optima

- **Convexity:** A function is convex if for every pair of points $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^D$ the line connecting $(\boldsymbol{\theta}_1, J(\boldsymbol{\theta}_1))$ to $(\boldsymbol{\theta}_2, J(\boldsymbol{\theta}_2))$ does not go below $J(\cdot)$.

Convexity

- A **convex** function has only one minimum.
- If J is convex, then it is possible to guarantee finding its global optima

- **Convexity:** A function is convex if for every pair of points $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^D$ the line connecting $(\boldsymbol{\theta}_1, J(\boldsymbol{\theta}_1))$ to $(\boldsymbol{\theta}_2, J(\boldsymbol{\theta}_2))$ does not go below $J(\cdot)$.
- For $D = 1$ convexity is guaranteed if $J''(\theta) \geq 0 \, \forall \, \theta$.

Solving an Optimization Problem

- So far, we have faced some optimization problems, which we have solved analytically.

Solving an Optimization Problem

- So far, we have faced some optimization problems, which we have solved analytically.
- Example: Closed-form solution for linear regression

$$\hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

- As the sum of squares loss is a convex function, $\hat{\mathbf{w}}$ is a global optimal solution.

Iterative Algorithms

- In other cases, it is not possible to obtain a closed-form solution
- Example: Logistic regression

Iterative Algorithms

- In other cases, it is not possible to obtain a closed-form solution
- Example: Logistic regression
- There is the need to use **iterative algorithms**

Iterative Algorithms

- In other cases, it is not possible to obtain a closed-form solution
- Example: Logistic regression
- There is the need to use **iterative algorithms**

Definition:

- An iterative algorithm computes a sequence of values $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(\tau)}$, where $\boldsymbol{\theta}^{(\tau)}$ denotes the value of $\boldsymbol{\theta}$ at the τ-th iteration
- For iterative descent methods we expect that $J(\boldsymbol{\theta}^{(\tau+1)}) < J(\boldsymbol{\theta}^{(\tau)}), \tau = 0, 1, \ldots$
- It is expected that $J(\boldsymbol{\theta}^{(\tau)}) \longrightarrow J^*$ as $\tau \longrightarrow \infty$

Iterative Algorithms: Elements

- Iterative algorithms require two elements:

Iterative Algorithms: Elements

- Iterative algorithms require two elements:
  1. An update rule that defines how to go from $\boldsymbol{\theta}^{(\tau)}$ to $\boldsymbol{\theta}^{(\tau+1)}$

Iterative Algorithms: Elements

- Iterative algorithms require two elements:
  1. An update rule that defines how to go from $\boldsymbol{\theta}^{(\tau)}$ to $\boldsymbol{\theta}^{(\tau+1)}$
  2. A stopping criterion that establishes when to stop iterating

Iterative Algorithms: Elements

- Iterative algorithms require two elements:
  1. An update rule that defines how to go from $\boldsymbol{\theta}^{(\tau)}$ to $\boldsymbol{\theta}^{(\tau+1)}$
  2. A stopping criterion that establishes when to stop iterating

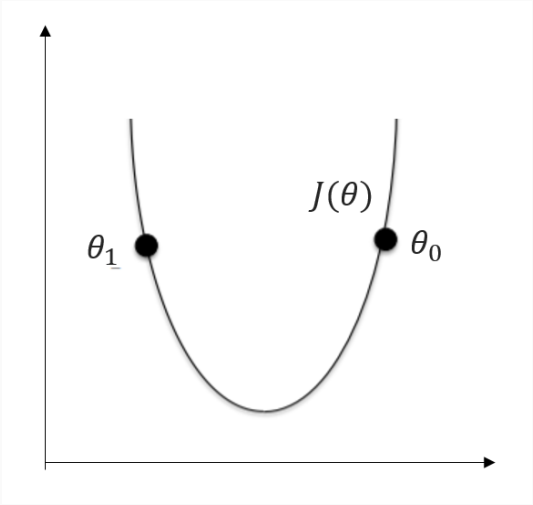
- In an ideal setting, the stopping criterion should be to stop when $\boldsymbol{\theta}^{(\tau)}$ converges to J^*
- This, however, cannot be guaranteed

Iterative Algorithms: Elements

- Iterative algorithms require two elements:
  1. An update rule that defines how to go from $\boldsymbol{\theta}^{(\tau)}$ to $\boldsymbol{\theta}^{(\tau+1)}$
  2. A stopping criterion that establishes when to stop iterating

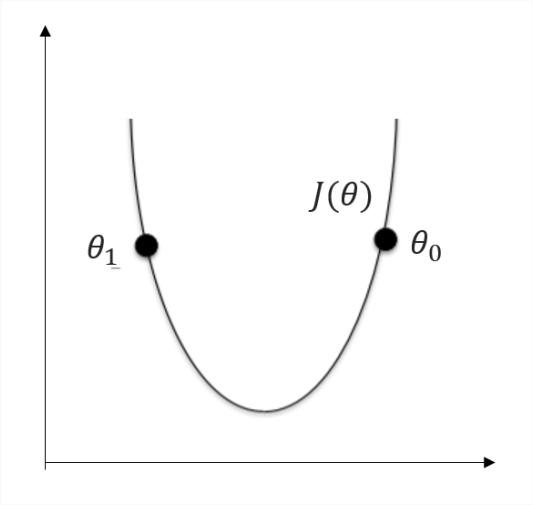
- In an ideal setting, the stopping criterion should be to stop when $\boldsymbol{\theta}^{(\tau)}$ converges to J^*
- This, however, cannot be guaranteed
- The goal is to have $\boldsymbol{\theta}^{(\tau)}$ not too far from J^*

Gradient Descent: Intuition



- Let $J(\theta) : \mathbb{R} \longrightarrow \mathbb{R}$ be the function in the plot
- **Goal:** To find a value θ_{min} for which $J(\theta)$ is minimal.

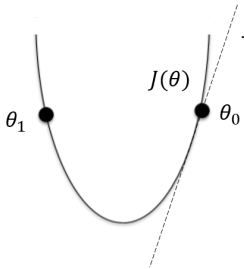
Gradient Descent: Intuition

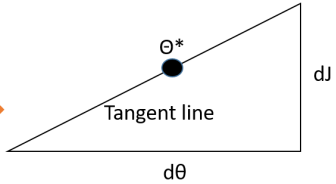


- Let $J(\theta) : \mathbb{R} \longrightarrow \mathbb{R}$ be the function in the plot
- **Goal:** To find a value θ_{min} for which $J(\theta)$ is minimal.

How we can achieve this through an iterative algorithm?

The Gradient





$\frac{dJ}{d\theta}$: slope

How much change in f if there is a change in θ

- The gradient is a multi-variable generalization of the slope
- For a function as in the current example, we can informally think of the gradient as the slope.
- The dashed line shows the slope of J at point $\theta = \theta_0$

A Sketch of the Algorithm

Algorithm 1 First sketch of gradient descent

$\theta min \leftarrow \text{RANDOM}(\theta)$
while $J'(\theta min) \neq 0$ **do**
    if $J'(\theta min) < 0$ **then**
        Move θmin in same direction
    end if
    if $J'(\theta min) > 0$ **then**
        Move θmin in opposite direction
    end if
end while

1. The Update Rule

- At $\tau = 0$ the value of $\boldsymbol{\theta}$ is initialized to some random starting vector $\boldsymbol{\theta}^{(0)}$ (see Algorithm 1)

1. The Update Rule

- At $\tau = 0$ the value of $\boldsymbol{\theta}$ is initialized to some random starting vector $\boldsymbol{\theta}^{(0)}$ (see Algorithm 1)

- At a given iteration τ, the gradient descent algorithm will update the parameter vector $\boldsymbol{\theta}$ using:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{1}$$

  with α a learning rate parameter.

1. The Update Rule

- At $\tau = 0$ the value of $\boldsymbol{\theta}$ is initialized to some random starting vector $\boldsymbol{\theta}^{(0)}$ (see Algorithm 1)

- At a given iteration τ, the gradient descent algorithm will update the parameter vector $\boldsymbol{\theta}$ using:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{1}$$

  with α a learning rate parameter.

- For D-dimensional functions, the update rule is applied in parallel to each dimension:

$$\theta_d := \theta_d - \alpha \cdot \frac{\partial J}{\partial \theta_d}(\boldsymbol{\theta})$$

2. The Stopping Criteria

- **Rule 1:** Reach maximum number of iterations τ_{max}

2. The Stopping Criteria

- **Rule 1:** Reach maximum number of iterations τ_{max}

- **Rule 2:** $\left\|\nabla J(\boldsymbol{\theta}^{(\tau)})\right\| \leq \epsilon$
- $\epsilon > 0$ here denoted the stopping tolerance

2. The Stopping Criteria

- **Rule 1:** Reach maximum number of iterations τ_{max}

- **Rule 2:** $\left\|\nabla J(\boldsymbol{\theta}^{(\tau)})\right\| \leq \epsilon$
- $\epsilon > 0$ here denoted the stopping tolerance

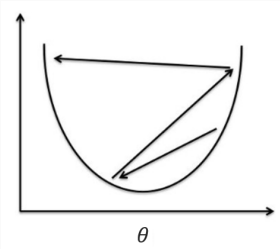
- Recall the goal is to have $J(\boldsymbol{\theta}^{(\tau)})$ not too far from J^*
- **Important:** $J(\boldsymbol{\theta}^{(\tau)})$ converges but, not necessarily to J^*

The Learning Rate α

- The learning rate α controls the size of the steps performed by the algorithm at every iteration

The Learning Rate α

- The learning rate α controls the size of the steps performed by the algorithm at every iteration
- If α is too large it can occur that $J(\boldsymbol{\theta}^{(\tau+1)}) > J(\boldsymbol{\theta}^{(\tau)})$



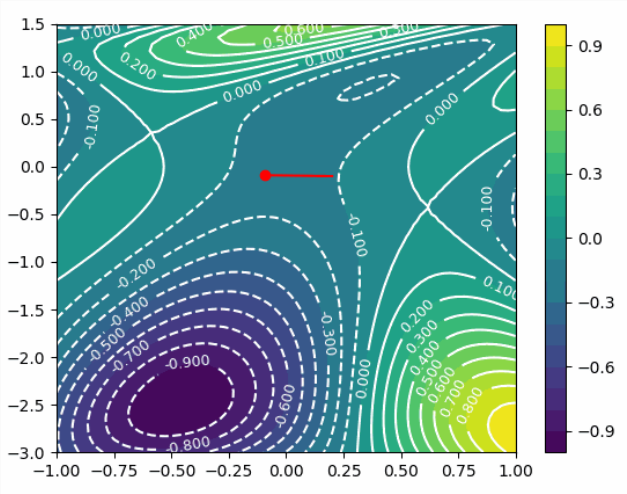
- **Simple solution:** varying the learning rate every τ
- Example:
  - If $J(\boldsymbol{\theta}^{(\tau+1)}) > J(\boldsymbol{\theta}^{(\tau)})$, set $\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)}$ and $\alpha^{(\tau+1)} = 0.5\alpha^{(\tau)}$
  - else $\alpha^{(\tau+1)} = 1.3\alpha^{(\tau)}$

The Gradient Descent Algorithm

Algorithm 2 Gradient Descent Algorithm

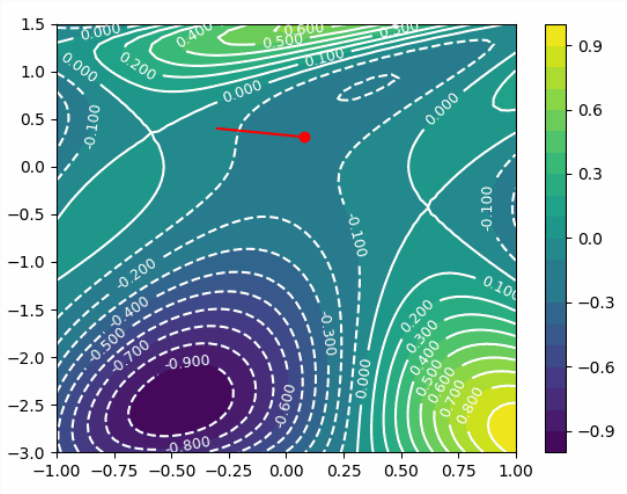
procedure GradientDescent(α, τ_{max})
  $\boldsymbol{\theta}^{(0)} \leftarrow$ random($\boldsymbol{\theta} \in \mathbb{R}^D$), $\alpha^{(0)} \leftarrow \alpha$, $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}^{(0)}$
  if $\|J'(\boldsymbol{\theta}^*)\| < \epsilon$ **then**
    return $\boldsymbol{\theta}^*$
  end if
  for $\tau = 0, 1, \ldots \tau_{max}$ **do**
    $\boldsymbol{\theta}_{tmp} \leftarrow \theta^{(\tau)} + \alpha^{(\tau)} \nabla J(\theta)$
    if $J(\boldsymbol{\theta}_{tmp}) \leq J(\boldsymbol{\theta}^{(\tau)})$ **then**
      $\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}_{tmp}$, $\alpha^{(\tau+1)} \leftarrow 1.3\alpha^{(\tau)}$
      $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}_{tmp}$
    else
      $\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)}$, $\alpha^{(\tau+1)} \leftarrow 0.5\alpha^{(\tau)}$
    end if
  end for
  return $\boldsymbol{\theta}^*$
end procedure

Example 1: Good Initialization, $\alpha = 1$, $\tau_{max} = 12$



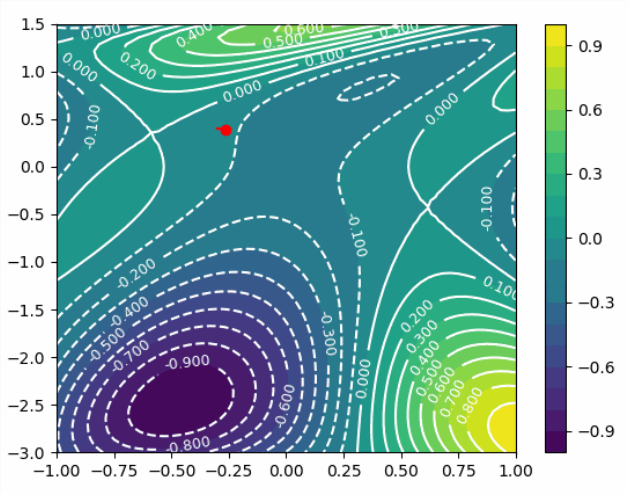
Credits: Simone Rossi

Example 2: Bad Initialization, $\alpha = 1$, $\tau_{max} = 10$



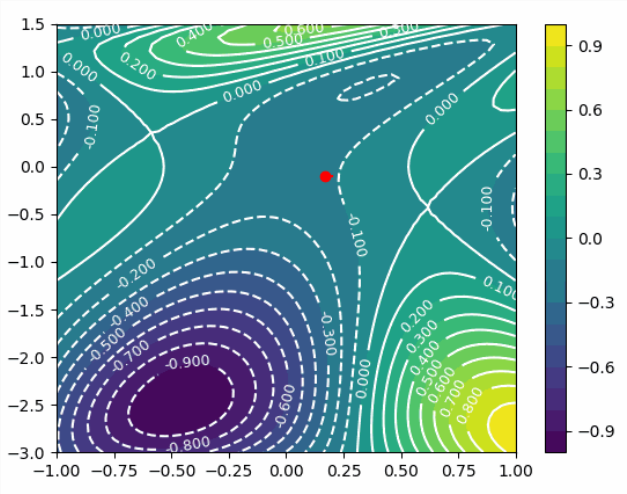
Credits: Simone Rossi

Example 3: Bad Initialization, $\alpha = 0.1$, $\tau_{max} = 100$



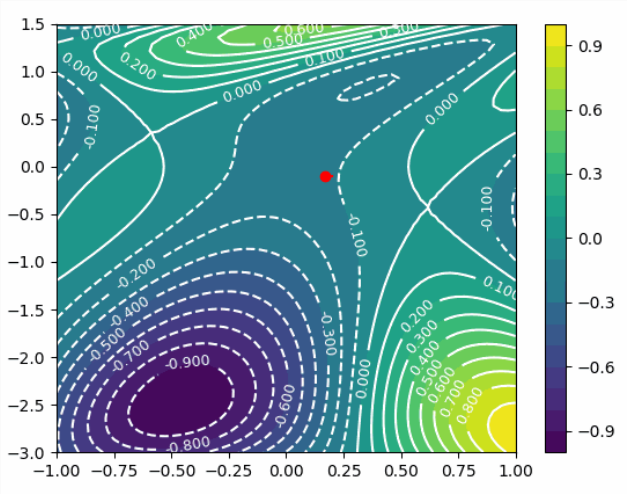
Credits: Simone Rossi

Example 4: Good Initialization, $\alpha = 0.1$, $\tau_{max} = 100$



Credits: Simone Rossi

Example 5: Good Initialization, $\alpha = 0.01$, $\tau_{max} = 100$



Credits: Simone Rossi

In Summary

Gradient descent is a **first-order iterative** optimization algorithm for finding a **local minimum** of a **differentiable function** J.

In Summary

Gradient descent is a **first-order iterative** optimization algorithm for finding a **local minimum** of a **differentiable function** J.

Keywords

- **first-order:** The algorithm relies on the computation of the first derivative (gradient) of J

In Summary

Gradient descent is a **first-order iterative** optimization algorithm for finding a **local minimum** of a **differentiable function** J.

Keywords

- **first-order:** The algorithm relies on the computation of the first derivative (gradient) of J
- **differentiable** For J there exists J' at every point in its domain

In Summary

Gradient descent is a **first-order iterative** optimization algorithm for finding a **local minimum** of a **differentiable function** J.

Keywords

- **first-order:** The algorithm relies on the computation of the first derivative (gradient) of J
- **differentiable** For J there exists J' at every point in its domain
- **iterative:** Computes a sequence of values $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(\tau)}$

In Summary

Gradient descent is a **first-order iterative** optimization algorithm for finding a **local minimum** of a **differentiable function** J.

Keywords

- **first-order:** The algorithm relies on the computation of the first derivative (gradient) of J
- **differentiable** For J there exists J' at every point in its domain
- **iterative:** Computes a sequence of values $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(\tau)}$
- **local minimum:** Unless the function is convex, it risks falling into local minima. It cannot guarantee convergence to the global optimum.

In Summary: Pros & Cons

Pros

- Scales well to large datasets
- No need to multiply matrices
- Solves many convex problems
- Unreasonable good heuristic for the approximate solution of non-convex problems

Cons

- Depends on good initialization
- Need to re-run several times to find a sufficiently good minimum
- Cannot cope with non-differentiable functions
- Slow over functions of elliptic form

Gradient Descent Variants

Batch Gradient Descent (Vanilla) - the gradient is estimated from the entire dataset:

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} J_i(\theta)$$

Gradient Descent Variants

Batch Gradient Descent (Vanilla) - the gradient is estimated from the entire dataset:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$$

Stochastic Gradient Descent (SGB) - the gradient is approximated by a gradient at a single example i randomly chosen at each iteration:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \alpha \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$$

Gradient Descent Variants

Batch Gradient Descent (Vanilla) - the gradient is estimated from the entire dataset:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$$

Stochastic Gradient Descent (SGB) - the gradient is approximated by a gradient at a single example i randomly chosen at each iteration:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \alpha \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$$

Mini-Batch - the gradient is approximated by a gradient estimated from a batch of size $M < N$, randomly chosen at each iteration:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \alpha \frac{1}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$$

Exercise: Linear Regression & Gradient Descent

Write down the update rule (Eq. 1) for the case of the sum of squares loss function.
Hints:

- $\theta \longrightarrow \mathbf{w}$
- $J \longrightarrow \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} l_{\mathbf{w}}(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
- $\hat{y}_i = \mathbf{w}^T \mathbf{x}_i$ (contracted version)

Your solution:

Exercise: Linear Regression & Gradient Descent

Write down the update rule (Eq. 1) for the case of the sum of squares loss function.
Hints:

- $\theta \longrightarrow \mathbf{w}$
- $J \longrightarrow \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} l_{\mathbf{w}}(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
- $\hat{y}_i = \mathbf{w}^T \mathbf{x}_i$ (contracted version)

Your solution:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} + 2\alpha \left(y_i - \mathbf{w}^T \mathbf{x}_i\right) \mathbf{x}_i$$

Note: Stochastic gradient descent rule

Solving the Logistic Regression Problem

Logistic Regression & Gradient Descent

As in the previous exercise, we need to identify θ and J, and to estimate a general expression for ∇J

- $\theta \longrightarrow \mathbf{w}$

Logistic Regression & Gradient Descent

As in the previous exercise, we need to identify θ and J, and to estimate a general expression for ∇J

- $\theta \longrightarrow \mathbf{w}$
- J: From slide 2 we have:

$$J(\mathbf{w}) = E(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log \left(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right)$$

Logistic Regression & Gradient Descent

As in the previous exercise, we need to identify θ and J, and to estimate a general expression for ∇J

- $\theta \longrightarrow \mathbf{w}$
- J: From slide 2 we have:

$$J(\mathbf{w}) = E(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log \left(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right)$$

- and we also have an expression for ∇J:

$$\nabla J = \frac{dE}{d\mathbf{w}} = \sum_{i=1}^{N} \left(\sigma(\mathbf{w}^T \mathbf{x}_i) - y_i\right) \mathbf{x}_i$$

Logistic Regression & Gradient Descent

If we plug-in all the elements together we get:

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \alpha \sum_{i=1}^{N} \left(\sigma(\mathbf{w}^T \mathbf{x}_i) - y_i\right) \mathbf{x}_i \\ &= \mathbf{w}^{(\tau)} + \alpha \sum_{i=1}^{N} \left(y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)\right) \mathbf{x}_i\end{aligned}$$

Observation 1: The contribution to the gradient from data point i is given by the *error* between the target value y_i and the prediction of the model $\sigma(\mathbf{w}^T \mathbf{x}_i)$, weighted by the input sample $\mathbf{x}_i$.

Observation 2: The loss function in logistic regression is a convex function.

Optimization

A Final Word on Optimization

- The fields of machine learning and optimization are increasingly intertwined.

A Final Word on Optimization

- The fields of machine learning and optimization are increasingly intertwined.
- Being a field on its own, there is an endless number of optimization algorithms and methods that cannot be covered in this course.

A Final Word on Optimization

- The fields of machine learning and optimization are increasingly intertwined.
- Being a field on its own, there is an endless number of optimization algorithms and methods that cannot be covered in this course.
- Example: The "defacto" algorithm for logistic regression is the Iterative reweighted least squares (IRLS) which is derived from the Newton-Raphson method

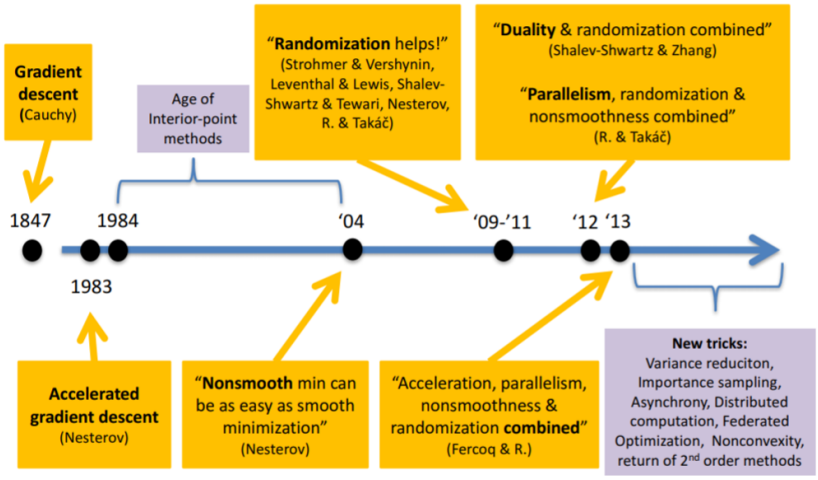
A Final Word on Optimization

- The fields of machine learning and optimization are increasingly intertwined.
- Being a field on its own, there is an endless number of optimization algorithms and methods that cannot be covered in this course.
- Example: The "defacto" algorithm for logistic regression is the Iterative reweighted least squares (IRLS) which is derived from the Newton-Raphson method
- Some other topics will be covered (e.g constrained optimization, Lagrange multipliers) but, the topic is too broad for a single course.
- For those interested in optimization: [OPTIM] Optimization Theory with Applications

A Final Word on Optimization

- The fields of machine learning and optimization are increasingly intertwined.
- Being a field on its own, there is an endless number of optimization algorithms and methods that cannot be covered in this course.
- Example: The "defacto" algorithm for logistic regression is the Iterative reweighted least squares (IRLS) which is derived from the Newton-Raphson method
- Some other topics will be covered (e.g constrained optimization, Lagrange multipliers) but, the topic is too broad for a single course.
- For those interested in optimization: [OPTIM] Optimization Theory with Applications
- **Assignment:** Investigate the Newton-Raphson method and how it can be used to estimate the logistic regression parameters

Brief, Biased and Severely Incomplete History of Big Data Optimization



Credits: Peter Richtarik Data Science Summer School DS3 cole Polytechnique 2017

Recap

Recap

- We introduced iterative algorithms for optimization
- We introduced Gradient Descent, an iterative optimization algorithm that allowed us to estimate the parameters in logistic regression.
- We studied the behavior of gradient descent in different configuration settings
- We formalized the optimization problem

In the coming lectures...

- Gradient Descent will prove to be a very powerful and useful tool

Key Concepts

- Log Odds
- Logistic Regression
- Cross-entropy
- Gradient Descent
- Iterative Algorithm
- Convex function

References

Further Reading and Useful Material

Source	Notes
Pattern Recognition and Machine Learning	Logistic regr., Iterative methods: Secs 3.1.3, 4.3
The Interplay of Optimization and Machine Learning Research	Research article (link)
Convex Optimization - S. Boyd	Very good book (link)
Process optimization - Northwestern Univ	Online book (link)
DS3 2017 Slides - Peter Richtarik	State of the art tutorial (link)
Scipy tutorial - Optimization	Code (link)