CSI5180. Machine Learning for Bioinformatics Applications

Fundamentals of Machine Learning — Gradient Descent

by

Marcel Turcotte

Version **November 6, 2019**

Preamble

Preamble

Fundamentals of Machine Learning — Gradient Descent

In this lecture, we focus on an essential building block for most learning algorithms, the **optimization** algorithm.

General objective :

- **Describe** the fundamental concepts of machine learning

Learning objectives

- In your own words, **explain** the role of the optimization algorithm for solving a linear regression problem.
- **Describe** the function of the (partial) derivative in the gradient descent algorithm.
- **Clarify** the role the learning rate, a hyper-parameter.
- **Compare** the **batch**, **stochastic** and **mini-batch** gradient descent algorithms.

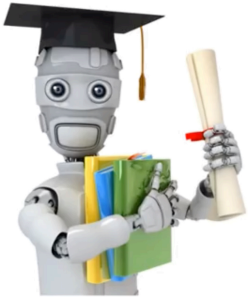
Reading:

- Largly based on Géron 2019, §4.

Plan

1. Preamble
2. Mathematics
3. Problem
4. Building blocks
5. Prologue

Gradient Descent - Andrew Ng (1/4)

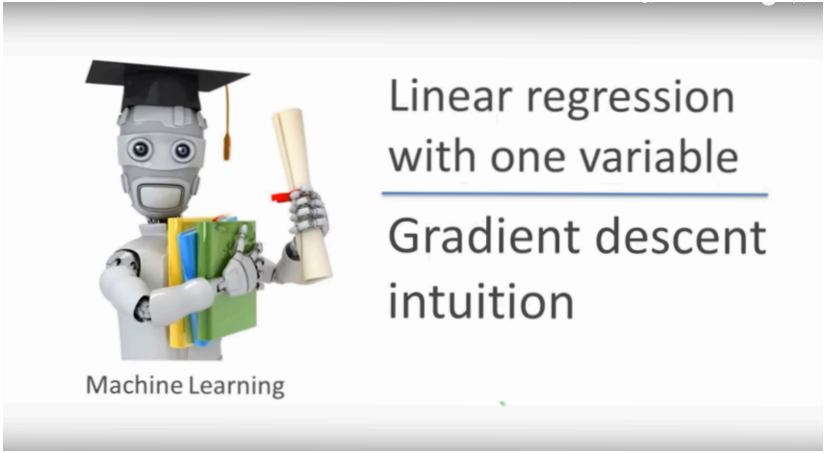


Linear regression with one variable

Gradient descent

https://youtu.be/F6GSRDoB-Cg

Gradient Descent - Andrew Ng (2/4)



https://youtu.be/YovTqTY-PYY

Gradient Descent - Andrew Ng (3/4)

Machine Learning

Linear regression with one variable

Gradient descent for linear regression

https://youtu.be/66rql7He62g

Normal Equation - Andrew Ng (4/4)

m **training examples,** n **features.**

Gradient Descent	Normal Equation
• Need to choose α.	• No need to choose α.
• Needs many iterations.	• Don't need to iterate.
• Works well even when n is large.	• Need to compute $(X^T X)^{-1}$ $n \times n$ $O(n^3)$
	• Slow if n is very large.

Andrew Ng

https://youtu.be/B-Ks01zR4HY

Mathematics

3Blue1Brown

- Essence of **linear algebra**
  - A series of 15 videos (10 to 15 minutes per video) providing "[a] geometric understanding of matrices, determinants, eigen-stuffs and more."
    - 6,662,732 views as of September 30, 2019.
- Essence of **calculus**
  - A series of 12 videos (15 to 20 minutes per video): "The goal here is to make calculus feel like something that you yourself could have discovered."
    - 2,309,726 views as of September 30, 2019.

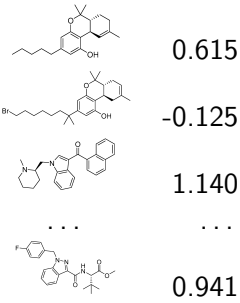
Problem

Supervised learning - regression

- The **data set** is a collection of **labelled** examples.
  - $\{(x_i, y_i)\}_{i=1}^{N}$
    - Each x_i is a **feature vector** with D dimensions.
    - $x_i^{(j)}$ is the value of the **feature** j of the example i, for $j \in 1 \dots D$ and $i \in 1 \dots N$.
  - The **label** y_i is a **real number**.
- **Problem**: given the data set as input, create a "**model**" that can be used to predict the value of y for an unseen x.

QSAR

- **QSAR** stands for **Quantitative Structure-Activity Relationship**
- As a machine learning problem,
  - Each x_i is a **chemical compound**
  - y_i is the **biological activity** of the compound x_i
    - Examples of biological activity include **toxicology** and **biodegradability**



HIV-1 reverse transcriptase inhibitors

- Viira, B., García-Sosa, A. T. & Maran, U. **Chemical structure and correlation analysis of HIV-1 NNRT and NRT inhibitors and database-curated, published inhibition constants with chemical structure in diverse datasets.** *J Mol Graph Model* **76**:205223 (2017).
- Each compound (**example**) in **ChemDB** has features such as the **number of atoms**, **area**, **solvation**, **coulombic**, **molecular weight**, **XLogP**, etc.
- A possible solution, a model, would look something like this:

$$\hat{y} = 44.418 - 35.133 \times x^{(1)} - 13.518 \times x^{(2)} + 0.766 \times x^{(3)}$$

Building blocks

Building blocks

In general, a **learning algorithm** has the following building blocks.

- A **model**, often consisting of a set of **weights** whose values will be **"learnt"**.

Building blocks

In general, a **learning algorithm** has the following building blocks.

- A **model**, often consisting of a set of **weights** whose values will be **"learnt"**.
- An **objective function**.

Building blocks

In general, a **learning algorithm** has the following building blocks.

- A **model**, often consisting of a set of **weights** whose values will be **"learnt"**.
- An **objective function**.
  - In the case of a **regression**, this is often a **loss function**, a function that quantifies misclassification. The **Root Mean Square Error** is a common loss function for regression problems.

$$\sqrt{\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2}$$

Building blocks

In general, a **learning algorithm** has the following building blocks.

- A **model**, often consisting of a set of **weights** whose values will be **"learnt"**.
- An **objective function**.
  - In the case of a **regression**, this is often a **loss function**, a function that quantifies misclassification. The **Root Mean Square Error** is a common loss function for regression problems.

$$\sqrt{\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2}$$

- **Optimization** algorithm

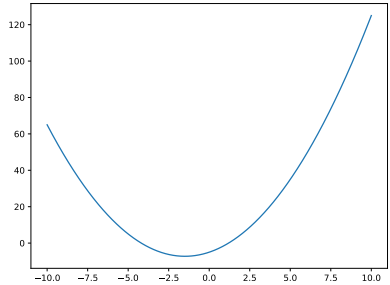
Optimization

- **Until** some termination criteria is met [1]:
  - **Evaluate** the loss function, comparing $h(x_i)$ to y_i.
  - **Make small changes to the weights**, in a way that reduces that the value of the loss function.

$\Rightarrow$ Let's derive a concrete algorithm called **gradient descent**.

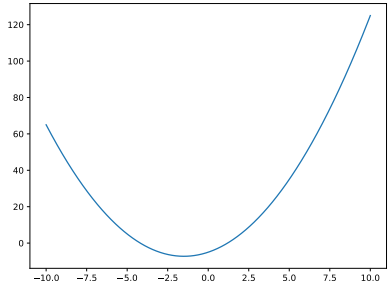
[1] E.g. the value of the loss function no longer decreases or maximum number of iterations.

Derivative



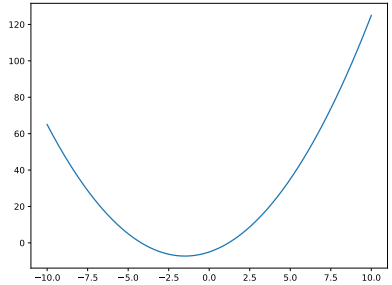
- The **derivative** of a real function describes how changes to the input value(s) will affect the output value.

Derivative



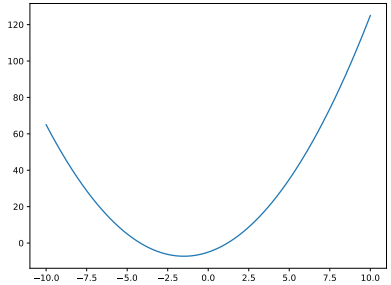
- The **derivative** of a real function describes how changes to the input value(s) will affect the output value.
- We focus on a **single (input) variable function for now**.

Derivative



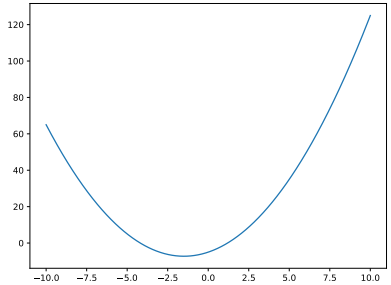
- When evaluated at a **single point**, the **derivative** of a single variable function can be seen as a line **tangent** to the graph of the function.

Derivative



- When the **slope** of the tangent line is **positive** (when the derivative is positive), this means that **increasing the value of the input variable** will **increase the value of the output**.
- Furthermore, the **magnitude** of the derivative indicates how **fast** or **slow** the output will change.

Derivative



- When the **slope** of the tangent line is **negative** (when the derivative is negative), this means that **increasing the value of the input variable** will **decrease the value of the output**.
- Furthermore, the **magnitude** of the derivative indicates how **fast** or **slow** the output will change.

Recall

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

Recall

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- The **Root Mean Square Error** (**RMSE**) is a common loss function for regression problems.

$$\sqrt{\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2}$$

Recall

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- The **Root Mean Square Error** (**RMSE**) is a common loss function for regression problems.

$$\sqrt{\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2}$$

- In practice, minimizing the **Mean Squared Error** (**MSE**) is easier and gives the same result.

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2$$

Gradient descent - single value

- Our **model**:

$$h(x_i) = \theta_0 + \theta_1 x_i^{(1)}$$

Gradient descent - single value

- Our **model**:

$$h(x_i) = \theta_0 + \theta_1 x_i^{(1)}$$

- Our **loss function**:

$$J(\theta_0, \theta_1) = \frac{1}{N} \sum_{1}^{N} [h(x_i) - y_i]^2$$

Gradient descent - single value

- Our **model**:

$$h(x_i) = \theta_0 + \theta_1 x_i^{(1)}$$

- Our **loss function**:

$$J(\theta_0, \theta_1) = \frac{1}{N} \sum_{1}^{N} [h(x_i) - y_i]^2$$

- **Problem**: find the values of θ_0 and θ_1 minimize J.

Gradient descent - single value

- **Gradient descent:**

Gradient descent - single value

- **Gradient descent:**
  - **Initialization:** θ_0 and θ_1 - either with random values or zeros.

Gradient descent - single value

- **Gradient descent:**
  - **Initialization:** θ_0 and θ_1 - either with random values or zeros.
  - **Loop:**

repeat until convergence: $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \text{for } j = 0 \text{ and } j = 1$$

$\}$

Gradient descent - single value

- **Gradient descent:**
  - **Initialization:** θ_0 and θ_1 - either with random values or zeros.
  - **Loop:**

repeat until convergence: $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \text{for } j = 0 \text{ and } j = 1$$

$\}$

- α is called the **learning rate** - this is the size of each step.

Gradient descent - single value

- **Gradient descent:**
  - **Initialization:** θ_0 and θ_1 - either with random values or zeros.
  - **Loop:**

repeat until convergence: $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \text{for } j = 0 \text{ and } j = 1$$

$\}$

- α is called the **learning rate** - this is the size of each step.
- $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is the **partial derivative** with respect to θ_j.

Gradient descent - single value

- **Gradient descent:**
  - **Initialization:** θ_0 and θ_1 - either with random values or zeros.
  - **Loop:**

repeat until convergence: $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \text{for } j = 0 \text{ and } j = 1$$

$\}$

- α is called the **learning rate** - this is the size of each step.
- $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is the **partial derivative** with respect to θ_j.
- For the algorithm to be mathematically sound, all the θ_j must be updated **simultaneously**.

Partial derivatives

Given

$$J(\theta_0, \theta_1) = \frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 = \frac{1}{N}\sum_{1}^{N}[\theta_0 + \theta_1 x_i - y_i]^2$$

We have

$$\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1) = \frac{2}{N}\sum_{i=1}^{N}(\theta_0 - \theta_1 x_i - y_i)$$

and

$$\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1) = \frac{2}{N}\sum_{i=1}^{N} x_i\,(\theta_0 + \theta_1 x_i - y_i)$$

Multivariate linear regression

$$h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \theta_3 x_i^{(3)} + \cdots + \theta_D x_i^{(D)}$$

$x_i^{(j)}$ = value of the feature j in the ith example

D = the number of features

Gradient descent - multivariate

The new **loss function** is

$$J(\theta_0, \theta_1, \ldots, \theta_D) = \frac{1}{N} \sum_{i=1}^{N} \left(h(x_i) - y_i\right)^2$$

Its **partial derivative**:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{2}{N} \sum_{i=1}^{N} x_i^{(j)} \left(\theta x_i - y_i\right)$$

where θ, x_i and y_i are vectors, and θx_i is a vector operation!

Gradient vector

The vector containing the partial derivative of J (with respect to θ_j, for $j \in \{0, 1 \dots D\}$) is called the **gradient vector**.

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_D} J(\theta) \end{pmatrix}$$

- This vector gives the direction of the **steepest ascent**.

Gradient vector

The vector containing the partial derivative of J (with respect to θ_j, for $j \in \{0, 1 \dots D\}$) is called the **gradient vector**.

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_D} J(\theta) \end{pmatrix}$$

- This vector gives the direction of the **steepest ascent**.
- It gives it name to the **gradient descent** algorithm:

$$\theta' = \theta - \alpha \nabla_\theta J(\theta)$$

Gradient descent - multivariate

The gradient descent algorithm becomes:

repeat until convergence: $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \ldots, \theta_D)$$

for $j \in [0, \ldots, D]$ **(update simultaneously)**

$\}$

Gradient descent - multivariate

The gradient descent algorithm becomes:

repeat until convergence: $\{$

$$\theta_0 := \theta_0 - \alpha \frac{2}{N} \sum_{i=1}^{N} x_i^0 (h(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{2}{N} \sum_{i=1}^{N} x_i^1 (h(x_i) - y_i)$$

$$\theta_2 := \theta_2 - \alpha \frac{2}{N} \sum_{i=1}^{N} x_i^2 (h(x_i) - y_i)$$

$$\dots$$

$\}$

Assumptions

What were our **assumptions**?

- The (objective/loss) function is **differentiable**.

Local vs. global

- A function is **convex** if for any pair of points on the graph of the function, the line connecting these two points lies above or on the graph [2]

[2] It would be convex downward or concave if those lines were below or on the graph of the function.

Local vs. global

- A function is **convex** if for any pair of points on the graph of the function, the line connecting these two points lies above or on the graph [2]
  - A **convex** function has a **single** minimum.

[2] It would be convex downward or concave if those lines were below or on the graph of the function.

Local vs. global

- A function is **convex** if for any pair of points on the graph of the function, the line connecting these two points lies above or on the graph [2]
  - A **convex** function has a **single** minimum.
    - The loss function for the linear regression (MSE) is convex.

[2] It would be convex downward or concave if those lines were below or on the graph of the function.

Local vs. global

- A function is **convex** if for any pair of points on the graph of the function, the line connecting these two points lies above or on the graph [2]
  - A **convex** function has a **single** minimum.
    - The loss function for the linear regression (MSE) is convex.
- For functions that are not convex, the gradient descent algorithm converges to a **local** minimum.

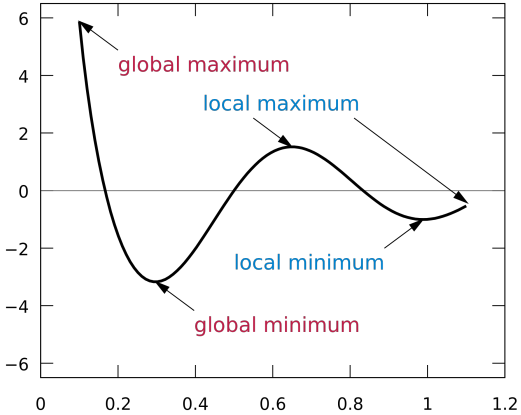
[2] It would be convex downward or concave if those lines were below or on the graph of the function.

Local vs. global

- A function is **convex** if for any pair of points on the graph of the function, the line connecting these two points lies above or on the graph [2]
  - A **convex** function has a **single** minimum.
    - The loss function for the linear regression (MSE) is convex.
- For functions that are not convex, the gradient descent algorithm converges to a **local** minimum.
- The loss function generally used with linear or logistic regressions, and Support Vector Machines (SVM) are convex, but not the ones for artificial neural networks.

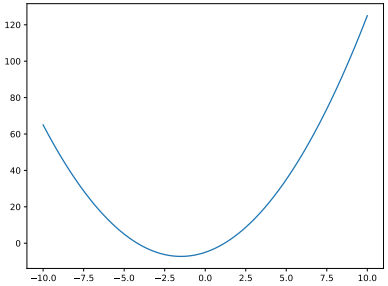
[2] It would be convex downward or concave if those lines were below or on the graph of the function.

Local vs. global



Source: https://commons.wikimedia.org/wiki/File:Extrema_example.svg

About the learning rate



- **Small steps**, low values for α, will make the algorithm **converge slowly**.
- **Large steps**, might cause the algorithm to **diverge**.
- Notice how the algorithm **slows down** naturally when approaching a minimum.

Batch gradient descent

- To be more precise, this algorithm is known as **batch gradient descent** since for each iteration, it processes the "whole batch" of training examples.
- Literature suggests that the algorithm might take more time to converge if the features are on different scales.

Batch gradient descent - drawback

- The **batch gradient descent** algorithm becomes very **slow** as the **number of training examples increases**.

Batch gradient descent - drawback

- The **batch gradient descent** algorithm becomes very **slow** as the **number of training examples increases**.
  - This is because **all** the training data is seen at **each iteration**. The algorithm is generally ran for a fixed number of iterations, say 1000.

Stochastic Gradient Descent

- The **stochastic gradient descent** algorithm randomly selects **one** training instance to calculate its gradient.

```
epochs = 10
for epoch in range(epochs):
    for i in range(N):
        selection = np.random.randint(N)
        # Calculate the gradient using selection
        # Update the weights
```

Stochastic Gradient Descent

- The **stochastic gradient descent** algorithm randomly selects **one** training instance to calculate its gradient.

```
epochs = 10
for epoch in range(epochs):
    for i in range(N):
        selection = np.random.randint(N)
        # Calculate the gradient using selection
        # Update the weights
```

  - This allows it to work with large training sets.

Stochastic Gradient Descent

- The **stochastic gradient descent** algorithm randomly selects **one** training instance to calculate its gradient.

```
epochs = 10
for epoch in range(epochs):
    for i in range(N):
        selection = np.random.randint(N)
        # Calculate the gradient using selection
        # Update the weights
```

  - This allows it to work with large training sets.
  - Its trajectory is not as regular as the batch algorithm.

Stochastic Gradient Descent

- The **stochastic gradient descent** algorithm randomly selects **one** training instance to calculate its gradient.

```
epochs = 10
for epoch in range(epochs):
    for i in range(N):
        selection = np.random.randint(N)
        # Calculate the gradient using selection
        # Update the weights
```

  - This allows it to work with large training sets.
  - Its trajectory is not as regular as the batch algorithm.
    - Because of its bumpy trajectory, it is often **better at finding the global minima**, when compared to batch.

Stochastic Gradient Descent

- The **stochastic gradient descent** algorithm randomly selects **one** training instance to calculate its gradient.

```
epochs = 10
for epoch in range(epochs):
    for i in range(N):
        selection = np.random.randint(N)
        # Calculate the gradient using selection
        # Update the weights
```

  - This allows it to work with large training sets.
  - Its trajectory is not as regular as the batch algorithm.
    - Because of its bumpy trajectory, it is often **better at finding the global minima**, when compared to batch.
    - Its bumpy trajectory makes it **bounce around the local minima**.

Stochastic Gradient Descent

- The **stochastic gradient descent** algorithm randomly selects **one** training instance to calculate its gradient.

```
epochs = 10
for epoch in range(epochs):
    for i in range(N):
        selection = np.random.randint(N)
        # Calculate the gradient using selection
        # Update the weights
```

  - This allows it to work with large training sets.
  - Its trajectory is not as regular as the batch algorithm.
    - Because of its bumpy trajectory, it is often **better at finding the global minima**, when compared to batch.
    - Its bumpy trajectory makes it **bounce around the local minima**.
    - A way around this is to decrease the learning rate as the number of epoch increases - this is called a **learning schedule**.

Stochastic Gradient Descent (SGD)

- It important that the examples are either **selected randomly** or **shuffled** before running the algorithm to make sure that the algorithm converges towards the global minima.

Mini-batch gradient descent

- At each step, rather than selecting one training example, as SGD does, **mini-batch gradient descent** randomly selects a **small number** of training examples to compute the gradients.

Mini-batch gradient descent

- At each step, rather than selecting one training example, as SGD does, **mini-batch gradient descent** randomly selects a **small number** of training examples to compute the gradients.
- Its trajectory is more regular, compared to SGD.

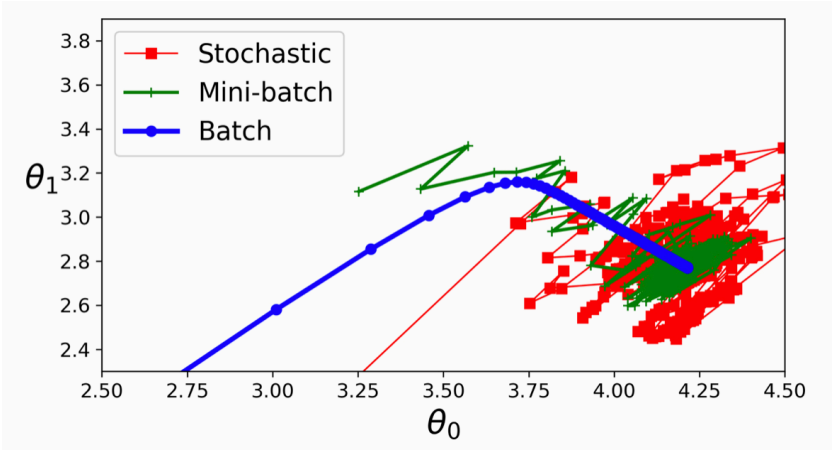
Mini-batch gradient descent

- At each step, rather than selecting one training example, as SGD does, **mini-batch gradient descent** randomly selects a **small number** of training examples to compute the gradients.
- Its trajectory is more regular, compared to SGD.
  - As the size of the mini-batches increases, the algorithm is more and more similar to bach gradient descent, which uses all the examples at each step.

Mini-batch gradient descent

- At each step, rather than selecting one training example, as SGD does, **mini-batch gradient descent** randomly selects a **small number** of training examples to compute the gradients.
- Its trajectory is more regular, compared to SGD.
  - As the size of the mini-batches increases, the algorithm is more and more similar to bach gradient descent, which uses all the examples at each step.
- It can take advantages of the hardware acceleration of matrix operations, in particular GPUs.

Batch, stochastic, and mini-batch



Source: Géron 2019, Figure 4.11

Normal Equation

Briefly...

- For some loss functions, a **closed-form solution** exists, i.e. the problem can be solved analytically.

Normal Equation

Briefly...

- For some loss functions, a **closed-form solution** exists, i.e. the problem can be solved analytically.
- This is the case for a **quadratic** function, such as the **mean squared error** (MSE).

Normal Equation

Briefly...

- For some loss functions, a **closed-form solution** exists, i.e. the problem can be solved analytically.
- This is the case for a **quadratic** function, such as the **mean squared error** (MSE).
- However, this involves computing an inverse matrix, which in turns involves computing the **singular value decomposition** (SVD) of the matrix.

Normal Equation

Briefly...

- For some loss functions, a **closed-form solution** exists, i.e. the problem can be solved analytically.
- This is the case for a **quadratic** function, such as the **mean squared error** (MSE).
- However, this involves computing an inverse matrix, which in turns involves computing the **singular value decomposition** (SVD) of the matrix.
  - Such algorithms have a computational time complexity between $\mathcal{O}(D^{2.4})$ to $\mathcal{O}(D^3)$, where D is the number of features.

Normal Equation

Briefly...

- For some loss functions, a **closed-form solution** exists, i.e. the problem can be solved analytically.
- This is the case for a **quadratic** function, such as the **mean squared error** (MSE).
- However, this involves computing an inverse matrix, which in turns involves computing the **singular value decomposition** (SVD) of the matrix.
  - Such algorithms have a computational time complexity between $\mathcal{O}(D^{2.4})$ to $\mathcal{O}(D^3)$, where D is the number of features.
  - However, these algorithms are linear with respect to the number of examples, N.

Summary

- **Normal Equation** is very slow when the number of features is large, say 100,000. However, the algorithm scales linearly with the number of examples.
- **Batch gradient descent** is **slow**, cannot be run on large data sets where **out-of-core** support is needed can work with a large number of features.
- **Stochastic gradient descent** is **fast**, can handle a large number of examples.
- **Mini-batch gradient descent** is **fast**, can handle a large number of examples. Takes advantage of hardware acceleration.

All three are implemented by **SGDRegressor** in **Scikit-Learn**.

Optimization and deep nets

We will briefly revisit the subject when talking about **deep artificial neural networks** for which **specialized optimization algorithms** exist.

- Momentum Optimization
- Nesterov Accelerated Gradient
- AdaGrad
- RMSProp
- Adam and Nadam

Final word

- Optimization is a vast subject. Other algorithms exist and are used in other contexts.
- Including
  - Particle swarm optimization (PSO), genetic algorithms (GAs), and artificial bee colony (ABC) algorithms.

Linear regression - summary

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

Linear regression - summary

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- The **Mean Squared Error** (**MSE**) is a

$$\frac{1}{N} \sum_{1}^{N} [h(x_i) - y_i]^2$$

Linear regression - summary

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- The **Mean Squared Error** (**MSE**) is a

$$\frac{1}{N} \sum_{1}^{N} [h(x_i) - y_i]^2$$

- **Batch**, **stochastic**, or **mini-batch gradient descent** can be used to find "optimal" values for the weights, θ_j for $j \in 0, 1 \ldots D$.

Linear regression - summary

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- The **Mean Squared Error** (**MSE**) is a

$$\frac{1}{N} \sum_{1}^{N} [h(x_i) - y_i]^2$$

- **Batch**, **stochastic**, or **mini-batch gradient descent** can be used to find "optimal" values for the weights, θ_j for $j \in 0, 1 \ldots D$.
- The result is a **regressor**. A function that can be used to predict the y value (the label) for some unseen example x.

TO DO 2020

- Consider saying a few works about **autodiff** - See Géron §D.

Prologue

Summary

- An **optimization** algorithm is used to find "optimal" values for the parameters of the **linear model** so as to **minimize** the value of the **losss function**

Summary

- An **optimization** algorithm is used to find "optimal" values for the parameters of the **linear model** so as to **minimize** the value of the **losss function**
- The **gradient** of the **loss function** plays a central role in the gradient descent algorithm. For each feature weight, it informs about the sign and magnitude of the required change.

Summary

- An **optimization** algorithm is used to find "optimal" values for the parameters of the **linear model** so as to **minimize** the value of the **losss function**
- The **gradient** of the **loss function** plays a central role in the gradient descent algorithm. For each feature weight, it informs about the sign and magnitude of the required change.
- The **learning rate** controls how fast or slow the algorithm learns. The algorithm might diverge is the learning rate is too high.

Summary

- An **optimization** algorithm is used to find "optimal" values for the parameters of the **linear model** so as to **minimize** the value of the **losss function**
- The **gradient** of the **loss function** plays a central role in the gradient descent algorithm. For each feature weight, it informs about the sign and magnitude of the required change.
- The **learning rate** controls how fast or slow the algorithm learns. The algorithm might diverge is the learning rate is too high.
- **Batch** gradient descent has a smooth trajectory, but becomes very slow when the number of examples is large.

Summary

- An **optimization** algorithm is used to find "optimal" values for the parameters of the **linear model** so as to **minimize** the value of the **losss function**
- The **gradient** of the **loss function** plays a central role in the gradient descent algorithm. For each feature weight, it informs about the sign and magnitude of the required change.
- The **learning rate** controls how fast or slow the algorithm learns. The algorithm might diverge is the learning rate is too high.
- **Batch** gradient descent has a smooth trajectory, but becomes very slow when the number of examples is large.
- **Stochastic** and **mini-batch** gradient descent are good alternatives that can handle large amounts of training examples.

Next module

- **Feature engineering**, **data imputation**, **dimensionality reduction**.

References


Aurélien Géron.
Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.
O'Reilly Media, 2nd edition, 2019.


Andriy Burkov.
The Hundred-Page Machine Learning Book.
Andriy Burkov, 2019.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EE**CS**)
University of Ottawa