

Deep Learning & Applied AI

Stochastic gradient descent

Emanuele Rodolà
rodola@di.uniroma1.it



SAPIENZA
UNIVERSITÀ DI ROMA

Recap

In deep learning, we deal with **highly parametrized models** called **deep neural networks**:

$$f_{\Theta}(\mathbf{x}) = \mathbf{y}$$



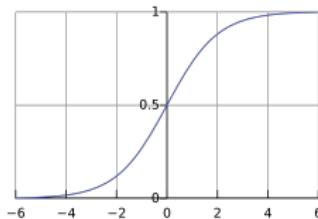
Recap: Logistic regression

What if we want to predict a **category** instead of a value?

$$f_{\Theta}(\text{ultrasound image}) = \{0, 1\}$$

General idea: Modify the loss to minimize over **categorical values**.

$$\ell_{\Theta}(\{x_i, y_i\}) = - \sum_{i=1}^n y_i \ln(\sigma(ax_i + b)) + (1 - y_i) \ln(1 - \sigma(ax_i + b))$$



Recap: Logistic regression

By looking at the partial derivative:

$$\frac{\partial}{\partial \textcolor{red}{a}} \ln(\sigma(\textcolor{red}{a}x_i + b)) = (1 - \sigma(\textcolor{red}{a}x_i + b))x_i$$

we see that the parameters enter the gradient in a **nonlinear** way.

Thus:

- $\nabla \ell_{\Theta} = 0$ is **not a linear system** that we can solve easily.
- $\nabla \ell_{\Theta} = 0$ is a **transcendental equation** \Rightarrow no analytical solution.

model	loss	solution
linear regression	convex	least squares
linear regression + Tikhonov	convex	least squares
logistic regression	convex	nonlinear optimization

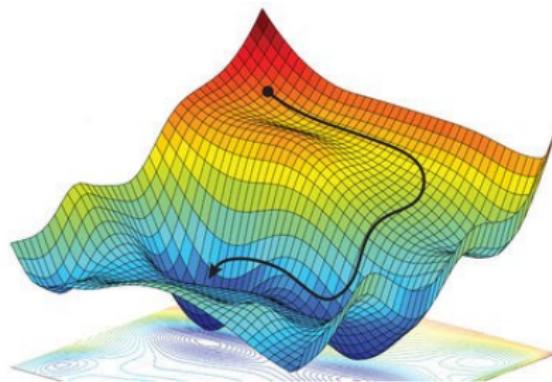
Gradient descent: Intuition

Gradient descent is a **first-order** iterative minimization algorithm.

Gradient descent: Intuition

Gradient descent is a **first-order** iterative minimization algorithm.

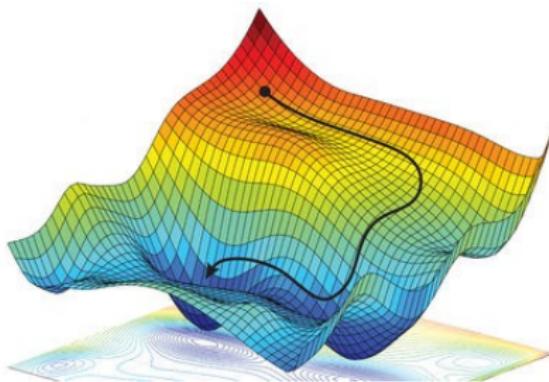
Example of a loss function $\ell_{\Theta} : \mathbb{R}^2 \rightarrow \mathbb{R}$:



Gradient descent: Intuition

Gradient descent is a **first-order** iterative minimization algorithm.

Example of a loss function $\ell_{\Theta} : \mathbb{R}^2 \rightarrow \mathbb{R}$:



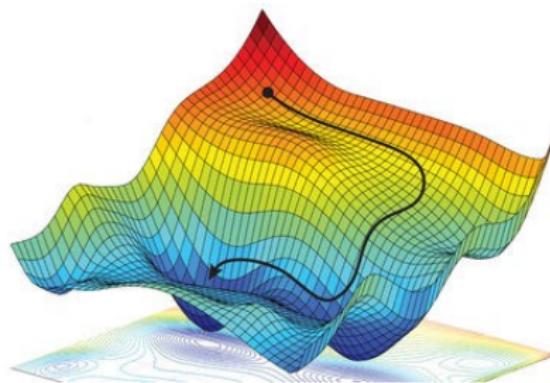
Overall idea: Move where the function decreases the most.

- ① Start from some point $\Theta^{(0)} \in \mathbb{R}^2$.

Gradient descent: Intuition

Gradient descent is a **first-order** iterative minimization algorithm.

Example of a loss function $\ell_{\Theta} : \mathbb{R}^2 \rightarrow \mathbb{R}$:



Overall idea: Move where the function decreases the most.

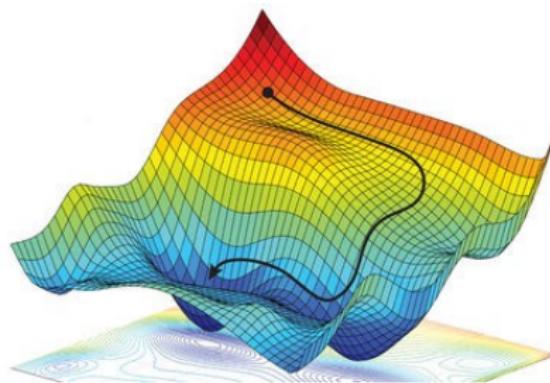
- ① Start from some point $\Theta^{(0)} \in \mathbb{R}^2$.
- ② Iteratively compute:

$$\Theta^{(t+1)} = \Theta^{(t)} - \alpha \nabla \ell_{\Theta^{(t)}}$$

Gradient descent: Intuition

Gradient descent is a **first-order** iterative minimization algorithm.

Example of a loss function $\ell_{\Theta} : \mathbb{R}^2 \rightarrow \mathbb{R}$:



Overall idea: Move where the function decreases the most.

- ① Start from some point $\Theta^{(0)} \in \mathbb{R}^2$.
- ② Iteratively compute:

$$\Theta^{(t+1)} = \Theta^{(t)} - \alpha \nabla \ell_{\Theta^{(t)}}$$

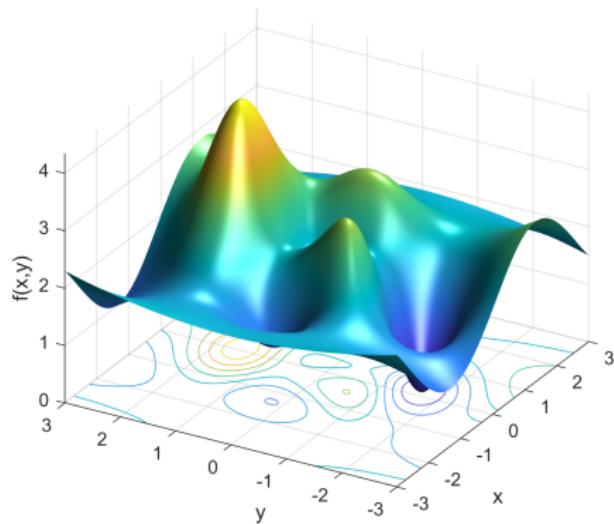
- ③ Stop when a minimum is reached.

Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

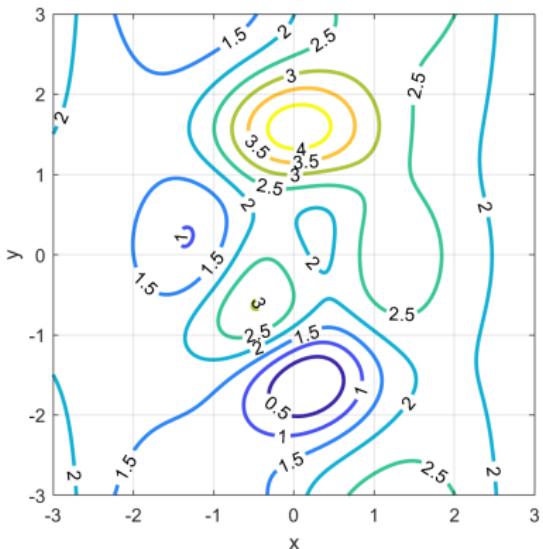
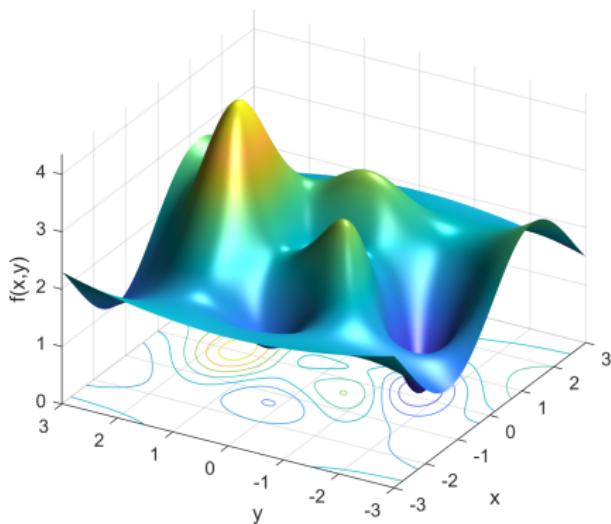
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



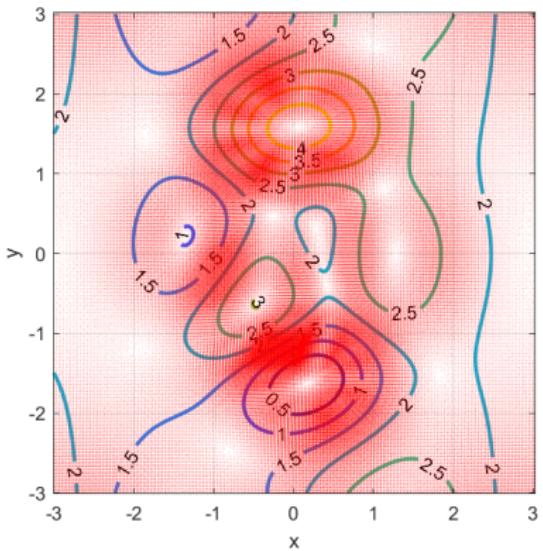
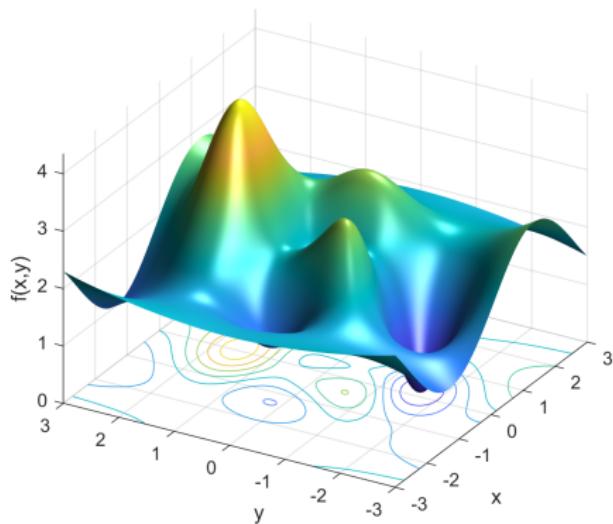
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



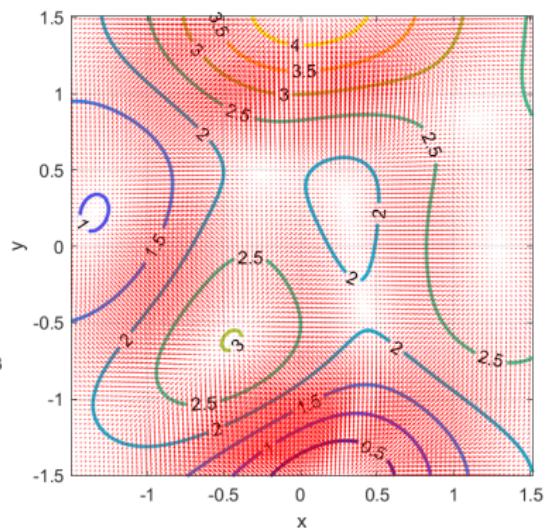
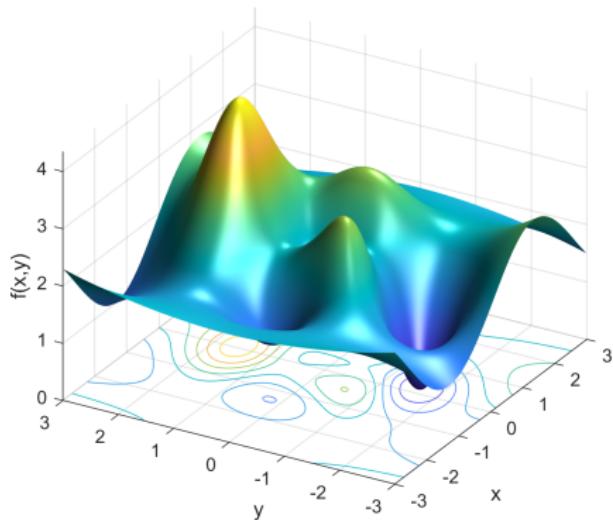
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



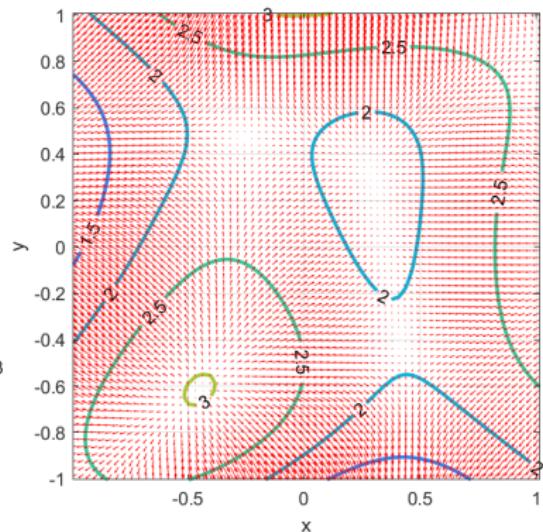
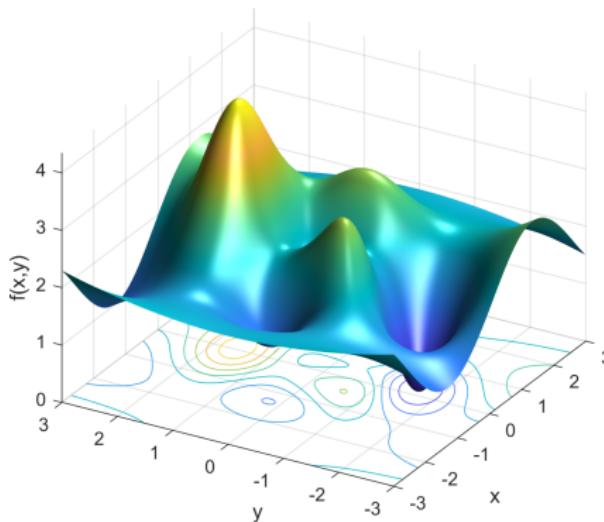
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



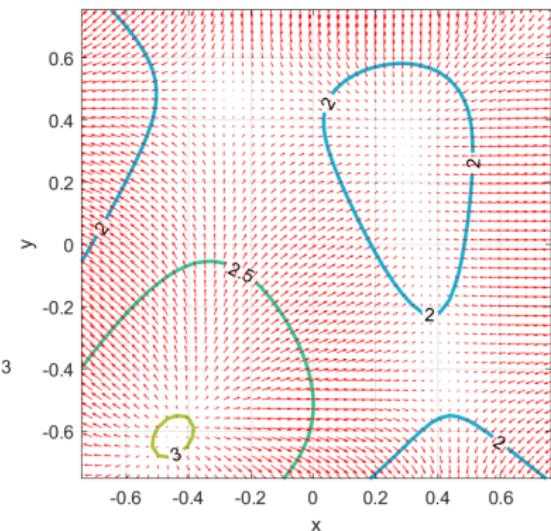
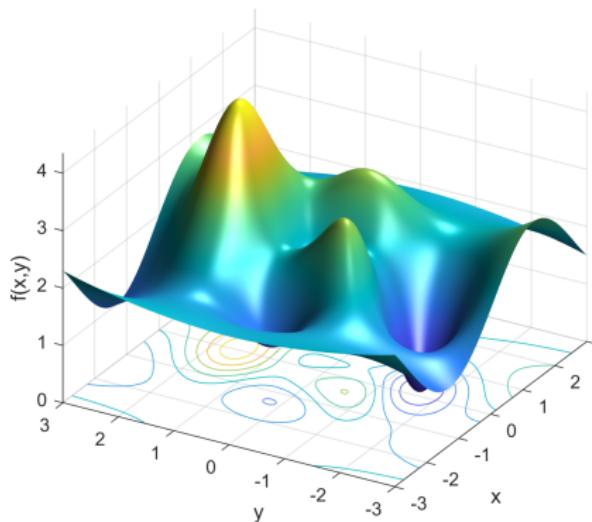
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



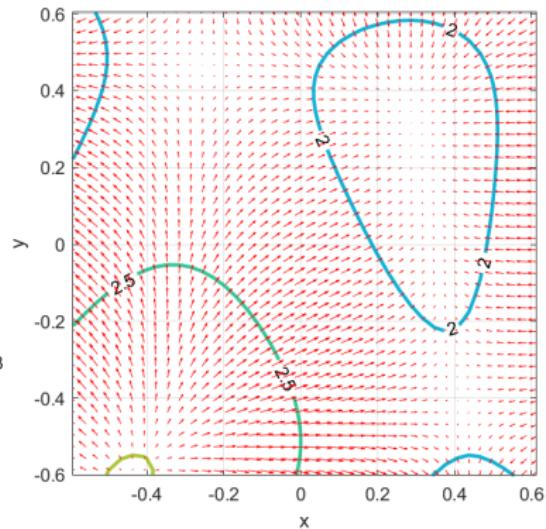
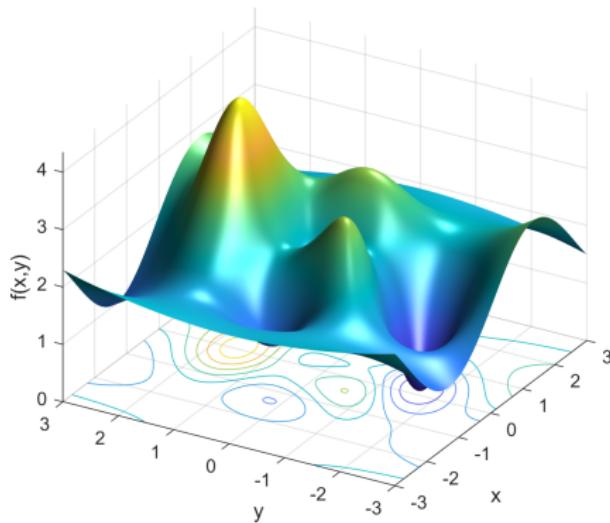
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



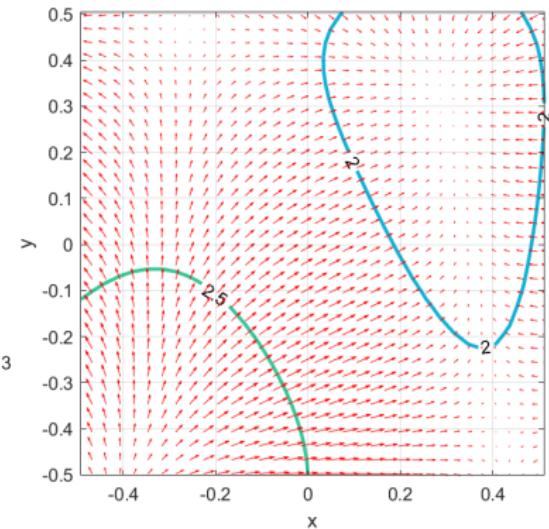
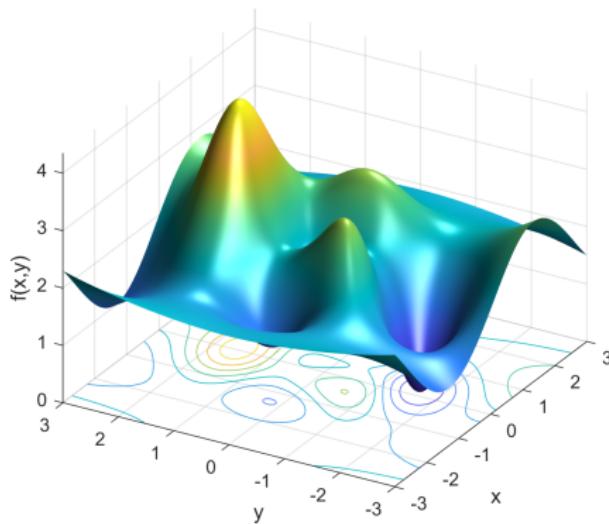
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



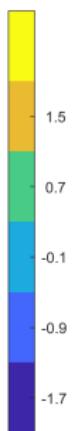
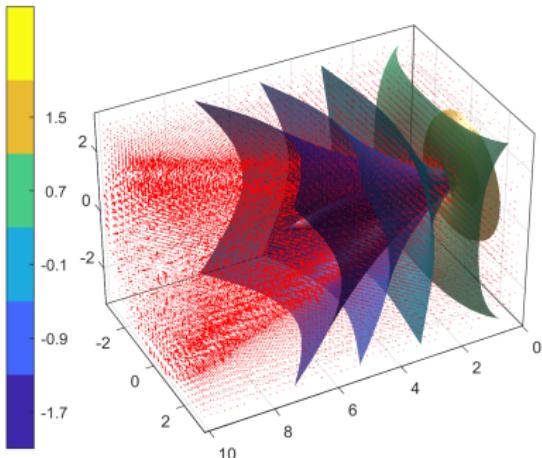
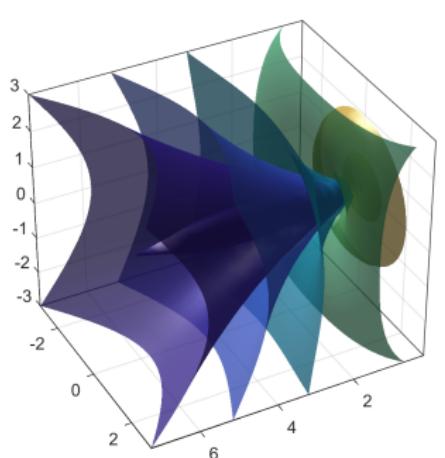
Gradient descent

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$



Gradient descent: High dimensions

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

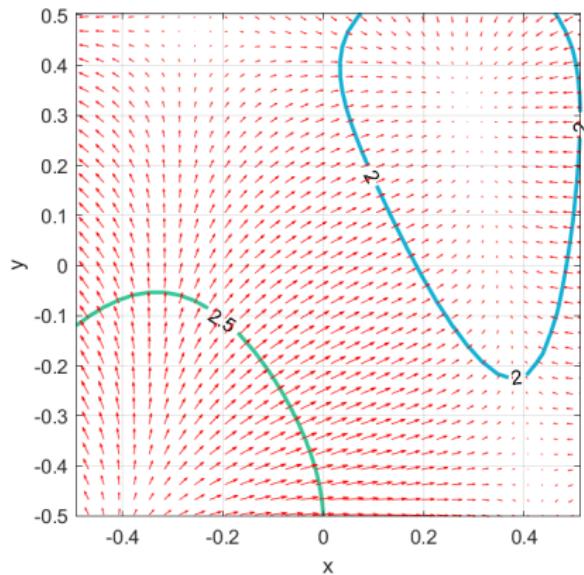


All we say is also valid in $\gg 2$ dimensions.

Gradient descent: Orthogonality

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

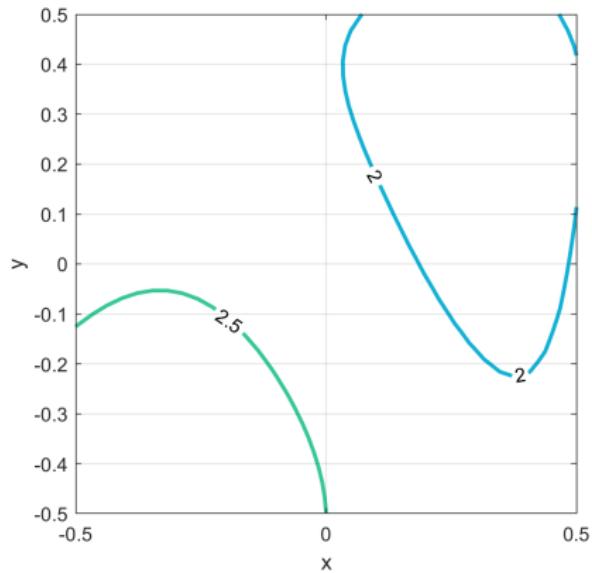
The gradient is **orthogonal** to level curves / level surfaces.



Gradient descent: Orthogonality

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

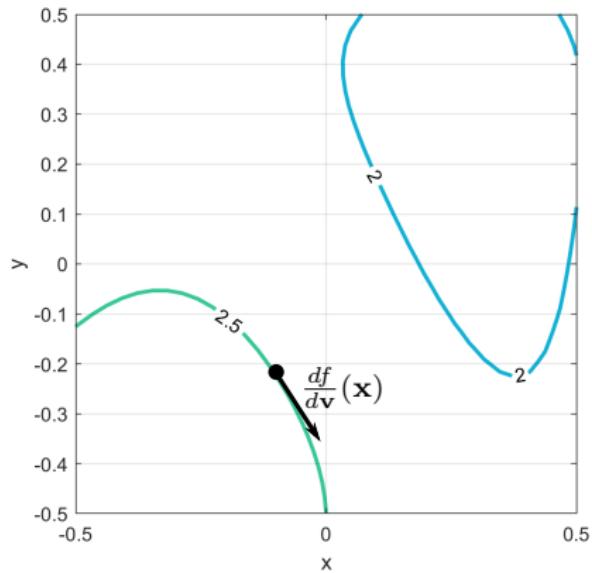
The gradient is **orthogonal** to level curves / level surfaces.



Gradient descent: Orthogonality

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

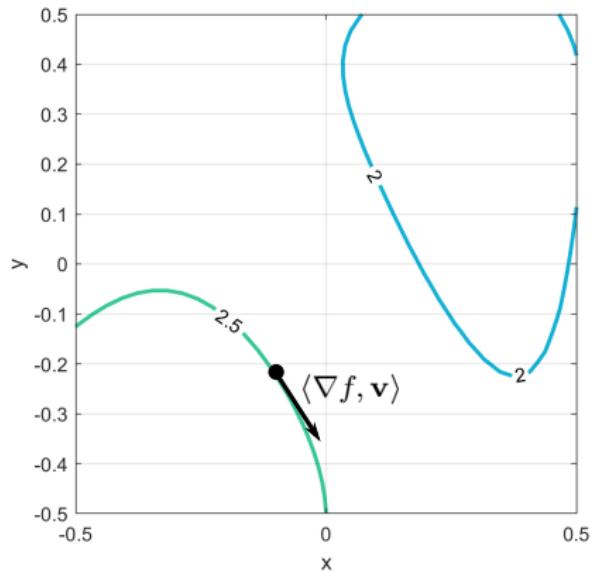
The gradient is **orthogonal** to level curves / level surfaces.



Gradient descent: Orthogonality

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

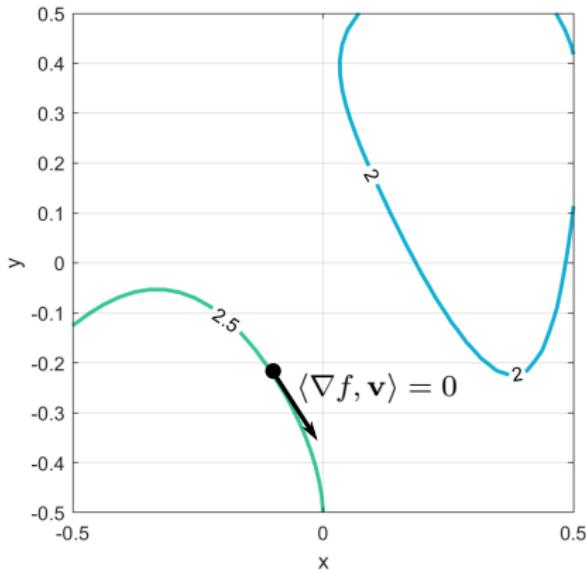
The gradient is **orthogonal** to level curves / level surfaces.



Gradient descent: Orthogonality

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

The gradient is **orthogonal** to level curves / level surfaces.

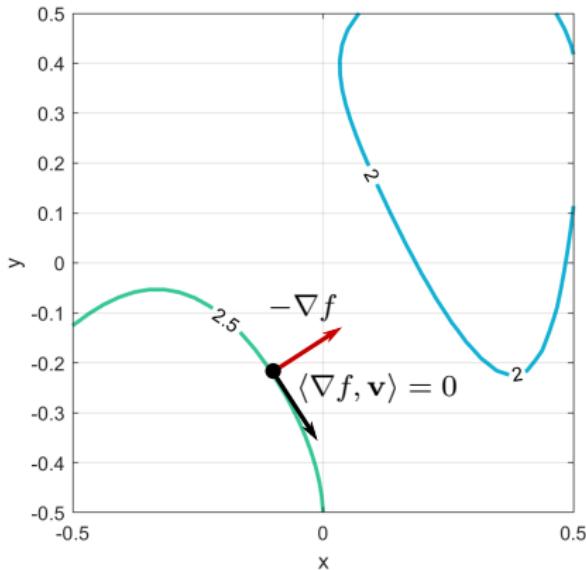


The directional derivative is **zero** along isocurves.

Gradient descent: Orthogonality

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

The gradient is **orthogonal** to level curves / level surfaces.



The directional derivative is **zero** along isocurves.

Gradient descent: Differentiability

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

Gradient descent requires f to be **differentiable** at all points.

Warning:

Gradient descent: Differentiability

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

Gradient descent requires f to be **differentiable** at all points.

Warning:

f has partial (or even directional) derivatives $\not\Rightarrow f$ is differentiable

Gradient descent: Differentiability

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

Gradient descent requires f to be **differentiable** at all points.

Warning:

f has partial (or even directional) derivatives $\not\Rightarrow f$ is differentiable

f has **continuous gradient** $\Rightarrow f$ is differentiable

Gradient descent: Differentiability

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

Gradient descent requires f to be **differentiable** at all points.

Warning:

f has partial (or even directional) derivatives $\not\Rightarrow f$ is differentiable

f has **continuous gradient** $\Rightarrow f$ is differentiable

See examples at: https://mathinsight.org/differentiability_multivariable_subtleties

Gradient descent: Stationary points

A **stationary point** is such that:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})^0$$

Gradient descent “gets stuck” at stationary points.

Gradient descent: Stationary points

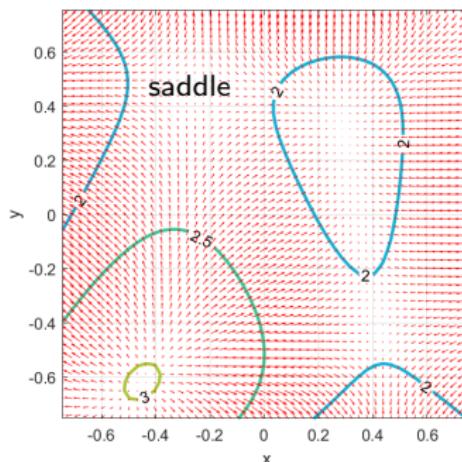
A **stationary point** is such that:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})^0$$

Gradient descent “gets stuck” at stationary points.

However:

- Stationary point $\not\Rightarrow$ local minimum



Gradient descent: Stationary points

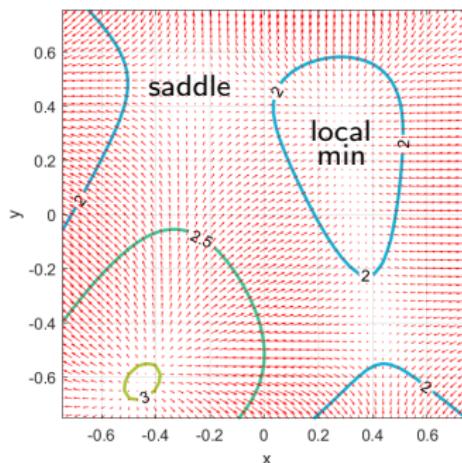
A **stationary point** is such that:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})^0$$

Gradient descent “gets stuck” at stationary points.

However:

- Stationary point $\not\Rightarrow$ local minimum $\not\Rightarrow$ global minimum.



Gradient descent: Stationary points

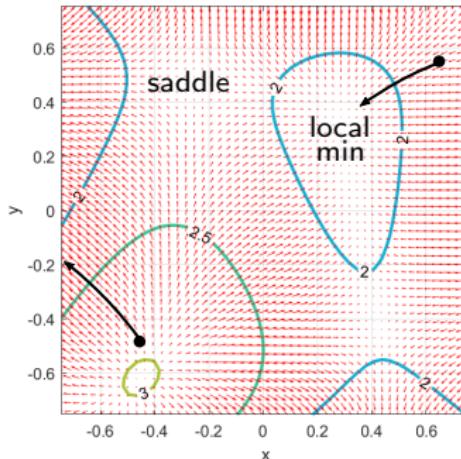
A **stationary point** is such that:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})^0$$

Gradient descent “gets stuck” at stationary points.

However:

- Stationary point $\not\Rightarrow$ local minimum $\not\Rightarrow$ global minimum.
- Which stationary point depends on the **initialization**.



Gradient descent: Learning rate

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

The parameter $\alpha > 0$ is also called learning rate in ML.

Gradient descent: Learning rate

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

The parameter $\alpha > 0$ is also called learning rate in ML.

Remark: The length of a step is not simply α , but $\alpha \|\nabla f\|$.

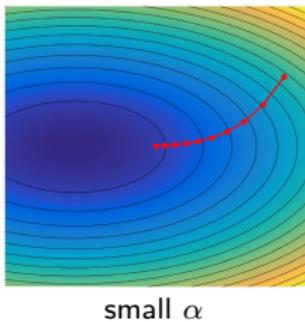
Gradient descent: Learning rate

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

The parameter $\alpha > 0$ is also called learning rate in ML.

Remark: The length of a step is not simply α , but $\alpha \|\nabla f\|$.

- Too small: slow convergence speed



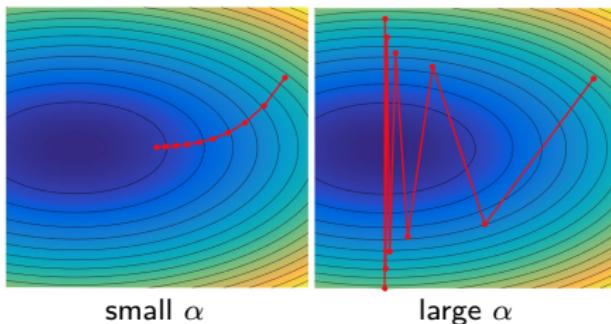
Gradient descent: Learning rate

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

The parameter $\alpha > 0$ is also called learning rate in ML.

Remark: The length of a step is not simply α , but $\alpha \|\nabla f\|$.

- Too small: slow convergence speed
- Too big: risk of **overshooting**



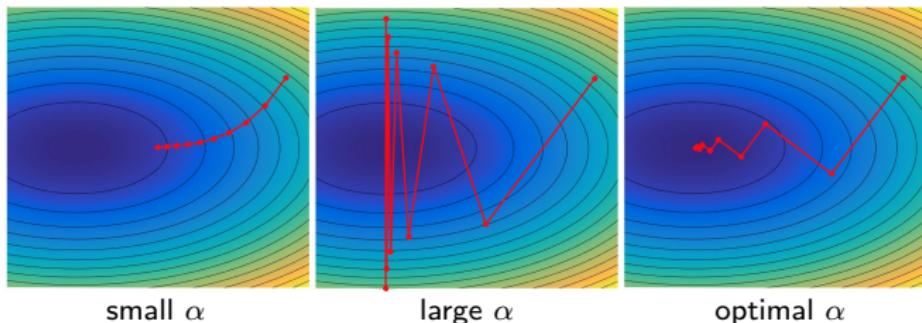
Gradient descent: Learning rate

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

The parameter $\alpha > 0$ is also called learning rate in ML.

Remark: The length of a step is not simply α , but $\alpha \|\nabla f\|$.

- Too small: slow convergence speed
- Too big: risk of **overshooting**
- Optimal values can be found via **line search** algorithms



$$\arg \min_{\alpha} f(\mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)}))$$

Decay and momentum

The learning rate can be **adaptive** or follow a **schedule**.

Decay and momentum

The learning rate can be **adaptive** or follow a **schedule**.

- Decrease α according to a **decay** parameter ρ :

Examples:

$$\alpha^{(t+1)} = \left(1 - \frac{t}{\rho}\right)\alpha^{(0)} + \frac{t}{\rho}\alpha^{(\rho)}, \quad \alpha^{(t+1)} = \frac{\alpha^{(t)}}{1 + \rho t}, \quad \alpha^{(t+1)} = \alpha^{(0)}e^{-\rho t}$$

Decay and momentum

The learning rate can be **adaptive** or follow a **schedule**.

- Decrease α according to a **decay** parameter ρ :

Examples:

$$\alpha^{(t+1)} = \left(1 - \frac{t}{\rho}\right)\alpha^{(0)} + \frac{t}{\rho}\alpha^{(\rho)}, \quad \alpha^{(t+1)} = \frac{\alpha^{(t)}}{1 + \rho t}, \quad \alpha^{(t+1)} = \alpha^{(0)}e^{-\rho t}$$

- Accumulate past gradients and keep moving in their direction:

$$\mathbf{v}^{(t+1)} = \lambda \mathbf{v}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)}) \quad \text{momentum}$$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \mathbf{v}^{(t+1)}$$

Decay and momentum

The learning rate can be **adaptive** or follow a **schedule**.

- Decrease α according to a **decay** parameter ρ :

Examples:

$$\alpha^{(t+1)} = \left(1 - \frac{t}{\rho}\right)\alpha^{(0)} + \frac{t}{\rho}\alpha^{(\rho)}, \quad \alpha^{(t+1)} = \frac{\alpha^{(t)}}{1 + \rho t}, \quad \alpha^{(t+1)} = \alpha^{(0)}e^{-\rho t}$$

- Accumulate past gradients and keep moving in their direction:

$$\mathbf{v}^{(t+1)} = \lambda \mathbf{v}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)}) \quad \text{momentum}$$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \mathbf{v}^{(t+1)}$$

Step length \propto how **aligned** is the sequence of gradients.

$$\frac{1}{1 - \lambda} \alpha \|\nabla f\|$$

Decay and momentum

The learning rate can be **adaptive** or follow a **schedule**.

- Decrease α according to a **decay** parameter ρ :

Examples:

$$\alpha^{(t+1)} = \left(1 - \frac{t}{\rho}\right)\alpha^{(0)} + \frac{t}{\rho}\alpha^{(\rho)}, \quad \alpha^{(t+1)} = \frac{\alpha^{(t)}}{1 + \rho t}, \quad \alpha^{(t+1)} = \alpha^{(0)}e^{-\rho t}$$

- Accumulate past gradients and keep moving in their direction:

$$\mathbf{v}^{(t+1)} = \lambda \mathbf{v}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)}) \quad \text{momentum}$$

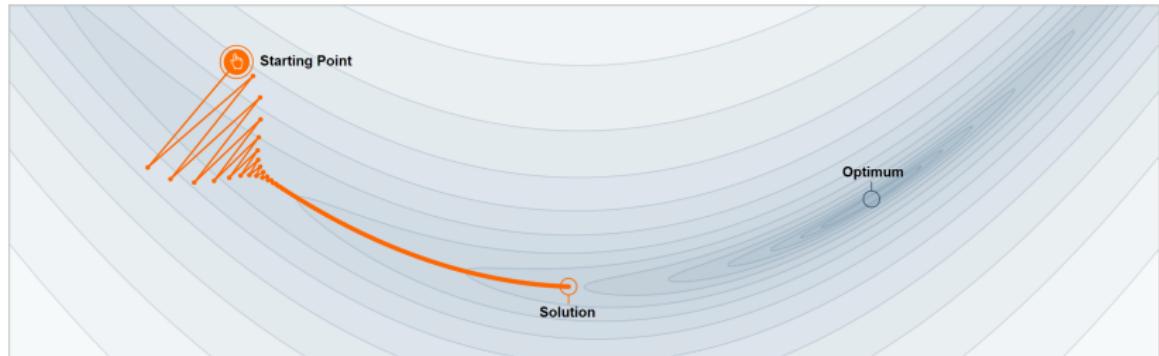
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \mathbf{v}^{(t+1)}$$

Step length \propto how **aligned** is the sequence of gradients.

$$\frac{1}{1 - \lambda} \alpha \|\nabla f\|$$

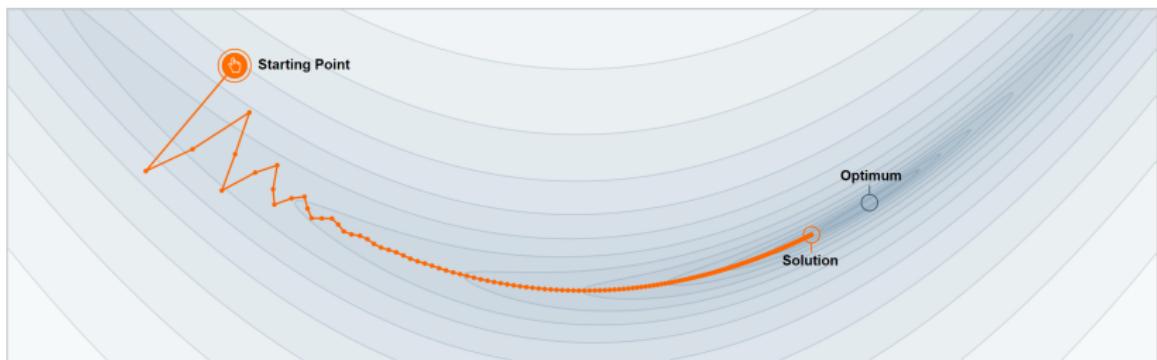
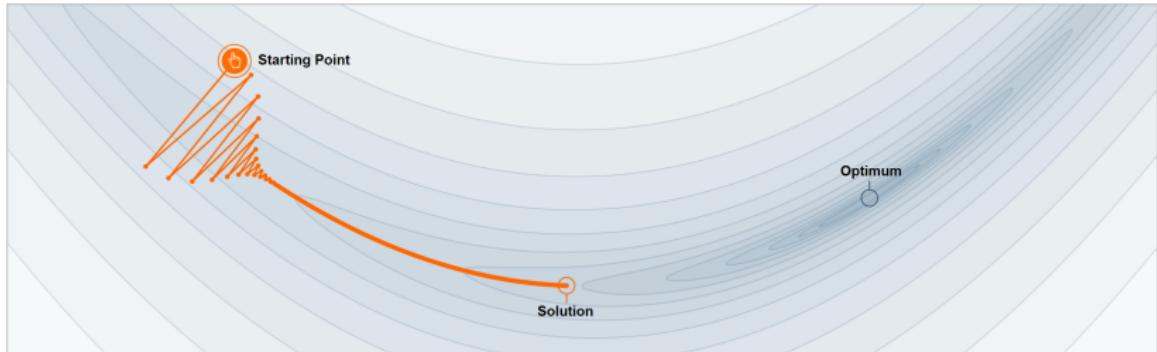
Acceleration effect for big λ + escape from local minima.

Momentum



Goh, "Why momentum really works", Distill 2017

Momentum



Goh, "Why momentum really works", Distill 2017

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)})$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha \nabla f(\mathbf{x}^{(0)})$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha \nabla f(\mathbf{x}^{(0)})$$

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \alpha \nabla f(\mathbf{x}^{(1)})$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\begin{aligned}\mathbf{x}^{(1)} &= \mathbf{x}^{(0)} - \alpha \nabla f(\mathbf{x}^{(0)}) \\ \mathbf{x}^{(2)} &= \mathbf{x}^{(1)} - \alpha \nabla f(\mathbf{x}^{(1)}) \\ &= \mathbf{x}^{(0)} - \alpha \nabla f(\mathbf{x}^{(0)}) - \alpha \nabla f(\mathbf{x}^{(1)})\end{aligned}$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha \nabla f(\mathbf{x}^{(0)})$$

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \alpha \nabla f(\mathbf{x}^{(1)})$$

$$= \mathbf{x}^{(0)} - \alpha \nabla f(\mathbf{x}^{(0)}) - \alpha \nabla f(\mathbf{x}^{(1)})$$

⋮

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \nabla f(\mathbf{x}^{(i)})$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \nabla f(\mathbf{x}^{(i)})$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \nabla f(\mathbf{x}^{(i)})$$

With momentum:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \frac{1 - \lambda^{t+1-i}}{1 - \lambda} \nabla f(\mathbf{x}^{(i)})$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \nabla f(\mathbf{x}^{(i)})$$

With momentum:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \frac{1 - \lambda^{t+1-i}}{1 - \lambda} \nabla f(\mathbf{x}^{(i)})$$

The more general form:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} + \alpha \sum_{i=1}^t \gamma_i^t \nabla f(\mathbf{x}^{(i)}) \quad \text{for some } \gamma_i$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \nabla f(\mathbf{x}^{(i)})$$

With momentum:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \frac{1 - \lambda^{t+1-i}}{1 - \lambda} \nabla f(\mathbf{x}^{(i)})$$

The more general form:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} + \alpha \sum_{i=1}^t \Gamma_i^t \nabla f(\mathbf{x}^{(i)}) \quad \text{for some diag. matrix } \Gamma_i$$

First-order acceleration methods

Let us try to unroll gradient descent:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \nabla f(\mathbf{x}^{(i)})$$

With momentum:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} - \alpha \sum_{i=1}^t \frac{1 - \lambda^{t+1-i}}{1 - \lambda} \nabla f(\mathbf{x}^{(i)})$$

The more general form:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(0)} + \alpha \sum_{i=1}^t \Gamma_i^t \nabla f(\mathbf{x}^{(i)}) \quad \text{for some diag. matrix } \Gamma_i$$

generalizes optimization algorithms like ADAM, AdaGrad, etc.

Gradient descent for deep learning

Gradient descent can be applied to **nonconvex** problems, without optimality guarantees.

Gradient descent for deep learning

Gradient descent can be applied to **nonconvex** problems, without optimality guarantees.

In order to gain **generalization**, the following consideration is crucial:

We are rarely interested in the **global** optimum.

Gradient descent for deep learning

Gradient descent can be applied to **nonconvex** problems, without optimality guarantees.

In order to gain **generalization**, the following consideration is crucial:

We are rarely interested in the **global** optimum.

Even for **convex** problems like:

- Linear regression
- Logistic regression

We get more **efficient** and **numerically stable** solutions.

Gradient descent for deep learning

Gradient descent can be applied to **nonconvex** problems, without optimality guarantees.

In order to gain **generalization**, the following consideration is crucial:

We are rarely interested in the **global** optimum.

Even for **convex** problems like:

- Linear regression (\mathbf{X} can be huge and must be inverted/factorized)
- Logistic regression

We get more **efficient** and **numerically stable** solutions.

Gradient descent for deep learning

Gradient descent can be applied to **nonconvex** problems, without optimality guarantees.

In order to gain **generalization**, the following consideration is crucial:

We are rarely interested in the **global** optimum.

Even for **convex** problems like:

- Linear regression (\mathbf{X} can be huge and must be inverted/factorized)
- Logistic regression (no closed form solution)

We get more **efficient** and **numerically stable** solutions.

Gradient descent for deep learning

In the general DL setting:

Each parameter gets updated so as to **decrease the loss**:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \ell}{\partial \theta_i}$$

The gradient tells us how to modify the parameters.

Gradient descent for deep learning

In the general DL setting:

Each parameter gets updated so as to **decrease the loss**:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \ell}{\partial \theta_i}$$

The gradient tells us how to modify the parameters.

- θ stores the neural network parameters, possibly **millions**

Gradient descent for deep learning

In the general DL setting:

Each parameter gets updated so as to **decrease the loss**:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \ell}{\partial \theta_i}$$

The gradient tells us how to modify the parameters.

- θ stores the neural network parameters, possibly **millions**
- The loss may be **non-convex** and **non-differentiable**

Gradient descent for deep learning

In the general DL setting:

Each parameter gets updated so as to **decrease the loss**:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \ell}{\partial \theta_i}$$

The gradient tells us how to modify the parameters.

- θ stores the neural network parameters, possibly **millions**
- The loss may be **non-convex** and **non-differentiable**
- Be aware of computational aspects (e.g. $\nabla \|\theta\|_1$?)

Stochastic gradient descent

Recall that the loss is usually defined over n training examples:

$$\ell_{\Theta}(\{x_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n (y_i - f_{\Theta}(x_i))^2$$

Stochastic gradient descent

Recall that the loss is usually defined over n training examples:

$$\ell_{\Theta}(\{x_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n \hat{\ell}_{\Theta}(\{x_i, y_i\})$$

Stochastic gradient descent

Recall that the loss is usually defined over n training examples:

$$\ell_{\Theta}(\{x_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n \hat{\ell}_{\Theta}(\{x_i, y_i\})$$

which requires computing the gradient for each term in the summation:

$$\nabla \ell_{\Theta}(\{x_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\{x_i, y_i\})$$

Stochastic gradient descent

Recall that the loss is usually defined over n training examples:

$$\ell_{\Theta}(\{x_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n \hat{\ell}_{\Theta}(\{x_i, y_i\})$$

which requires computing the gradient for each term in the summation:

$$\nabla \ell_{\Theta}(\{x_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\{x_i, y_i\})$$

Two **bottlenecks** make gradient descent impractical:

- Number of examples
- Number of parameters

Wilson and Martinez, "The general inefficiency of batch training for gradient descent learning", Neural Networks 2003

Mini-batches

$$\nabla \ell_{\Theta}(\{x_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\{x_i, y_i\})$$

Mini-batches

$$\nabla \ell_{\Theta}(\mathcal{T}) = \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\mathcal{T})$$

Mini-batches

$$\nabla \ell_{\Theta}(\mathcal{T}) = \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\mathcal{T})$$

Compute $\nabla \ell_{\Theta}$ for a **small** representative subset of $m \ll n$ examples:

$$\frac{1}{m} \sum_{i=1}^m \nabla \hat{\ell}_{\Theta}(\mathcal{B}) \approx \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\mathcal{T})$$

The **mini-batch** $\mathcal{B} \subset \mathcal{T}$ is drawn uniformly.

Mini-batches

$$\nabla \ell_{\Theta}(\mathcal{T}) = \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\mathcal{T})$$

Compute $\nabla \ell_{\Theta}$ for a **small** representative subset of $m \ll n$ examples:

$$\frac{1}{m} \sum_{i=1}^m \nabla \hat{\ell}_{\Theta}(\mathcal{B}) \approx \frac{1}{n} \sum_{i=1}^n \nabla \hat{\ell}_{\Theta}(\mathcal{T})$$

The **mini-batch** $\mathcal{B} \subset \mathcal{T}$ is drawn uniformly.

The true gradient $\nabla \ell_{\Theta}$ is approximated, but with a significant **speed-up**.

Example: MNIST dataset

$$n = 60,000, \ m = 10 \quad \Rightarrow \quad 6,000 \times \text{speedup}$$

Stochastic gradient descent

The algorithm is as follows:

- ➊ Initialize θ .

Stochastic gradient descent

The algorithm is as follows:

- ➊ Initialize θ .
- ➋ Pick a mini-batch \mathcal{B} .

Stochastic gradient descent

The algorithm is as follows:

- ➊ Initialize θ .
- ➋ Pick a mini-batch \mathcal{B} .
- ➌ Update with the downhill step (use momentum if desired):

$$\theta \leftarrow \theta - \alpha \nabla \ell_{\theta}(\mathcal{B})$$

Stochastic gradient descent

The algorithm is as follows:

- ➊ Initialize θ .
- ➋ Pick a mini-batch \mathcal{B} .
- ➌ Update with the downhill step (use momentum if desired):

$$\theta \leftarrow \theta - \alpha \nabla \ell_{\theta}(\mathcal{B})$$

- ➍ Go back to step (2).

Stochastic gradient descent

The algorithm is as follows:

- ➊ Initialize θ .
- ➋ Pick a mini-batch \mathcal{B} .
- ➌ Update with the downhill step (use momentum if desired):

$$\theta \leftarrow \theta - \alpha \nabla \ell_{\theta}(\mathcal{B})$$

- ➍ Go back to step (2).

When steps (2)-(4) cover the entire training set \mathcal{T} we have an [epoch](#).

Like gradient descent, the algorithm proceeds for many epochs.

Stochastic gradient descent

The algorithm is as follows:

- ➊ Initialize θ .
- ➋ Pick a mini-batch \mathcal{B} .
- ➌ Update with the downhill step (use momentum if desired):

$$\theta \leftarrow \theta - \alpha \nabla \ell_{\theta}(\mathcal{B})$$

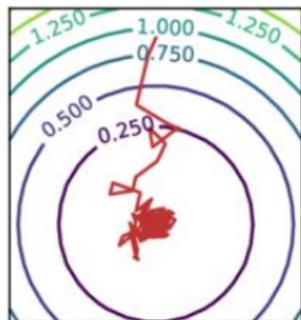
- ➍ Go back to step (2).

When steps (2)-(4) cover the entire training set \mathcal{T} we have an [epoch](#).

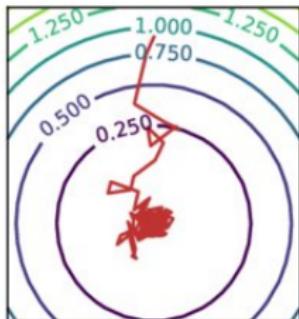
Like gradient descent, the algorithm proceeds for many epochs.

Remark: The update cost is [constant](#) regardless of the size of \mathcal{T} .

Stochastic gradient descent



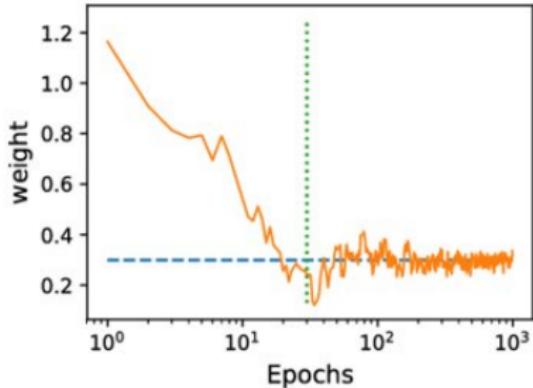
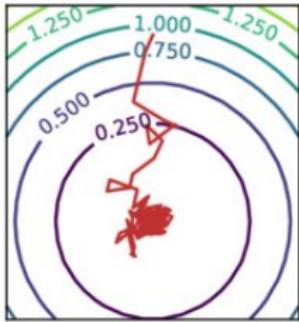
Stochastic gradient descent



SGD does not stop at the minimum.

Oscillations are due to the noise induced by the random sampling.

Stochastic gradient descent



SGD does not stop at the minimum.

Oscillations are due to the noise induced by the random sampling.

Asymptotic upper bounds

For a **convex** problem, there is a true minimizer:

$$f^* = \arg \min_f \ell(f)$$

Asymptotic upper bounds

For a **convex** problem, there is a true minimizer. Consider the inequality:

$$\left| \underbrace{\ell(f_\Theta)}_{\text{GD/SGD}} - \underbrace{\ell(f^*)}_{\text{true}} \right| < \underbrace{\rho}_{\text{accuracy}}$$

Asymptotic upper bounds

For a **convex** problem, there is a true minimizer. Consider the inequality:

$$\left| \underbrace{\ell(f_\Theta)}_{\text{GD/SGD}} - \underbrace{\ell(f^*)}_{\text{true}} \right| < \underbrace{\rho}_{\text{accuracy}}$$

n training examples

d parameters

κ, ν are constants related to the conditioning of the problem

	cost per iteration	iterations to reach ρ
GD	$O(n d)$	$O(\kappa \log \frac{1}{\rho})$
SGD	$O(d)$	$\frac{\nu \kappa^2}{\rho} + o(\frac{1}{\rho})$

Asymptotic upper bounds

For a **convex** problem, there is a true minimizer. Consider the inequality:

$$|\underbrace{\ell(f_\Theta)}_{\text{GD/SGD}} - \underbrace{\ell(f^*)}_{\text{true}}| < \underbrace{\rho}_{\text{accuracy}}$$

n training examples

d parameters

κ, ν are constants related to the conditioning of the problem

	cost per iteration	iterations to reach ρ
GD	$O(n d)$	$O(\kappa \log \frac{1}{\rho})$
SGD	$O(d)$	$\frac{\nu \kappa^2}{\rho} + o(\frac{1}{\rho})$

SGD does not depend on the number of examples,
implying better generalization

Suggested reading

Distill article on why momentum really works:

<https://distill.pub/2017/momentum/>

Seminal paper on using mini-batches for training:

<http://axon.cs.byu.edu/papers/Wilson.nn03.batch.pdf>

Seminal paper on GD vs. SGD performance:

<https://papers.nips.cc/paper/>

3323-the-tradeoffs-of-large-scale-learning.pdf