Mathematics of Deep Learning

Non-convex optimization

Lessons: Kevin Scaman
TDs: Mathieu Even

Class overview

1.	Introduction and general overview	03/01
2.	**Non-convex optimization**	10/01
3.	Structure of ReLU networks and group invariances	17/01
4.	Approximation guarantees	24/01
5.	Stability and robustness	31/01
6.	Infinite width limit of NNs	07/02
7.	Generative models	14/02
8.	Exam	21/02

First-order optimization

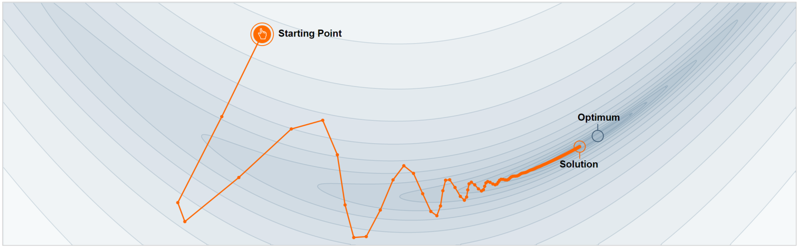
Gradient descent and co.

First-order optimization

- Find a **minimizer** $\theta^\star \in \mathbb{R}^d$ of a given objective function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$,

$$\theta^\star \in \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta)$$

- Using an iterative algorithm relying on the **gradient** $\nabla \mathcal{L}(\theta_t)$ at each iteration $t \geqslant 0$.



source: https://distill.pub/2017/momentum/

First-order optimization

Iterative optimization algorithms

- **Initialization:** $\theta_0 \in \mathbb{R}^d$ (important in practice!).
- **Iteration:** Usually $\theta_{t+1} = \varphi_t\left(\theta_t, \nabla\mathcal{L}(\theta_t), s_t\right)$ where s_t is a hidden variable that is also updated at each iteration.
- **Stopping time:** $T > 0$ (also important in practice!).

First-order optimization

Iterative optimization algorithms

- **Initialization:** $\theta_0 \in \mathbb{R}^d$ (important in practice!).
- **Iteration:** Usually $\theta_{t+1} = \varphi_t\left(\theta_t, \nabla\mathcal{L}(\theta_t), s_t\right)$ where s_t is a hidden variable that is also updated at each iteration.
- **Stopping time:** $T > 0$ (also important in practice!).

Main difficulties in neural network training

First-order optimization

Iterative optimization algorithms

- **Initialization:** $\theta_0 \in \mathbb{R}^d$ (important in practice!).
- **Iteration:** Usually $\theta_{t+1} = \varphi_t\left(\theta_t, \nabla\mathcal{L}(\theta_t), s_t\right)$ where s_t is a hidden variable that is also updated at each iteration.
- **Stopping time:** $T > 0$ (also important in practice!).

Main difficulties in neural network training

- **Non-convexity:** If $\mathcal{L}$ is **convex**, i.e. $\forall\theta, \theta', \mathcal{L}(\frac{\theta+\theta'}{2}) \leqslant \frac{\mathcal{L}(\theta)+\mathcal{L}(\theta')}{2}$, the optimization problem is **simple**. Most theoretical results use this assumption to prove convergence.

First-order optimization

Iterative optimization algorithms

- **Initialization:** $\theta_0 \in \mathbb{R}^d$ (important in practice!).
- **Iteration:** Usually $\theta_{t+1} = \varphi_t\left(\theta_t, \nabla\mathcal{L}(\theta_t), s_t\right)$ where s_t is a hidden variable that is also updated at each iteration.
- **Stopping time:** $T > 0$ (also important in practice!).

Main difficulties in neural network training

- **Non-convexity:** If $\mathcal{L}$ is **convex**, i.e. $\forall\theta, \theta', \mathcal{L}(\frac{\theta+\theta'}{2}) \leqslant \frac{\mathcal{L}(\theta)+\mathcal{L}(\theta')}{2}$, the optimization problem is **simple**. Most theoretical results use this assumption to prove convergence.
- **High dimensionality:** number of parameters $d \gg 1000$.

First-order optimization

Iterative optimization algorithms

- **Initialization:** $\theta_0 \in \mathbb{R}^d$ (important in practice!).
- **Iteration:** Usually $\theta_{t+1} = \varphi_t(\theta_t, \nabla\mathcal{L}(\theta_t), s_t)$ where s_t is a hidden variable that is also updated at each iteration.
- **Stopping time:** $T > 0$ (also important in practice!).

Main difficulties in neural network training

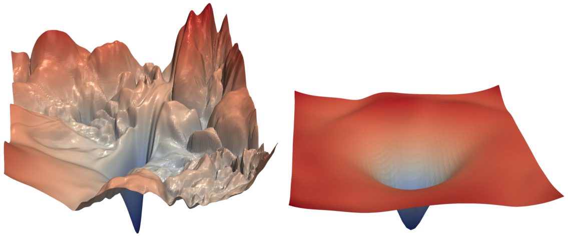
- **Non-convexity:** If $\mathcal{L}$ is **convex**, i.e. $\forall \theta, \theta', \mathcal{L}(\frac{\theta+\theta'}{2}) \leqslant \frac{\mathcal{L}(\theta)+\mathcal{L}(\theta')}{2}$, the optimization problem is **simple**. Most theoretical results use this assumption to prove convergence.
- **High dimensionality:** number of parameters $d \gg 1000$.
- **Access to the gradient:** the gradient of $\mathcal{L}$ is too expensive to compute! In practice, $\nabla\mathcal{L}(\theta_t)$ is replaced by a **stochastic** or **mini-batch** approximation $\tilde{\nabla}_t$.

Loss landscape

Training a neural network requires solving a difficult non-convex optimization problem

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \ell\left(g_\theta(x_i), y_i\right)$$

Ex: loss landscape around the optimum for ResNet-56 trained on CIFAR10.


(a) without skip connections (b) with skip connections

source: Visualizing the Loss Landscape of Neural Nets. Li et.al., 2018.

Types of irregularities

- Non-convexity,

Types of irregularities

- Non-convexity,
- Multiple local minima,

Types of irregularities

- Non-convexity,
- Multiple local minima,
- Spurious stationary points (e.g. saddle points),

Types of irregularities

- Non-convexity,
- Multiple local minima,
- Spurious stationary points (e.g. saddle points),
- Sharp variations (high curvature),

Types of irregularities

- Non-convexity,
- Multiple local minima,
- Spurious stationary points (e.g. saddle points),
- Sharp variations (high curvature),
- Local explosion (large values),

Types of irregularities

- Non-convexity,
- Multiple local minima,
- Spurious stationary points (e.g. saddle points),
- Sharp variations (high curvature),
- Local explosion (large values),
- Plateaux (flat regions),

Types of irregularities

- Non-convexity,
- Multiple local minima,
- Spurious stationary points (e.g. saddle points),
- Sharp variations (high curvature),
- Local explosion (large values),
- Plateaux (flat regions),
- ...

⚠ In general, the regularity of the objective will depend on the architecture of the neural network, and part of DL research is devoted to finding architecture that are easy to train.

Ideal optimization theory for DL training

- Should provide **fast gradient computation** for composition of modules.
- Should explain performances of **non-convex SGD** (and its variants).
- Should work in **high-dimensional** spaces.
- Should extend to **non-smooth** objectives.
- Should have assumptions that are **reasonable for neural networks**.

Next steps

1. Understand how the **gradient is computed** in Pytorch.
2. Understand why **stochastic gradient works**.

Some warnings about optimization in deep learning

Our final goal is to reduce the **population risk**, i.e. $\mathbb{E}(\ell(g_\theta(X), Y))$!

- We need to pay attention to **overfitting** in addition to using the optimization algorithm to reduce the training error.
- In this class, we focus specifically on the **performance** of the optimization algorithm in minimizing the objective function, rather than the model's generalization error.
- In the next lessons, we will see techniques to avoid **overfitting**.

Automatic differentiation

A short recap on differentiating composite functions

Existing approaches to compute gradients

- **Finite differences:** small perturbations $g'(x) \approx \frac{g(x+\varepsilon)-g(x)}{\varepsilon}$. Leads to **round-off** errors.

Existing approaches to compute gradients

- **Finite differences:** small perturbations $g'(x) \approx \frac{g(x+\varepsilon)-g(x)}{\varepsilon}$. Leads to **round-off** errors.
- **Symbolic differentiation:** keeps **symbolic expressions** at each step of the process.

Existing approaches to compute gradients

- **Finite differences:** small perturbations $g'(x) \approx \frac{g(x+\varepsilon)-g(x)}{\varepsilon}$. Leads to **round-off** errors.
- **Symbolic differentiation:** keeps **symbolic expressions** at each step of the process.
- **Automatic differentiation:** clever use of the **chain rule**.

Existing approaches to compute gradients

- **Finite differences:** small perturbations $g'(x) \approx \frac{g(x+\varepsilon)-g(x)}{\varepsilon}$. Leads to **round-off** errors.
- **Symbolic differentiation:** keeps **symbolic expressions** at each step of the process.
- **Automatic differentiation:** clever use of the **chain rule**.

Chain rule (simple version)

Let $f, g : \mathbb{R} \to \mathbb{R}$ differentiable, then

$$(f \circ g)' = (f' \circ g) \cdot g'$$

Recap: derivatives of multi-dimensional functions

Definition (Jacobian matrix)

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ a differentiable function. Its Jacobian $J_f(x) \in \mathbb{R}^{m \times n}$ is the matrix whose coordinates are the partial derivatives:

$$J_f(x) = \begin{bmatrix} \nabla f_1(x)^\top \\ \cdots \\ \nabla f_m(x)^\top \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}$$

Recap: derivatives of multi-dimensional functions

Definition (Jacobian matrix)

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ a differentiable function. Its Jacobian $J_f(x) \in \mathbb{R}^{m\times n}$ is the matrix whose coordinates are the partial derivatives:

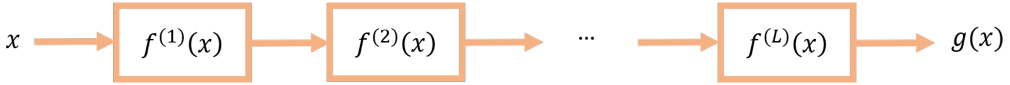
$$J_f(x) = \begin{bmatrix} \nabla f_1(x)^\top \\ \cdots \\ \nabla f_m(x)^\top \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}$$

Chain rule (multi-dimensional version)

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^p \to \mathbb{R}^n$ differentiable, then

$$J_{f\circ g} = (J_f \circ g) \times J_g$$

Derivative of a composition of functions

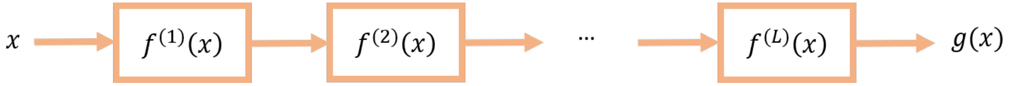


Composite function

- Let $f^{(l)} : \mathbb{R}^{d^{(l-1)}} \to \mathbb{R}^{d^{(l)}}$ and $g(x) = g^{(L)}(x)$ where

$$g^{(l)}(x) = f^{(l)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x)$$

Derivative of a composition of functions



Composite function

- Let $f^{(l)} : \mathbb{R}^{d(l-1)} \to \mathbb{R}^{d(l)}$ and $g(x) = g^{(L)}(x)$ where

$$g^{(l)}(x) = f^{(l)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x)$$

- Then, the Jacobian matrix (i.e. matrix of derivatives) of g is

$$J_g(x) = J_{f^{(L)}}\left(g^{(L-1)}(x)\right) \times \cdots \times J_{f^{(2)}}\left(g^{(1)}(x)\right) \times J_{f^{(1)}}(x)$$

Derivative of a composition of functions

$$\begin{tikzcd} x \arrow[r] & \boxed{f^{(1)}(x)} \arrow[r] & \boxed{f^{(2)}(x)} \arrow[r] & \cdots \arrow[r] & \boxed{f^{(L)}(x)} \arrow[r] & g(x) \end{tikzcd}$$

Composite function

- Let $f^{(l)} : \mathbb{R}^{d(l-1)} \to \mathbb{R}^{d(l)}$ and $g(x) = g^{(L)}(x)$ where

$$g^{(l)}(x) = f^{(l)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x)$$

- Then, the Jacobian matrix (i.e. matrix of derivatives) of g is

$$J_g(x) = J_{f^{(L)}}\left(g^{(L-1)}(x)\right) \times \cdots \times J_{f^{(2)}}\left(g^{(1)}(x)\right) \times J_{f^{(1)}}(x)$$

- What is the **computational complexity** to compute the Jacobian matrix?

Computational complexity

Finite differences

- The gradient of g can be approximated by **finite differences**: $\nabla g(x)_i \approx \frac{g(x+\varepsilon e_i)-g(x)}{\varepsilon}$
- **Computational complexity:** proportional to **input dimension**.

Computational complexity

Finite differences

- The gradient of g can be approximated by **finite differences**: $\nabla g(x)_i \approx \frac{g(x+\varepsilon e_i)-g(x)}{\varepsilon}$
- **Computational complexity:** proportional to **input dimension**.

Matrix product

- We have $\nabla g(x)^\top = J_L \times \cdots \times J_2 \times J_1$ where $J_l = J_{f^{(l)}}\left(g^{(l-1)}(x)\right)$.

Computational complexity

Finite differences

- The gradient of g can be approximated by **finite differences**: $\nabla g(x)_i \approx \frac{g(x+\varepsilon e_i)-g(x)}{\varepsilon}$
- **Computational complexity:** proportional to **input dimension**.

Matrix product

- We have $\nabla g(x)^\top = J_L \times \cdots \times J_2 \times J_1$ where $J_l = J_{f^{(l)}}\left(g^{(l-1)}(x)\right)$.
- There are $(L-1)!$ ways to compute this product of L matrices.

Computational complexity

Finite differences

- The gradient of g can be approximated by **finite differences**: $\nabla g(x)_i \approx \frac{g(x+\varepsilon e_i)-g(x)}{\varepsilon}$
- **Computational complexity:** proportional to **input dimension**.

Matrix product

- We have $\nabla g(x)^\top = J_L \times \cdots \times J_2 \times J_1$ where $J_l = J_{f^{(l)}}\left(g^{(l-1)}(x)\right)$.
- There are $(L-1)!$ ways to compute this product of L matrices.
- **Forward propagation:** Compute $\nabla g(x)^\top = (J_L \times (J_{L-1} \times \cdots \times (J_2 \times J_1)))$. Requires computation intensive **matrix-matrix products**.

Computational complexity

Finite differences

- The gradient of g can be approximated by **finite differences**: $\nabla g(x)_i \approx \frac{g(x+\varepsilon e_i)-g(x)}{\varepsilon}$
- **Computational complexity:** proportional to **input dimension**.

Matrix product

- We have $\nabla g(x)^\top = J_L \times \cdots \times J_2 \times J_1$ where $J_l = J_{f^{(l)}}\left(g^{(l-1)}(x)\right)$.
- There are $(L-1)!$ ways to compute this product of L matrices.
- **Forward propagation:** Compute $\nabla g(x)^\top = (J_L \times (J_{L-1} \times \cdots \times (J_2 \times J_1)))$. Requires computation intensive **matrix-matrix products**.
- **Backward propagation:** Compute $\nabla g(x)^\top = (((J_L \times J_{L-1}) \times \cdots \times J_2) \times J_1)$. If output is 1-dimensional, only needs **matrix-vector products**!

Which algorithm is faster?

Complexity for gradients of MLPs

- Let $g_\theta : \mathbb{R}^d \to \mathbb{R}$ an MLP of width $w \geqslant d$ and depth $L \geqslant 1$.
- **Function value:**
- **Finite differences:**
- **Forward propagation:**
- **Backward propagation:**

Which algorithm is faster?

Complexity for gradients of MLPs

- Let $g_\theta : \mathbb{R}^d \to \mathbb{R}$ an MLP of width $w \geqslant d$ and depth $L \geqslant 1$.
- **Function value:** $O(w^2 L)$ operations.
- **Finite differences:**
- **Forward propagation:**
- **Backward propagation:**

Which algorithm is faster?

Complexity for gradients of MLPs

- Let $g_\theta : \mathbb{R}^d \to \mathbb{R}$ an MLP of width $w \geqslant d$ and depth $L \geqslant 1$.
- **Function value:** $O(w^2 L)$ operations.
- **Finite differences:** $O(dw^2 L)$ operations.
- **Forward propagation:**
- **Backward propagation:**

Which algorithm is faster?

Complexity for gradients of MLPs

- Let $g_\theta : \mathbb{R}^d \to \mathbb{R}$ an MLP of width $w \geqslant d$ and depth $L \geqslant 1$.
- **Function value:** $O(w^2 L)$ operations.
- **Finite differences:** $O(dw^2 L)$ operations.
- **Forward propagation:** $O(dw^2 L)$ operations.
- **Backward propagation:**

Which algorithm is faster?

Complexity for gradients of MLPs

- Let $g_\theta : \mathbb{R}^d \to \mathbb{R}$ an MLP of width $w \geqslant d$ and depth $L \geqslant 1$.
- **Function value:** $O(w^2 L)$ operations.
- **Finite differences:** $O(dw^2 L)$ operations.
- **Forward propagation:** $O(dw^2 L)$ operations.
- **Backward propagation:** $O(w^2 L)$ operations.

Which algorithm is faster?

Complexity for gradients of MLPs

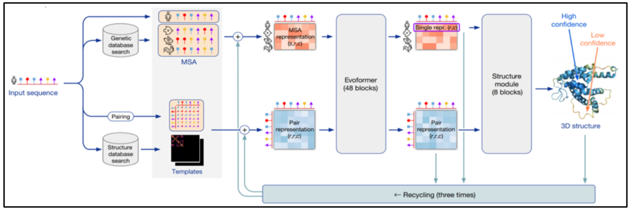
- Let $g_\theta : \mathbb{R}^d \to \mathbb{R}$ an MLP of width $w \geqslant d$ and depth $L \geqslant 1$.
- **Function value:** $O(w^2 L)$ operations.
- **Finite differences:** $O(dw^2 L)$ operations.
- **Forward propagation:** $O(dw^2 L)$ operations.
- **Backward propagation:** $O(w^2 L)$ operations.

Intuition for gradients w.r.t. parameters

- Finite differences requires **two function calls per parameter**.
- Backprop requires **O(1) function calls for the whole gradient**.
- Interpretation as parameter testing:
  - Each partial derivative w.r.t. a parameter indicates if this parameter can describe the data.
  - With backprop, we can test **all parameters at once**.

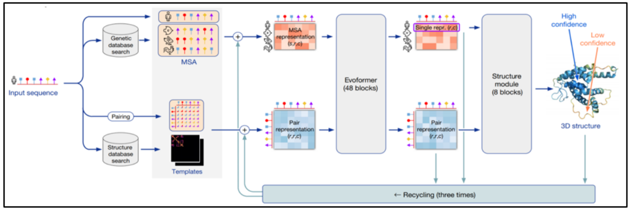
Computation graphs: intuition

Complex neural network architecture (e.g. AlphaFold)

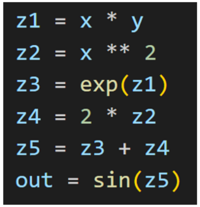

Input sequence
Genetic database search
MSA
Pairing
Structure database search
Templates
MSA representation (s,r,c)
Pair representation (r,r,c)
Evoformer (48 blocks)
Single repr. (r,c)
Pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (three times)

Computation graphs: intuition

Complex neural network architecture (e.g. AlphaFold)



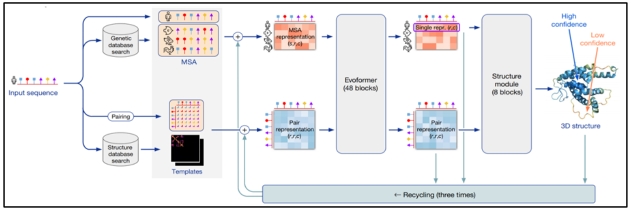
Code (e.g. Python)



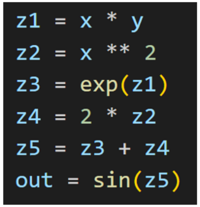
```
z1 = x * y
z2 = x ** 2
z3 = exp(z1)
z4 = 2 * z2
z5 = z3 + z4
out = sin(z5)
```

Computation graphs: intuition

Complex neural network architecture (e.g. AlphaFold)

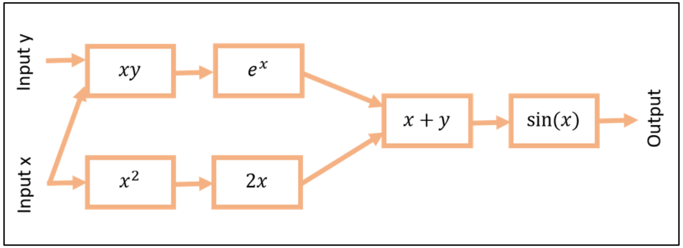

Input sequence
Genetic database search
MSA
Pairing
Structure database search
Templates
MSA representation (s,r,c)
Pair representation (r,r,c)
Evoformer (48 blocks)
Single repr. (r,c)
Pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (three times)

Code (e.g. Python)



```
z1 = x * y
z2 = x ** 2
z3 = exp(z1)
z4 = 2 * z2
z5 = z3 + z4
out = sin(z5)
```

Computation graph (DAG of mathematical operations)



Computation graphs: formal definition

Definition (computation graph)

- Let $G = (V, E)$ be a *directed acyclic graph* (DAG) encoding a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.

Computation graphs: formal definition

Definition (computation graph)

- Let $G = (V, E)$ be a *directed acyclic graph* (DAG) encoding a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.
- **Parameters:** For any root $r \in R$, let $x^{(r)} = \theta^{(r)}$ be an input or parameter.

Computation graphs: formal definition

Definition (computation graph)

- Let $G = (V, E)$ be a *directed acyclic graph* (DAG) encoding a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.
- **Parameters:** For any root $r \in R$, let $x^{(r)} = \theta^{(r)}$ be an input or parameter.
- **Layers:** For any other node $v \in V/R$, let $x^{(v)} = f^{(v)}\left((x^{(w)})_{w \in \text{Parents}(v)}\right)$.

Computation graphs: formal definition

Definition (computation graph)

- Let $G = (V, E)$ be a *directed acyclic graph* (DAG) encoding a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.
- **Parameters:** For any root $r \in R$, let $x^{(r)} = \theta^{(r)}$ be an input or parameter.
- **Layers:** For any other node $v \in V/R$, let $x^{(v)} = f^{(v)}\left((x^{(w)})_{w \in \text{Parents}(v)}\right)$.
- **Output:** The output of the leaf node $x^{(f)} = \mathcal{L}(\theta) \in \mathbb{R}$ where $\theta = (\theta^{(r)})_{r \in R}$.

Computation graphs: formal definition

Definition (computation graph)

- Let $G = (V, E)$ be a *directed acyclic graph* (DAG) encoding a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.
- **Parameters:** For any root $r \in R$, let $x^{(r)} = \theta^{(r)}$ be an input or parameter.
- **Layers:** For any other node $v \in V/R$, let $x^{(v)} = f^{(v)}\left((x^{(w)})_{w \in \mathsf{Parents}(v)}\right)$.
- **Output:** The output of the leaf node $x^{(f)} = \mathcal{L}(\theta) \in \mathbb{R}$ where $\theta = (\theta^{(r)})_{r \in R}$.

Properties

- Essentially **all programmable functions** can be decomposed this way.

Computation graphs: formal definition

Definition (computation graph)

- Let $G = (V, E)$ be a *directed acyclic graph* (DAG) encoding a function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.
- **Parameters:** For any root $r \in R$, let $x^{(r)} = \theta^{(r)}$ be an input or parameter.
- **Layers:** For any other node $v \in V/R$, let $x^{(v)} = f^{(v)}\left((x^{(w)})_{w \in \text{Parents}(v)}\right)$.
- **Output:** The output of the leaf node $x^{(f)} = \mathcal{L}(\theta) \in \mathbb{R}$ where $\theta = (\theta^{(r)})_{r \in R}$.

Properties

- Essentially **all programmable functions** can be decomposed this way.
- **Chain rule:** partial gradient $\frac{\partial x^{(f)}}{\partial x^{(v)}}$ for a node $v \in V$ from that of its children.

$$\frac{\partial x^{(f)}}{\partial x^{(v)}} = \sum_{w \in \text{Children}(v)} \frac{\partial f^{(w)}\left((x^{(w')})_{w' \in \text{Parents}(w)}\right)}{\partial x^{(v)}}^{\top} \frac{\partial x^{(f)}}{\partial x^{(w)}}$$

The backpropagation algorithm (Rumelhart et al., 1986)

- Composed of 2 steps: a **forward pass** (FP) and a **backward pass** (BP).

The backpropagation algorithm (Rumelhart et al., 1986)

- Composed of 2 steps: a **forward pass** (FP) and a **backward pass** (BP).
- **FP:** For all $r \in R$, let $y^{(r)} = x_r$ the inputs (or parameters), and, for all $v \in V/R$, we compute iteratively **from roots to leaf**,

$$y^{(v)} = f^{(v)}\left(\left(y^{(w)}\right)_{w \in \text{Parents}(v)}\right)$$

The backpropagation algorithm (Rumelhart et al., 1986)

- Composed of 2 steps: a **forward pass** (FP) and a **backward pass** (BP).
- **FP:** For all $r \in R$, let $y^{(r)} = x_r$ the inputs (or parameters), and, for all $v \in V/R$, we compute iteratively **from roots to leaf**,

$$y^{(v)} = f^{(v)}\left((y^{(w)})_{w \in \text{Parents}(v)}\right)$$

- **BP:** Let $z^{(f)} = 1$ and, for $v \in V/F$, we compute iteratively **from leaf to roots**,

$$z^{(v)} = \sum_{w \in \text{Children}(v)} \frac{\partial f^{(w)}\left((y^{(w')})_{w' \in \text{Parents}(w)}\right)}{\partial x^{(v)}}^{\top} z^{(w)}$$

- Then, for all $r \in R$, we have $\frac{\partial \mathcal{L}(\theta)}{\partial \theta^{(r)}} = z^{(r)}$.

Non-convex optimization

Convergence to local/global minima

Optimizing non-convex functions is hard...

Assumptions

- The objective function is **non-convex**, **differentiable** and β-**smooth**, i.e. $\forall \theta, \theta' \in \mathbb{R}^d$,

$$\|\nabla \mathcal{L}(\theta) - \nabla \mathcal{L}(\theta')\|_2 \leqslant \beta \|\theta - \theta'\|_2$$

- We access unbiased noisy gradients $\tilde{\nabla}_t$ where $\mathbb{E}(\tilde{\nabla}_t) = \nabla \mathcal{L}(\theta_t)$ and $\mathrm{var}(\tilde{\nabla}_t) \leqslant \sigma^2$.

Optimizing non-convex functions is hard...

Assumptions

- The objective function is **non-convex**, **differentiable** and β-**smooth**, i.e. $\forall \theta, \theta' \in \mathbb{R}^d$,

$$\|\nabla \mathcal{L}(\theta) - \nabla \mathcal{L}(\theta')\|_2 \leqslant \beta \|\theta - \theta'\|_2$$

- We access unbiased noisy gradients $\tilde{\nabla}_t$ where $\mathbb{E}(\tilde{\nabla}_t) = \nabla \mathcal{L}(\theta_t)$ and $\mathrm{var}(\tilde{\nabla}_t) \leqslant \sigma^2$.

Proposition (worst-case convergence to global optimum)

For any first-order algorithm, there exists a smooth function $\mathcal{L}$ such that approx. error is at least

$$\mathcal{L}(\theta_t) - \mathcal{L}(\theta^\star) = \Omega(t^{-2/d})$$

This is prohibitive for large dimensional spaces (i.e. $d \geqslant 100$)!

Convergence of SGD… to a stationary point

Theorem (convergence of non-convex SGD)

Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ be a smooth function and $\Delta = \mathcal{L}(\theta_0) - \mathcal{L}(\theta^\star)$. Then, SGD with step-size $\eta \leqslant \frac{1}{\beta}$ achieves the error

$$\mathbb{E}\Big[\min_{t \leqslant T} \|\nabla \mathcal{L}(\theta_t)\|^2\Big] \leqslant \frac{2\Delta}{\eta T} + \beta\eta\sigma^2$$

Convergence of SGD... to a stationary point

Theorem (convergence of non-convex SGD)

Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ be a smooth function and $\Delta = \mathcal{L}(\theta_0) - \mathcal{L}(\theta^\star)$. Then, SGD with step-size $\eta \leqslant \frac{1}{\beta}$ achieves the error

$$\mathbb{E}\big[\min_{t \leqslant T} \|\nabla \mathcal{L}(\theta_t)\|^2\big] \leqslant \frac{2\Delta}{\eta T} + \beta\eta\sigma^2$$

- Convergence in expectation implies cv. with **high probability** using Markov inequality.
- Convergence of the **best iterate** (i.e. smallest gradient norm). :(

Convergence of SGD... to a stationary point

Theorem (convergence of non-convex SGD)

Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ be a smooth function and $\Delta = \mathcal{L}(\theta_0) - \mathcal{L}(\theta^\star)$. Then, SGD with step-size $\eta \leqslant \frac{1}{\beta}$ achieves the error

$$\mathbb{E}\Big[\min_{t \leqslant T} \|\nabla \mathcal{L}(\theta_t)\|^2\Big] \leqslant \frac{2\Delta}{\eta T} + \beta\eta\sigma^2$$

- Convergence in expectation implies cv. with **high probability** using Markov inequality.
- Convergence of the **best iterate** (i.e. smallest gradient norm). :(
- Without noise, a **constant step-size** $\eta = 1/\beta$ is optimal.

Convergence of SGD… to a stationary point

Theorem (convergence of non-convex SGD)

Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ be a smooth function and $\Delta = \mathcal{L}(\theta_0) - \mathcal{L}(\theta^\star)$. Then, SGD with step-size $\eta \leqslant \frac{1}{\beta}$ achieves the error

$$\mathbb{E}\big[\min_{t \leqslant T} \|\nabla \mathcal{L}(\theta_t)\|^2\big] \leqslant \frac{2\Delta}{\eta T} + \beta \eta \sigma^2$$

- Convergence in expectation implies cv. with **high probability** using Markov inequality.
- Convergence of the **best iterate** (i.e. smallest gradient norm). :(
- Without noise, a **constant step-size** $\eta = 1/\beta$ is optimal.
- Gradient noise adds a constant term. If constant step-size, **no convergence**.

Convergence of SGD... to a stationary point

Theorem (convergence of non-convex SGD)

Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ be a smooth function and $\Delta = \mathcal{L}(\theta_0) - \mathcal{L}(\theta^\star)$. Then, SGD with step-size $\eta \leqslant \frac{1}{\beta}$ achieves the error

$$\mathbb{E}\big[\min_{t \leqslant T} \|\nabla \mathcal{L}(\theta_t)\|^2\big] \leqslant \frac{2\Delta}{\eta T} + \beta\eta\sigma^2$$

- Convergence in expectation implies cv. with **high probability** using Markov inequality.
- Convergence of the **best iterate** (i.e. smallest gradient norm). :(
- Without noise, a **constant step-size** $\eta = 1/\beta$ is optimal.
- Gradient noise adds a constant term. If constant step-size, **no convergence**.
- Convergence only possible for **decreasing step-sizes**, with optimal cv. in $O(1/\sqrt{T})$.

Convergence to a local minimum

How to obtain local minimum?

- A local minimum can be defined using second order derivatives:
  1. **Stationarity:** $\nabla\mathcal{L}(\theta) = 0$
  2. **Convexity:** the Hessian $H_{\mathcal{L}}(x)$ is SDP.

Convergence to a local minimum

How to obtain local minimum?

- A local minimum can be defined using second order derivatives:
  1. **Stationarity:** $\nabla\mathcal{L}(\theta) = 0$
  2. **Convexity:** the Hessian $H_{\mathcal{L}}(x)$ is SDP.

Convergence to a local minimum (Jin et.al., 2017)

- Adding a small noise allows the parameter to escape saddle points.
- Additional assumption: the Hessian $H_{\mathcal{L}}$ is ρ-Lipschitz w.r.t. spectral norm.
- With probability at least $1-\delta$, the number of iterations to reach a gradient norm $\|\nabla\mathcal{L}(\theta_t)\| \leqslant \varepsilon$ and near-convexity $\lambda_1(H_{\mathcal{L}}(\theta_t)) \geqslant -\sqrt{\rho\varepsilon}$ is bounded by

$$O\left(\frac{\beta\Delta}{\varepsilon^2}\log\left(\frac{d\beta\Delta}{\varepsilon\delta}\right)^4\right)$$

Recap

- The loss landscape of DL training is **non-convex** and potentially difficult to optimize.
- Convergence to a **global minimum prohibitive** in high-dimensional spaces.
- GD converges to a **stationary point** with constant step-sizes.
- SGD converges (more slowly) to a **stationary point** with decreasing step-sizes.
- Adding noise is necessary to converge to a **local minimum** (Jin et.al., 2017).

Recap

- The loss landscape of DL training is **non-convex** and potentially difficult to optimize.
- Convergence to a **global minimum prohibitive** in high-dimensional spaces.
- GD converges to a **stationary point** with constant step-sizes.
- SGD converges (more slowly) to a **stationary point** with decreasing step-sizes.
- Adding noise is necessary to converge to a **local minimum** (Jin et.al., 2017).
- We need **stronger assumptions** on the objective function to go beyond...

Beyond local minimisation

The Łojasiewicz condition

A look at the proof of convergence of SGD

- By smoothness, we have, for $\theta_{t+1} = \theta_t - \eta G_t$,

$$\mathbb{E}(\mathcal{L}(\theta_{t+1})) - \mathbb{E}(\mathcal{L}(\theta_t)) \leqslant -\eta \left(1 - \frac{\beta\eta}{2}\right) \mathbb{E}(\|\nabla\mathcal{L}(\theta_t)\|^2) + \frac{\beta\eta^2\sigma^2}{2}$$

A look at the proof of convergence of SGD

- By smoothness, we have, for $\theta_{t+1} = \theta_t - \eta G_t$,

$$\mathbb{E}(\mathcal{L}(\theta_{t+1})) - \mathbb{E}(\mathcal{L}(\theta_t)) \leqslant -\eta \left(1 - \frac{\beta\eta}{2}\right) \mathbb{E}(\|\nabla\mathcal{L}(\theta_t)\|^2) + \frac{\beta\eta^2\sigma^2}{2}$$

- If the gradient is large, then the gradient step improves the function value.

A look at the proof of convergence of SGD

- By smoothness, we have, for $\theta_{t+1} = \theta_t - \eta G_t$,

$$\mathbb{E}(\mathcal{L}(\theta_{t+1})) - \mathbb{E}(\mathcal{L}(\theta_t)) \leqslant -\eta \left(1 - \frac{\beta\eta}{2}\right) \mathbb{E}(\|\nabla\mathcal{L}(\theta_t)\|^2) + \frac{\beta\eta^2\sigma^2}{2}$$

- If the gradient is large, then the gradient step improves the function value.
- When $\mathcal{L}$ is α-strongly convex, we have $\|\nabla\mathcal{L}(\theta_t)\|^2 \geqslant 2\alpha(\mathcal{L}(\theta_t) - \mathcal{L}(\theta^\star))$.

A look at the proof of convergence of SGD

- By smoothness, we have, for $\theta_{t+1} = \theta_t - \eta G_t$,

$$\mathbb{E}(\mathcal{L}(\theta_{t+1})) - \mathbb{E}(\mathcal{L}(\theta_t)) \leqslant -\eta \left(1 - \frac{\beta\eta}{2}\right) \mathbb{E}(\|\nabla \mathcal{L}(\theta_t)\|^2) + \frac{\beta\eta^2\sigma^2}{2}$$

- If the gradient is large, then the gradient step improves the function value.
- When $\mathcal{L}$ is α-strongly convex, we have $\|\nabla \mathcal{L}(\theta_t)\|^2 \geqslant 2\alpha(\mathcal{L}(\theta_t) - \mathcal{L}(\theta^\star))$.
- If $\eta \leqslant 1/\beta$, this implies, for $\varepsilon_t = \mathbb{E}(\mathcal{L}(\theta_t)) - \mathbb{E}(\mathcal{L}(\theta^\star))$,

$$\varepsilon_{t+1} \leqslant (1 - \alpha\eta)\,\varepsilon_t + \frac{\beta\eta^2\sigma^2}{2}$$

The Polyak-Łojasiewicz condition

Definition (Polyak & Łojasiewicz, 1963)

A function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is said to verify the μ-Polyak-Łojasiewicz (PL) condition iff

$$\|\nabla \mathcal{L}(\theta_t)\|^2 \geqslant 2\mu \left(\mathcal{L}(\theta_t) - \mathcal{L}(\theta^\star)\right)$$

where $\theta^\star \in \mathbb{R}^d$ is a global minimum of the function $\mathcal{L}$ and $\mu > 0$ is a constant.

The Polyak-Łojasiewicz condition

Definition (Polyak & Łojasiewicz, 1963)

A function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is said to verify the μ-Polyak-Łojasiewicz (PL) condition iff

$$\|\nabla \mathcal{L}(\theta_t)\|^2 \geqslant 2\mu \left(\mathcal{L}(\theta_t) - \mathcal{L}(\theta^\star)\right)$$

where $\theta^\star \in \mathbb{R}^d$ is a global minimum of the function $\mathcal{L}$ and $\mu > 0$ is a constant.

Theorem (convergence of SGD under μ-PL)

If $\mathcal{L}$ is β-smooth and verifies the PL condition, then, with $\eta \leqslant \frac{1}{\beta}$, SGD achieves the precision

$$\mathbb{E}(\mathcal{L}(\theta_T) - \mathcal{L}(\theta^\star)) \leqslant \Delta \left(1 - \mu\eta\right)^T + \frac{\beta\eta\sigma^2}{2\mu}$$

Exponential convergence rate $O(e^{-T})$ without noise, and $O(\ln(T)/T)$ otherwise.

Beyond strongly convex functions

Is the PL condition satisfied for more than strongly-convex functions?

Examples

- For $\mathcal{L}(\theta) = (\theta_1 - \cos(\theta_2))^2$, we have $\|\nabla \mathcal{L}(\theta)\|^2 =$

Beyond strongly convex functions

Is the PL condition satisfied for more than strongly-convex functions?

Examples

- For $\mathcal{L}(\theta) = (\theta_1 - \cos(\theta_2))^2$, we have $\|\nabla \mathcal{L}(\theta)\|^2 = 4\mathcal{L}(\theta)(1 + \sin(\theta_2)^2) \geqslant 4\mathcal{L}(\theta)$.

Beyond strongly convex functions

Is the PL condition satisfied for more than strongly-convex functions?

Examples

- For $\mathcal{L}(\theta) = (\theta_1 - \cos(\theta_2))^2$, we have $\|\nabla\mathcal{L}(\theta)\|^2 = 4\mathcal{L}(\theta)(1 + \sin(\theta_2)^2) \geqslant 4\mathcal{L}(\theta)$.
- More gl. if $\mathcal{L}(\theta) = g(\theta)^2$ and $\|\nabla g(\theta)\| \geqslant c$ for any $\theta \in \mathbb{R}^d$, then $\|\nabla\mathcal{L}(\theta)\|^2 \geqslant 4c^2\mathcal{L}(\theta)$.

Beyond strongly convex functions

Is the PL condition satisfied for more than strongly-convex functions?

Examples

- For $\mathcal{L}(\theta) = (\theta_1 - \cos(\theta_2))^2$, we have $\|\nabla \mathcal{L}(\theta)\|^2 = 4\mathcal{L}(\theta)(1 + \sin(\theta_2)^2) \geqslant 4\mathcal{L}(\theta)$.
- More gl. if $\mathcal{L}(\theta) = g(\theta)^2$ and $\|\nabla g(\theta)\| \geqslant c$ for any $\theta \in \mathbb{R}^d$, then $\|\nabla \mathcal{L}(\theta)\|^2 \geqslant 4c^2\mathcal{L}(\theta)$.

Theorem (PL condition for compositions)

Let $\mathcal{L}(\theta) = (f \circ g)(\theta)$ where f satisfies the μ-PL condition and g is such that, $\forall \theta \in \mathbb{R}^d$

$$\sigma_{\min}\left(J_g(\theta)^\top\right) \geqslant \varepsilon\,,$$

where $\sigma_{\min}(M) = \min_{x \neq 0} \|Mx\|/\|x\|$ is the smallest singular value of the matrix M. Then $\mathcal{L}$ verifies the μ'-PL condition with $\mu' = \mu\varepsilon^2$.

PL for neural networks

Theorem (PL condition for MSE loss)

Let $\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(g_\theta(x_i), y_i)$ where $\ell(y, y') = \|y - y'\|_2^2$ and the model g_θ is such that

$$\sigma_{\min}\left(\left(J_{g,\theta}(x_1, \theta)^\top \;\middle|\; \cdots \;\middle|\; J_{g,\theta}(x_N, \theta)^\top\right)\right) \geqslant \varepsilon$$

then $\mathcal{L}$ verifies the μ-PL condition with $\mu = 4\varepsilon^2/N$.

PL for neural networks

Theorem (PL condition for MSE loss)

Let $\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(g_\theta(x_i), y_i)$ where $\ell(y, y') = \|y - y'\|_2^2$ and the model g_θ is such that

$$\sigma_{\min}\left(\left(J_{g,\theta}(x_1, \theta)^\top \;\middle|\; \cdots \;\middle|\; J_{g,\theta}(x_N, \theta)^\top\right)\right) \geqslant \varepsilon$$

then $\mathcal{L}$ verifies the μ-PL condition with $\mu = 4\varepsilon^2/N$.

- For **over-parameterized neural networks**, this quantity is usually controlled for $\theta = \theta_0$ (if the weights are **properly initialized**, see lesson 5), and valid on a neighborhood around initialization (linked with the **Neural Tangent Kernel**, see lesson 6). For example, **uniform conditioning** (Liu et al., 2020) assumes that the singular value is lower bounded for all $\theta \in \mathcal{B}(\theta_0, R)$.

Beyond smooth minimisation

Smoothing and noise

Smoothness of the objective

Is the objective function really smooth?

Issues

1. Smoothness usually breaks as θ tends to infinity (e.g. $\theta \mapsto \theta^3$ or 3-layer MLPs).
2. MLPs are non-smooth as soon as the activation function is not differentiable (e.g. ReLU networks).

Smoothness of the objective

Is the objective function really smooth?

Issues

1. Smoothness usually breaks as θ tends to infinity (e.g. $\theta \mapsto \theta^3$ or 3-layer MLPs).
2. MLPs are non-smooth as soon as the activation function is not differentiable (e.g. ReLU networks).

Solutions

1. PL also provides convergence with local smoothness around initialization.
2. If the model is not locally smooth/differentiable, two solutions:
   - Extend the notion of derivative to Lipschitz functions (Clarke differential).
   - Approximate the objective function with a smooth function.

Randomized smoothing

Definition (Duchi et.al., 2011)

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a function and $\gamma > 0$. Then, let $f_\gamma : \mathbb{R}^d \to \mathbb{R}$ be defined as

$$f^\gamma(\theta) = \mathbb{E}(f(\theta + \gamma X))$$

where $X \sim \mathcal{N}(0, I_d)$ is a Gaussian random variable.

Randomized smoothing

Definition (Duchi et.al., 2011)

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a function and $\gamma > 0$. Then, let $f_\gamma : \mathbb{R}^d \to \mathbb{R}$ be defined as

$$f^\gamma(\theta) = \mathbb{E}(f(\theta + \gamma X))$$

where $X \sim \mathcal{N}(0, I_d)$ is a Gaussian random variable.

Theorem

If f is L-Lipschitz, then f^γ is L/γ-smooth and $f(\theta) \leqslant f^\gamma(\theta) \leqslant f(\theta) + \gamma L\sqrt{d}$.

- Randomized smoothing transforms a **Lipschitz function** into a **smooth function**!
- We can then apply SGD and use previous convergence results.

Randomized smoothing

Approximation of the smooth gradient

- The gradient of the smooth function is $\nabla f^\gamma(\theta) = \mathbb{E}(\nabla f(\theta + \gamma X))$.
- Can be approximated by $\widehat{\nabla} f(\theta) = \frac{1}{K}\sum_{k\in[\![1,K]\!]} \nabla f(\theta + \gamma X_k)$ where $X_k \sim \mathcal{N}(0, I_d)$ are i.i.d. Gaussian r.v.
- Adds a gradient noise of variance

$$\sigma^2 = \frac{\mathrm{var}\left(\nabla f(\theta + \gamma X)\right)}{K} \leqslant \frac{L^2}{K}$$

- Usually we take $K \propto T$ to obtain convergence.

Recap

- The loss lanscape of DL training is **non-convex** and potentially difficult to optimize.
- Convergence to a global minimum for any smooth function is **prohibitive in high-dimensional spaces** (exponential in $d/2$).
- SGD (+ noise) can converge, within an error $\varepsilon > 0$, to a **local minimum** of any smooth function in roughly $O(\varepsilon^{-2})$ iterations.
- By relaxing the convexity constraint to a **PL condition**, one can obtain **convergence to the global optimum**.
- The PL condition is verified for neural networks whose **singular values of the Jacobian are bounded from below**.