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

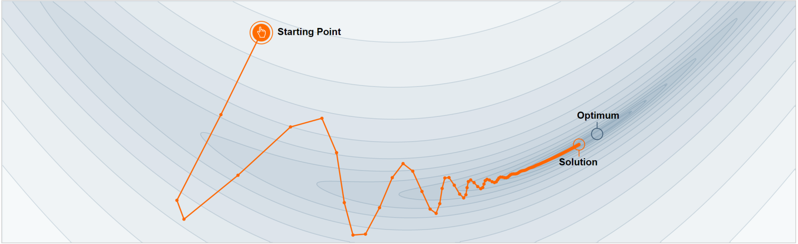
Gradient descent and co.

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 $\theta_{t+1} = \varphi_t\left(\theta_t, \nabla\mathcal{L}(\theta_t), s_t\right)$ 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

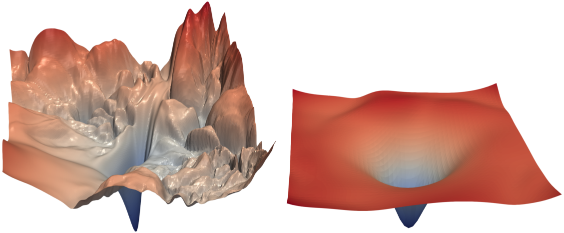
- **Non-convexity:** If $\mathcal{L}$ is **convex**, i.e. $\forall \theta, \theta', \mathcal{L}(\frac{\theta+\theta'}{2}) \leqslant \frac{\mathcal{L}(\theta)+\mathcal{L}(\theta')}{2}$, the optimization problem is **simple**. Most theoretical results use this assumption to prove convergence.
- **High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

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

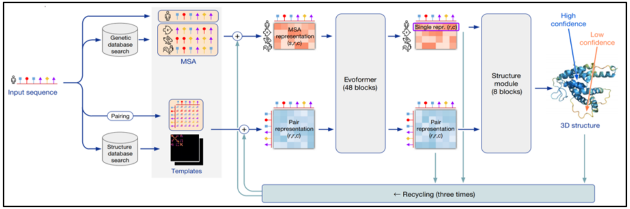
- Should provide **fast gradient computation** for composition of modules.
- Should explain performances of **non-convex SGD** (and its variants).
- Should work in **high-dimensional** spaces.
- Should extend to **non-smooth** objectives.
- Should have assumptions that are **reasonable for neural networks**.

Automatic differentiation

Differentiating composite functions

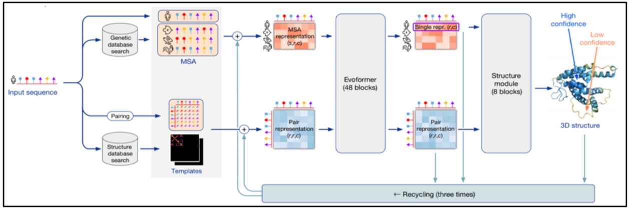
Computation graphs

Complex neural network architecture (e.g. AlphaFold)



Computation graphs

Complex neural network architecture (e.g. AlphaFold)

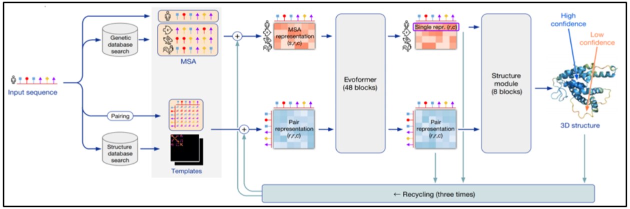


Code (e.g. Python)

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

Computation graphs

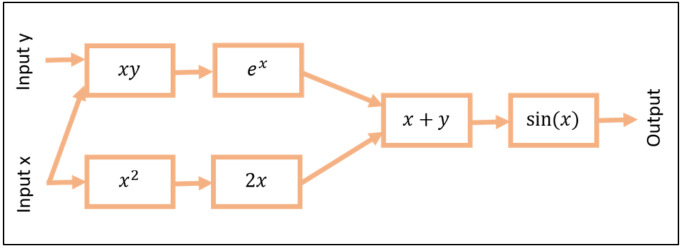
Complex neural network architecture (e.g. AlphaFold)



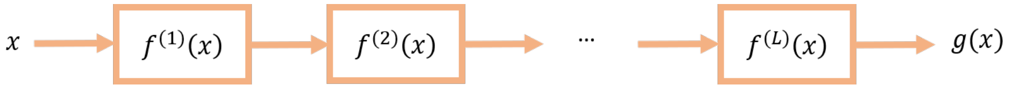
Code (e.g. Python)

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

Computation graph (DAG of mathematical operations)



Derivative of a composition of functions

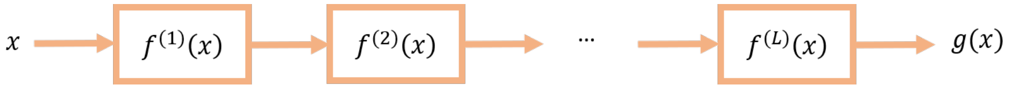


Composite function

- 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

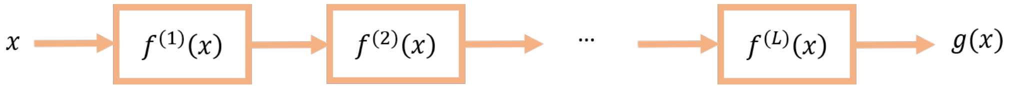
- Let $f^{(l)} : \mathbb{R}^{d(l-1)} \to \mathbb{R}^{d(l)}$ and $g(x) = g^{(L)}(x)$ where

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

- 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



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?

Simplifying assumptions

Assumptions

- The input is d-dimensional: $d^{(0)} = d$.
- The output is one dimensional: $d^{(L)} = 1$.
- Each layer $l \in [\![1, L]\!]$ is made of a **simple function**:
  - The function $f^{(l)}(x)$ takes a time T_F to compute.
  - Matrix-vector multiplication with the Jacobian $J_{f^{(l)}}(x)v$ or $wJ_{f^{(l)}}(x)$ takes a time T_B to compute.

Simplifying assumptions

Assumptions

- The input is d-dimensional: $d^{(0)} = d$.
- The output is one dimensional: $d^{(L)} = 1$.
- Each layer $l \in [\![1, L]\!]$ is made of a **simple function**:
  - The function $f^{(l)}(x)$ takes a time T_F to compute.
  - Matrix-vector multiplication with the Jacobian $J_{f^{(l)}}(x)v$ or $wJ_{f^{(l)}}(x)$ takes a time T_B to compute.

Example: linear layers

- The function is: $f^{(l)}(x) = Mx$.
- The Jacobian is: $J_{f^{(l)}}(x) = M^\top$.
- Then $T_F = T_B = d^{(l-1)}d^{(l)}$.

Finite differences approach

Naïve approach

- The gradient of g can be approximated by **finite differences**:

$$\nabla g(x)_i \approx \frac{g(x + \varepsilon e_i) - g(x)}{\varepsilon}$$

Finite differences approach

Naïve approach

- 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: $(d+1)LT_F$ proportional to **input dimension**.
- Memory cost: $\max_{l \in [\![1,L]\!]} d^{(l)}$.

We didn't use of the fact that g is a composition!

Matrix multiplication approach

Back to the particular form of the Jacobian

- We have $\nabla g(x)^\top = J_{f^{(L)}}\left(g^{(L-1)}(x)\right) \times \cdots \times J_{f^{(2)}}\left(g^{(1)}(x)\right) \times J_{f^{(1)}}(x)$.

Matrix multiplication approach

Back to the particular form of the Jacobian

- We have $\nabla g(x)^\top = J_{f^{(L)}}\left(g^{(L-1)}(x)\right) \times \cdots \times J_{f^{(2)}}\left(g^{(1)}(x)\right) \times J_{f^{(1)}}(x)$.
- There are $(L-1)!$ ways to compute products of L matrices.
- When output is 1-dimensional, most efficient way is **from output to input**.

Matrix multiplication approach

Back to the particular form of the Jacobian

- We have $\nabla g(x)^\top = J_{f^{(L)}}\left(g^{(L-1)}(x)\right) \times \cdots \times J_{f^{(2)}}\left(g^{(1)}(x)\right) \times J_{f^{(1)}}(x)$.
- There are $(L-1)!$ ways to compute products of L matrices.
- When output is 1-dimensional, most efficient way is **from output to input**.

Backpropagation algorithm (Rumelhart et al., 1986)

- We start from the input $x_0 = 1$ and

$$x_l = J_{f^{(l)}}\left(g^{(l-1)}(x)\right)^\top x_{l-1}$$

- The gradient is $\nabla g(x) = x_L$.

Matrix multiplication approach

Back to the particular form of the Jacobian

- We have $\nabla g(x)^\top = J_{f^{(L)}}\left(g^{(L-1)}(x)\right) \times \cdots \times J_{f^{(2)}}\left(g^{(1)}(x)\right) \times J_{f^{(1)}}(x)$.
- There are $(L-1)!$ ways to compute products of L matrices.
- When output is 1-dimensional, most efficient way is **from output to input**.

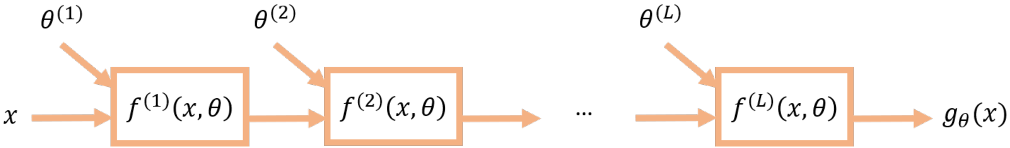
Backpropagation algorithm (Rumelhart et al., 1986)

- We start from the input $x_0 = 1$ and

$$x_l = J_{f^{(l)}}\left(g^{(l-1)}(x)\right)^\top x_{l-1}$$

- The gradient is $\nabla g(x) = x_L$.
- Computational complexity: $L(T_F + T_B)$.
- Memory cost: $\sum_{l \in [\![0, L-1]\!]} d^{(l)} + \max_{l \in [\![0, L]\!]} d^{(l)}$.

Sequential networks

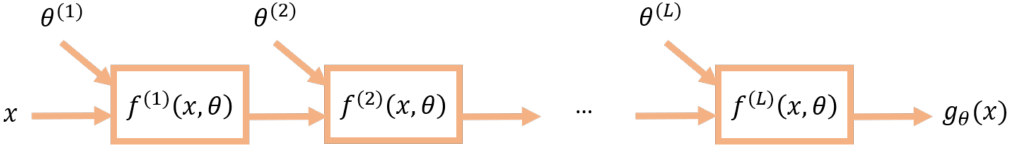


Definition (sequential networks)

- **Parameters:** Let $\theta = (\theta^{(1)}, \ldots, \theta^{(L)}) \in \mathbb{R}^p$ where $p = \sum_{l \in [\![1, L]\!]} p^{(l)}$.
- **Layers:** Let $f^{(l)} : \mathbb{R}^{d^{(l-1)}} \times \mathbb{R}^{p^{(l)}} \to \mathbb{R}^{d^{(l)}}$.
- **Output:** Then let $g_\theta(x) = g^{(L)}(x, \theta)$ where $g^{(0)}(x, \theta) = x$ and $\forall l \in [\![1, L]\!]$,

$$g^{(l)}(x, \theta) = f^{(l)}\left(g^{(l-1)}(x, \theta), \theta^{(l)}\right)$$

Derivatives of sequential networks



Chain rule

- We denote as $J_{f,x}(x,y)$ the Jacobian matrix of $x \mapsto f(x,y)$.
- To derive $w.r.t.$ $\theta^{(l)}$, we can treat x and $\theta^{(k)}$ for $k \neq l$ as fixed constants. We thus have a composite function and

$$J_{g,\theta^{(l)}}(x,\theta) = J_{f^{(L)},x}(x^{(L)},\theta^{(L)}) \times \cdots \times J_{f^{(l+1)},x}(x^{(l+1)},\theta^{(l+1)}) \times J_{f^{(l)},\theta}(x^{(l)},\theta^{(l)})$$

where $x^{(l)} = g^{(l-1)}(x,\theta)$.

Finite differences vs. forward vs. backward

Computational complexity

- **Finite differences:**
- **Backward propagation:**

Finite differences vs. forward vs. backward

Computational complexity

- **Finite differences:** $(p+1)LT_F$.
- **Backward propagation:**

Finite differences vs. forward vs. backward

Computational complexity

- **Finite differences:** $(p+1)LT_F$.
- **Backward propagation:** $L(T_F + 2T_B)$.

Finite differences vs. forward vs. backward

Computational complexity

- **Finite differences:** $(p+1)LT_F$.
- **Backward propagation:** $L(T_F + 2T_B)$.

Intuition

- Finite differences requires **one function call per parameter**.
- When $T_F \approx T_B$, backprop requires **three function calls for the whole gradient**.
- Interpretation as hypothesis testing:
  - Each partial derivative w.r.t. a parameter indicates if this parameter can describe the data.
  - With backprop, we can test **all hypotheses (i.e. all parameters) at once**.

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

$$\mathcal{L}(\theta_t) - \mathcal{L}(\theta^\star) = \Omega(t^{-1/d})$$

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 $\eta = \min\left\{\frac{1}{\beta}, \sqrt{\frac{2\Delta}{T\beta\sigma^2}}\right\}$ achieves the error

$$\mathbb{E}\Big[\min_{t\leqslant T} \|\nabla\mathcal{L}(\theta_t)\|\Big] \leqslant \frac{4\beta\Delta}{T} + \sqrt{\frac{8\beta\Delta\sigma^2}{T}}$$

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 $\eta = \min\left\{\frac{1}{\beta}, \sqrt{\frac{2\Delta}{T\beta\sigma^2}}\right\}$ achieves the error

$$\mathbb{E}\left[\min_{t \leqslant T} \|\nabla\mathcal{L}(\theta_t)\|\right] \leqslant \frac{4\beta\Delta}{T} + \sqrt{\frac{8\beta\Delta\sigma^2}{T}}$$

- 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 $\eta = \min\left\{\frac{1}{\beta}, \sqrt{\frac{2\Delta}{T\beta\sigma^2}}\right\}$ achieves the error

$$\mathbb{E}\Big[\min_{t\leqslant T} \|\nabla\mathcal{L}(\theta_t)\|\Big] \leqslant \frac{4\beta\Delta}{T} + \sqrt{\frac{8\beta\Delta\sigma^2}{T}}$$

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

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 $\eta = \min\left\{\frac{1}{\beta}, \sqrt{\frac{2\Delta}{T\beta\sigma^2}}\right\}$ achieves the error

$$\mathbb{E}\Big[\min_{t \leqslant T} \|\nabla\mathcal{L}(\theta_t)\|\Big] \leqslant \frac{4\beta\Delta}{T} + \sqrt{\frac{8\beta\Delta\sigma^2}{T}}$$

- Convergence in expectation implies cv. with high probability using Markov inequality.
- Convergence of the best iterate (i.e. smallest gradient norm). :(
- Without noise, $\eta = 1/\beta$ is optimal, and gives a convergence in $O(1/T)$.
- With noise, if η is fixed, there is a lower limit to the error.

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 $\eta = \min\left\{\frac{1}{\beta}, \sqrt{\frac{2\Delta}{T\beta\sigma^2}}\right\}$ achieves the error

$$\mathbb{E}\Big[\min_{t \leqslant T} \|\nabla \mathcal{L}(\theta_t)\|\Big] \leqslant \frac{4\beta\Delta}{T} + \sqrt{\frac{8\beta\Delta\sigma^2}{T}}$$

- Convergence in expectation implies cv. with high probability using Markov inequality.
- Convergence of the best iterate (i.e. smallest gradient norm). :(
- Without noise, $\eta = 1/\beta$ is optimal, and gives a convergence in $O(1/T)$.
- With noise, if η is fixed, there is a lower limit to the error.
- If $\eta = O(1/\sqrt{T})$ gives an optimal convergence 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

- SGD converges to a **stationary point** in time $O(\varepsilon^{-2})$.
- SGD + small noise converges to a **local minimum** in time $O(\varepsilon^{-2}\log(\varepsilon^{-1})^4)$.
- Convergence to a **global minimum impossible** in less than $\Omega(\varepsilon^{-d})$ for smooth functions.
- 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 $\|\nabla\mathcal{L}(\theta_t)\|^2 \geqslant 2\alpha(\mathcal{L}(\theta_t) - \mathcal{L}(\theta^\star))$.
- This implies, for $\varepsilon_t = \mathbb{E}(\mathcal{L}(\theta_t)) - \mathbb{E}(\mathcal{L}(\theta^\star))$,

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

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

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

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

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

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

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

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

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

- 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).
- 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.