Mathematics of Deep Learning

Structure and group invariances

Lessons: Kevin Scaman
TDs: Mathieu Even

Dauphine Université Paris | PSL

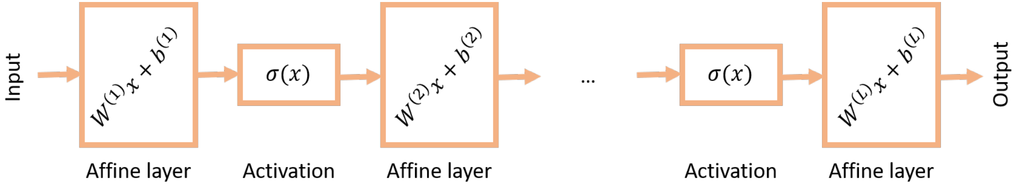
Class overview

1. Introduction and general overview 03/01
2. Non-convex optimization 10/01
3. **Structure 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

ReLU networks

Shape and structure of the output

ReLU networks (recap)



Definition (MLP)

Let $L \geqslant 1$, $(d^{(l)})_{l \in [\![0,L]\!]} \in {\mathbb{N}^*}^{L+1}$, and $\sigma(x) = \max\{0, x\}$. A *ReLU network* is an MLP with ReLU activations, i.e. :

$$g_\theta(x) = f^{(2L-1)} \circ f^{(2L-2)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x)$$

where $\forall l \in [\![1, L]\!]$, $f^{(2l-1)}(x) = W^{(l)}x + b^{(l)}$, $f^{(2l)}(x) = \sigma(x)$, $W^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l-1)}}$, $b^{(l)} \in \mathbb{R}^{d^{(l)}}$.

Simple properties

Definition (ReLU networks)

For $d, d' > 0$, let $\text{ReLU}_{d,d'}$ be the space of all ReLU networks s.t. $d^{(0)} = d$ and $d^{(L)} = d'$.

Simple properties

Definition (ReLU networks)

For $d, d' > 0$, let $\mathrm{ReLU}_{d,d'}$ be the space of all ReLU networks s.t. $d^{(0)} = d$ and $d^{(L)} = d'$.

Lemma (stability)

$\mathrm{ReLU}_{d,d}$ is **stable** by **addition** and **composition**. That is, $\forall g, g' \in \mathrm{ReLU}_{d,d}$,

$$g + g' \in \mathrm{ReLU}_{d,d} \quad \text{and} \quad g \circ g' \in \mathrm{ReLU}_{d,d}$$

Simple properties

Definition (ReLU networks)

For $d, d' > 0$, let $\text{ReLU}_{d,d'}$ be the space of all ReLU networks s.t. $d^{(0)} = d$ and $d^{(L)} = d'$.

Lemma (stability)

$\text{ReLU}_{d,d}$ is **stable** by **addition** and **composition**. That is, $\forall g, g' \in \text{ReLU}_{d,d}$,

$$g + g' \in \text{ReLU}_{d,d} \quad \text{and} \quad g \circ g' \in \text{ReLU}_{d,d}$$

Lemma (continuity and piecewise linearity)

A ReLU network is **continuous** and **piecewise linear**.

Simple properties

Definition (ReLU networks)

For $d, d' > 0$, let $\text{ReLU}_{d,d'}$ be the space of all ReLU networks s.t. $d^{(0)} = d$ and $d^{(L)} = d'$.

Lemma (stability)

$\text{ReLU}_{d,d}$ is **stable** by **addition** and **composition**. That is, $\forall g, g' \in \text{ReLU}_{d,d}$,

$$g + g' \in \text{ReLU}_{d,d} \quad \text{and} \quad g \circ g' \in \text{ReLU}_{d,d}$$

Lemma (continuity and piecewise linearity)

A ReLU network is **continuous** and **piecewise linear**.

Proof.

By continuity and piecewise linearity of a composition of continuous and piecewise linear functions. □

Structure of ReLU networks in practice

ReLU networks create affine regions

- Case of two layers and $d^{(2)} = 1$: $g_\theta(x) = \sum_i w_i^{(2)} \sigma(\langle w_i^{(1)}, x\rangle + b_i) + c$

Structure of ReLU networks in practice

ReLU networks create affine regions

- Case of two layers and $d^{(2)} = 1$: $g_\theta(x) = \sum_i w_i^{(2)} \sigma(\langle w_i^{(1)}, x \rangle + b_i) + c$
- Each ReLU activation can create a new affine region.

Structure of ReLU networks in practice

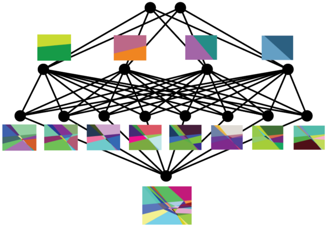
ReLU networks create affine regions

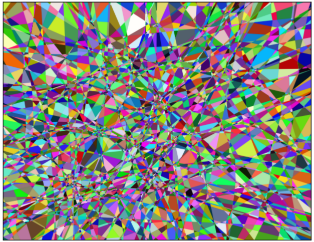
- Case of two layers and $d^{(2)} = 1$: $g_\theta(x) = \sum_i w_i^{(2)} \sigma(\langle w_i^{(1)}, x \rangle + b_i) + c$
- Each ReLU activation can create a new affine region.
- A large number of regions are created by the network.

Structure of ReLU networks in practice

ReLU networks create affine regions

- Case of two layers and $d^{(2)} = 1$: $g_\theta(x) = \sum_i w_i^{(2)} \sigma(\langle w_i^{(1)}, x \rangle + b_i) + c$
- Each ReLU activation can create a new affine region.
- A large number of regions are created by the network.
- Example of affine regions of a ReLU network trained on MNIST:





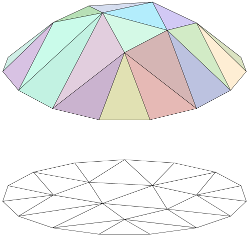
(image credits: Hanin & Rolnik, 2019)

Piecewise linear approximations

Definition (piecewise linearity)

A function $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ is a **continuous piecewise linear** function if there exists a **finite** set of closed and connected regions $(P_k)_{k\in[\![1,m]\!]} \subset \mathcal{P}(\mathbb{R}^d)$ such that $\cup_{k\in[\![1,m]\!]} P_k = \mathbb{R}^d$ and, for all $k \in [\![1, m]\!]$, f is affine on P_k, i.e. there exists $W_k \in \mathbb{R}^{d'\times d}, b_k \in \mathbb{R}$ s.t. $\forall x \in P_k$, $f(x) = W_k x + b_k$.

- We denote as number of regions of f the minimum number m of regions $(P_k)_{k\in[\![1,m]\!]}$ such that f is affine on them.
- As the P_k are closed, the function is necessarily **continuous**.
- As the number of regions is finite, the maximal regions are also **polytopes**.



(image credits: Wikipedia)

Piecewise linear approximations

Theorem (Arora et.al., 2018)

Every ReLU network is piecewise linear, and every continuous piecewise linear function $f : \mathbb{R}^d \to \mathbb{R}$ can be represented by a ReLU network with at most $\lceil \log_2(d+1) \rceil + 1$ depth.

Piecewise linear approximations

Theorem (Arora et.al., 2018)

Every ReLU network is piecewise linear, and every continuous piecewise linear function $f : \mathbb{R}^d \to \mathbb{R}$ can be represented by a ReLU network with at most $\lceil \log_2(d+1) \rceil + 1$ depth.

Proof.

- ReLU networks are continuous and piecewise linear by construction.

Piecewise linear approximations

Theorem (Arora et.al., 2018)

Every ReLU network is piecewise linear, and every continuous piecewise linear function $f : \mathbb{R}^d \to \mathbb{R}$ can be represented by a ReLU network with at most $\lceil \log_2(d+1) \rceil + 1$ depth.

Proof.

- ReLU networks are continuous and piecewise linear by construction.
- The other side is based on a universal representation of piecewise-linear functions: $f(x) = \sum_j s_j \max_{i \in S_j} \ell_i(x)$ where $s_j \in \{-1, 1\}$, $S_j \subset [\![1, K]\!]$ and $\{\ell_i\}_{i \in [\![1,K]\!]}$ are K affine functions.

Piecewise linear approximations

Theorem (Arora et.al., 2018)

Every ReLU network is piecewise linear, and every continuous piecewise linear function $f : \mathbb{R}^d \to \mathbb{R}$ can be represented by a ReLU network with at most $\lceil \log_2(d+1) \rceil + 1$ depth.

Proof.

- ReLU networks are continuous and piecewise linear by construction.
- The other side is based on a universal representation of piecewise-linear functions: $f(x) = \sum_j s_j \max_{i \in S_j} \ell_i(x)$ where $s_j \in \{-1, 1\}$, $S_j \subset [\![1, K]\!]$ and $\{\ell_i\}_{i \in [\![1,K]\!]}$ are K affine functions.
- See Exercise. ☺

□

What next?

Model complexity

- We saw that a depth of $\lceil \log_2(d+1) \rceil + 1$ is sufficient for any function with d regions.
- It does not say how this constructed network deals with approximation and noise.
- It does not say how to design the ReLU network in practice (decreasing/constant/increasing layer size?).
- The number of regions can be used as a proxy for **complexity of the model**.

This notion of complexity is not perfect, as the linear regions are not independent...

Number of piecewise linear regions (case $L = 2$)

Theorem (Arora et.al., 2018)

Given any piecewise linear function $f : \mathbb{R} \to \mathbb{R}$ with $m \geqslant 2$ pieces there exists a 2-layer ReLU network with at most $d^{(1)} \leqslant m$ that can represent f. Moreover, any 2-layer ReLU network that represents f has size at least $d^{(1)} \geqslant m - 1$.

Number of piecewise linear regions (case $L = 2$)

Theorem (Arora et.al., 2018)

Given any piecewise linear function $f : \mathbb{R} \to \mathbb{R}$ with $m \geqslant 2$ pieces there exists a 2-layer ReLU network with at most $d^{(1)} \leqslant m$ that can represent f. Moreover, any 2-layer ReLU network that represents f has size at least $d^{(1)} \geqslant m - 1$.

Proof.

- First, if $f(x) = \sum_{i=1}^{d^{(1)}} w_i^{(2)} \sigma(w_i^{(1)} x + b_i) + c$ and has m regions, then f has $m - 1$ breaking points. However, the number of non-differentiable points is smaller than $d^{(1)}$.

Number of piecewise linear regions (case $L = 2$)

Theorem (Arora et.al., 2018)

Given any piecewise linear function $f : \mathbb{R} \to \mathbb{R}$ with $m \geqslant 2$ pieces there exists a 2-layer ReLU network with at most $d^{(1)} \leqslant m$ that can represent f. Moreover, any 2-layer ReLU network that represents f has size at least $d^{(1)} \geqslant m - 1$.

Proof.

- First, if $f(x) = \sum_{i=1}^{d^{(1)}} w_i^{(2)} \sigma(w_i^{(1)} x + b_i) + c$ and has m regions, then f has $m - 1$ breaking points. However, the number of non-differentiable points is smaller than $d^{(1)}$.
- Recursively: for $m = 2$, $f(x) = a_1 \sigma(x - x_0) - a_2 \sigma(x_0 - x) + f(x_0)$.

Number of piecewise linear regions (case $L = 2$)

Theorem (Arora et.al., 2018)

Given any piecewise linear function $f : \mathbb{R} \to \mathbb{R}$ with $m \geqslant 2$ pieces there exists a 2-layer ReLU network with at most $d^{(1)} \leqslant m$ that can represent f. Moreover, any 2-layer ReLU network that represents f has size at least $d^{(1)} \geqslant m - 1$.

Proof.

- First, if $f(x) = \sum_{i=1}^{d^{(1)}} w_i^{(2)} \sigma(w_i^{(1)} x + b_i) + c$ and has m regions, then f has $m - 1$ breaking points. However, the number of non-differentiable points is smaller than $d^{(1)}$.
- Recursively: for $m = 2$, $f(x) = a_1 \sigma(x - x_0) - a_2 \sigma(x_0 - x) + f(x_0)$.
- If OK for m and f has $m + 1$ regions, we take the last breaking point x_m and remove it from f by taking $g(x) = f(x) - (a_{m+1} - a_m)\sigma(x - x_m)$ and apply recursion.

□

Number of piecewise linear regions (case $L > 2$)

- Adding a neuron in a layer tends to **add** a new affine region.
- Adding a layer tends to **multiply** the number of affine regions.

Number of piecewise linear regions (case $L > 2$)

- Adding a neuron in a layer tends to **add** a new affine region.
- Adding a layer tends to **multiply** the number of affine regions.

Simple bound

- Each ReLU activation creates a halfspace cut.
- This multiplies at most the number of regions by 2.
- We thus have $m \leqslant 2^D$ where $D = \sum_{i=1}^{L-1} d^{(i)}$ is the number of ReLU activations.
- There are ReLU networks that achieve such an exponential number of regions.

Number of piecewise linear regions (case $L > 2$)

Hyperplane arrangements (Zaslavsky, 1975)

The number of regions defined by n hyperplanes in $\mathbb{R}^d$ is at most $\sum_{i=0}^{d} \binom{n}{i}$.

Number of piecewise linear regions (case $L > 2$)

Hyperplane arrangements (Zaslavsky, 1975)

The number of regions defined by n hyperplanes in $\mathbb{R}^d$ is at most $\sum_{i=0}^{d} \binom{n}{i}$.

More refined bound (Raghu et al., 2017)

- Each activation layer can be seen as $d^{(l)}$ hyperplanes in a space of dimension $d^{(l-1)}$.

Number of piecewise linear regions (case $L > 2$)

Hyperplane arrangements (Zaslavsky, 1975)

The number of regions defined by n hyperplanes in $\mathbb{R}^d$ is at most $\sum_{i=0}^{d} \binom{n}{i}$.

More refined bound (Raghu et al., 2017)

- Each activation layer can be seen as $d^{(l)}$ hyperplanes in a space of dimension $d^{(l-1)}$.
- At most, each of these $\sum_{i=0}^{d^{(l-1)}} \binom{d^{(l)}}{i}$ regions contain all the regions created by the previous layers, hence

$$m \leqslant \prod_{l=1}^{L-1} \sum_{i=0}^{d^{(l-1)}} \binom{d^{(l)}}{i}$$

Number of piecewise linear regions (case $L > 2$)

Hyperplane arrangements (Zaslavsky, 1975)

The number of regions defined by n hyperplanes in $\mathbb{R}^d$ is at most $\sum_{i=0}^{d} \binom{n}{i}$.

More refined bound (Raghu et al., 2017)

- Each activation layer can be seen as $d^{(l)}$ hyperplanes in a space of dimension $d^{(l-1)}$.
- At most, each of these $\sum_{i=0}^{d^{(l-1)}} \binom{d^{(l)}}{i}$ regions contain all the regions created by the previous layers, hence

$$m \leqslant \prod_{l=1}^{L-1} \sum_{i=0}^{d^{(l-1)}} \binom{d^{(l)}}{i}$$

Recap

- ReLU networks encode **continuous** and **piece-linear** functions.

Recap

- ReLU networks encode **continuous** and **piece-linear** functions.
- Only $\lceil \log_2(d+1) \rceil + 1$ **layers** are **needed** to encode any piece-linear function.

Recap

- ReLU networks encode **continuous** and **piece-linear** functions.
- Only $\lceil \log_2(d+1) \rceil + 1$ **layers** are **needed** to encode any piece-linear function.
- Complexity of the function measured by the number m of **pieces**, and

$$m \leqslant \prod_{l=1}^{L-1} \sum_{i=0}^{d^{(l-1)}} \binom{d^{(l)}}{i}$$

Recap

- ReLU networks encode **continuous** and **piece-linear** functions.
- Only $\lceil \log_2(d+1) \rceil + 1$ **layers** are **needed** to encode any piece-linear function.
- Complexity of the function measured by the number m of **pieces**, and

$$m \leqslant \prod_{l=1}^{L-1} \sum_{i=0}^{d^{(l-1)}} \binom{d^{(l)}}{i}$$

- As piece-wise linear function can approximate any continuous function, **ReLU networks do not assume any specific structure or invariance on the data**.

Recap

- ReLU networks encode **continuous** and **piece-linear** functions.
- Only $\lceil \log_2(d+1) \rceil + 1$ **layers** are **needed** to encode any piece-linear function.
- Complexity of the function measured by the number m of **pieces**, and

$$m \leqslant \prod_{l=1}^{L-1} \sum_{i=0}^{d^{(l-1)}} \binom{d^{(l)}}{i}$$

- As piece-wise linear function can approximate any continuous function, **ReLU networks do not assume any specific structure or invariance on the data**.

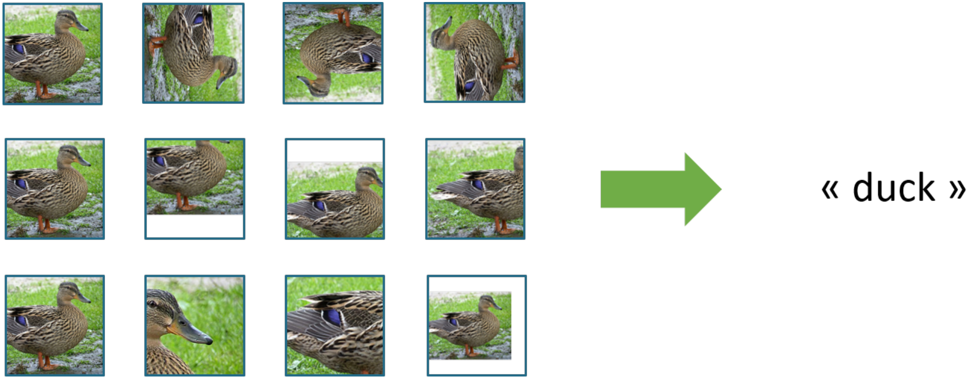
ReLU networks with $\lceil \log_2(d+1) \rceil + 1$ layers may not be easily trained!

Group invariances and CNNs

Invariance and equivariance to input transformations

Invariances in object recognition tasks

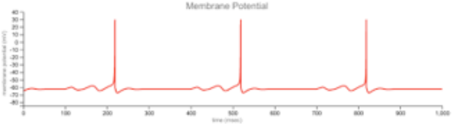
Ideally, we would like an architecture that does not depend on orientation, scale, position, lighting conditions,... of the object.



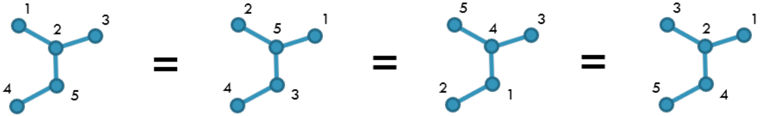
Invariances beyong image recognition

Prior information hardwired in the architecture

- Inductive biases play a key role in the performance of DL models
- **Times series:** translations, periodicity, symmetry, causality



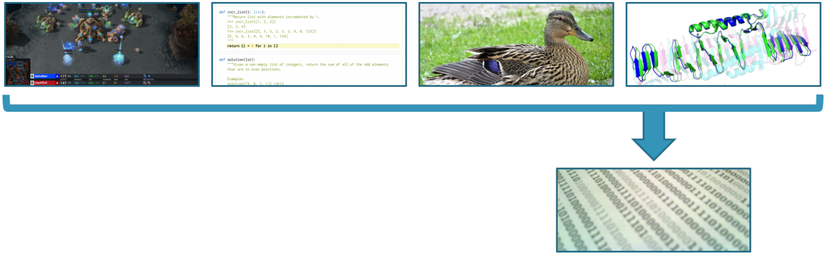
- **Graphs:** permutations of the indices



Importance of inductive bias

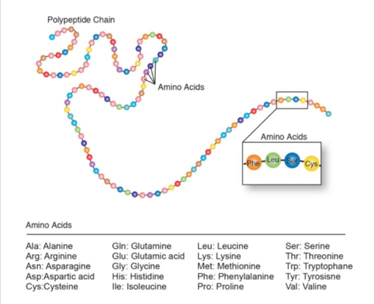
Bits as universal representations

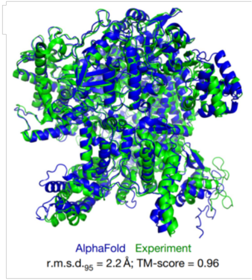
- All data that is stored on a hard drive can be represented as a sequence of 0s and 1s...
- ... but RNNs are **not** the solution to everything!
- **Imposing the right bias** is vital to help the model learn the right **patterns**, **structures** and **invariances**.



Practical example: AlphaFold 2

- **Objective:** find the 3D structure of a protein based on its amino acid sequence.
- **Invariance:** the output is invariant by translation and rotation.





https://www.genome.gov/genetics-glossary/Amino-Acids

Transformations of the input space

- $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ is the space of functions $f : \mathcal{X} \to \mathcal{Y}$ from input space $\mathcal{X}$ to output space $\mathcal{Y}$.
- We denote as **transformation** a function $\tau \in \mathcal{F}(\mathcal{X}, \mathcal{X})$ mapping $\mathcal{X}$ to itself.
- We denote as $\mathcal{T} \subset \mathcal{F}(\mathcal{X}, \mathcal{X})$ a **set of transformations** of the (input) space $\mathcal{X}$.

Transformations of the input space

- $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ is the space of functions $f : \mathcal{X} \to \mathcal{Y}$ from input space $\mathcal{X}$ to output space $\mathcal{Y}$.
- We denote as **transformation** a function $\tau \in \mathcal{F}(\mathcal{X}, \mathcal{X})$ mapping $\mathcal{X}$ to itself.
- We denote as $\mathcal{T} \subset \mathcal{F}(\mathcal{X}, \mathcal{X})$ a **set of transformations** of the (input) space $\mathcal{X}$.

Examples

- Translations by a vector: $\mathcal{T} = \{\tau_c\}_{c \in \mathbb{R}^d}$ s.t. $\forall x \in \mathbb{R}^d$, $\tau_c(x) = x + c$.
- Rotations of complex numbers: $\mathcal{T} = \{\tau_\theta\}_{\theta \in [0, 2\pi)}$ s.t. $\forall x \in \mathbb{C}$, $\tau_\theta(x) = xe^{i\theta}$.
- Projections on the coordinates: $\mathcal{T} = \{\tau_i\}_{i \in [\![1, d]\!]}$ s.t. $\forall x \in \mathbb{R}^d$, $\tau_i(x) = x_i e_i$.

A transformation is not necessarily bijective!

Invariance and equivariance

Definition (invariance)

A function $f : \mathcal{X} \to \mathcal{Y}$ is invariant w.r.t. the transformations $\mathcal{T}$ iff, for all $x \in \mathcal{X}$ and $\tau \in \mathcal{T}$,

$$f \circ \tau(x) = f(x)$$

Definition (equivariance)

A function $f : \mathcal{X} \to \mathcal{X}$ is equivariant w.r.t. the transformations $\mathcal{T}$ iff, for all $x \in \mathcal{X}$ and $\tau \in \mathcal{T}$,

$$f \circ \tau(x) = \tau \circ f(x)$$

In other words, f **commutes** with τ.

Invariance and equivariance

Lemma (equivalence graph)

Let $G = (V, E)$ be the graph defined by $V = \mathcal{X}$ and $\{x, y\} \in E$ if and only if $\exists \tau \in \mathcal{T}$ s.t. $\tau(x) = y$ or $\tau(y) = x$. Then, a fuction is $\mathcal{T}$-invariant if and only if it is constant on the connected components of G.

Lemma (generated group)

A function invariant (resp. equivariant) to a set of bijective transformations $\mathcal{T}$ is also invariant (resp. equivariant) to the group of transformations generated by $\mathcal{T}$ and composition.

Group actions

Definition (group actions)

A group $\mathcal{G}$ acting on a space $\mathcal{X}$ is a mapping $\tau : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$ that verifies (with the notation $\tau_g \in \mathcal{F}(\mathcal{X}, \mathcal{X})$ s.t. $\tau_g(x) = \tau(g, x)$):

1. **Identity:** if $e \in \mathcal{G}$ is the identity element, then $\tau_e = \mathrm{Id}$.
2. **Compatibility:** $\forall g, h \in \mathcal{G}$, we have $\tau_g \circ \tau_h = \tau_{gh}$.

This action defines a set of transformations $\mathcal{T}_{\mathcal{G}} = \{\tau_g\}_{g \in \mathcal{G}}$.

Examples

- Periodicity: $\mathcal{G} = \mathbb{Z}$ and $\tau_k(x) = x + kv$ where $v > 0$ is the period.
- Permutation: $\mathcal{G} = S_d$ and $\tau_\sigma(x)_i = x_{\sigma(i)}$ where $\sigma \in S_d$ is a permutation of the indices.

Back to images

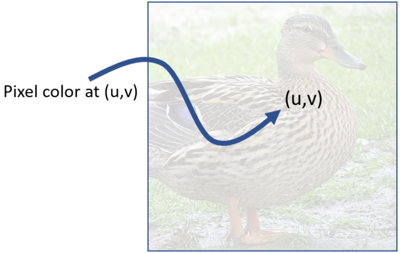
Transformation of the input $\neq$ transformation of the underlying space!

Functions as input

- Often, the **input is itself a function**, e.g. pixels of an image, intensity of a signal...
- We thus have $\mathcal{X} = \mathcal{F}(\mathcal{S}, \mathbb{R}^d)$, where $\mathcal{S}$ is the (usually finite) underlying space.

Examples

- Sets: $\mathcal{S} = [\![1, N]\!]$. Then, $x = (x_i)_{i \in [\![1,N]\!]}$.
- Images: $\mathcal{S} = [\![1, N]\!] \times [\![1, M]\!]$ and $d = 3$. Then, $x = (x_{ij})_{i \in [\![1,N]\!], j \in [\![1,M]\!]}$.
- Infinite images: $\mathcal{S} = \mathbb{R}^2$ and $d = 3$. Then, $x : \mathbb{R}^2 \mapsto \mathbb{R}^3$.
- Time series: $\mathcal{S} = \mathbb{R}$. Then, $x : \mathbb{R} \mapsto \mathbb{R}^d$.



From underlying space to input space

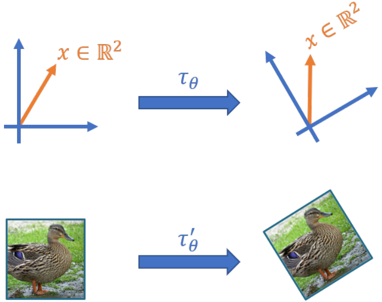
Lemma

If $\mathcal{G}$ is a group acting on $\mathcal{S}$ and $\{\tau_g\}$ are the associated transformations, then we can define an action on $\mathcal{F}(\mathcal{S}, \mathbb{R}^d)$ via:

$$\tau'_g(f)(x) = f(\tau_g(x))$$

Examples

- For example, the group of 2D rotations induces a group of transformations on the images.



Generic architecture for invariant neural networks

A (naïve) recipe for invariant neural networks

- A simple solution to create invariant neural networks is to sum or average over all transformations:

$$f_{\mathsf{inv}}(x) = \sum_{\tau \in \mathcal{T}} f(\tau(x))$$

- Ok for small transformation sets, **prohibitive in most cases** (permutations: $|S_n| = n!$).

Generic architecture for invariant neural networks

A (naïve) recipe for invariant neural networks

- A simple solution to create invariant neural networks is to sum or average over all transformations:

$$f_{\text{inv}}(x) = \sum_{\tau \in \mathcal{T}} f(\tau(x))$$

- Ok for small transformation sets, **prohibitive in most cases** (permutations: $|S_n| = n!$).
- A more tractable alternative is to take **one transformation at random**. Can lead to a large variance, and weak theoretical guarantees.

Generic architecture for invariant neural networks

A (naïve) recipe for invariant neural networks

- A simple solution to create invariant neural networks is to sum or average over all transformations:

$$f_{\text{inv}}(x) = \sum_{\tau \in \mathcal{T}} f(\tau(x))$$

- Ok for small transformation sets, **prohibitive in most cases** (permutations: $|S_n| = n!$).
- A more tractable alternative is to take **one transformation at random**. Can lead to a large variance, and weak theoretical guarantees.
- In practice, we often augment the dataset $\mathcal{D}_n = \{(x_i, y_i)\}_{i \in [\![1,n]\!]}$ with transformed inputs:

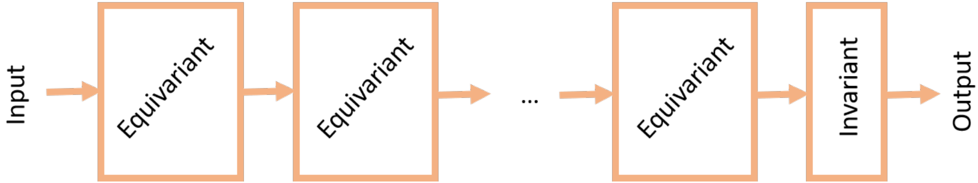
$$\mathcal{D}'_n = \{(\tau(x_i), y_i)\}_{i \in [\![1,n]\!], \tau \in \mathcal{T}}$$

- **Drawback:** Increases **training time** and **size of the model**.

Generic architecture for invariant neural networks

A (better) recipe for invariant neural networks

- Sequence of equivariant operations (usually affine + activations).
- Final invariant operation.



- In practice, we need to design **equivariant affine layers** (activation are usually ok).

The case of translation equivariance (finite setting)

To simplify our analysis, we consider translations on the discrete circle: $\mathcal{S} = [\![1, N]\!]$ and, for any translation distance $u \in [\![1, N]\!]$ and any input $x \in \mathbb{R}^N$,

$$\tau_u(x)_i = x_{i+u[N]}$$

Lemma (convolutions)

The only linear functions that are **translation equivariant** w.r.t. the underlying space $\mathcal{S} = [\![1, N]\!]$ are the **convolutions**.

The case of translation equivariance (finite setting)

To simplify our analysis, we consider translations on the discrete circle: $\mathcal{S} = [\![1, N]\!]$ and, for any translation distance $u \in [\![1, N]\!]$ and any input $x \in \mathbb{R}^N$,

$$\tau_u(x)_i = x_{i+u[N]}$$

Lemma (convolutions)

The only linear functions that are **translation equivariant** w.r.t. the underlying space $\mathcal{S} = [\![1, N]\!]$ are the **convolutions**.

Proof.

- By linearity, we have $f(x)_i = \sum_j M_{i,j} x_j$.
- Then, by invariance, $\sum_j M_{i,j} x_{j+u[N]} = \sum_j M_{i+u[N],j} x_j$ and $\forall i, j, u$,

$$M_{i,j} = M_{i+u[N],j+u[N]}$$

The case of translation equivariance (continuous setting)

We now consider translation on the plane: $\mathcal{S} = \mathbb{R}^2$ and, for any translation vector $v \in \mathbb{R}^2$ and any input image $x : \mathbb{R}^2 \to \mathbb{R}$, $\tau_v(x) : u \mapsto x(u+v)$. As the input space is infinite dimensional, we limit ourselves to integral operators of the form: $f(x) : u \mapsto \int_w K(u,w)x(w)dw$.

Convolutions as equivariant integral operators

The only integral operators that are **translation equivariant** w.r.t. the underlying space $\mathcal{S} = \mathbb{R}^2$ are the **convolutions**.

The case of translation equivariance (continuous setting)

We now consider translation on the plane: $\mathcal{S} = \mathbb{R}^2$ and, for any translation vector $v \in \mathbb{R}^2$ and any input image $x : \mathbb{R}^2 \to \mathbb{R}$, $\tau_v(x) : u \mapsto x(u+v)$. As the input space is infinite dimensional, we limit ourselves to integral operators of the form: $f(x) : u \mapsto \int_w K(u,w)x(w)dw$.

Convolutions as equivariant integral operators

The only integral operators that are **translation equivariant** w.r.t. the underlying space $\mathcal{S} = \mathbb{R}^2$ are the **convolutions**.

Proof.

- We have $f \circ \tau_v(x)(u) = \int_w K(u,w)x(w+v)dw = \int_w K(u,w-v)x(w)dw$.
- We have $\tau_v \circ f(x)(u) = \int_w K(u+v,w)x(w)dw$.
- As the two terms should be equal for any function x, we have, $\forall u, w$, $K(u,w) = K(u-v,0)$ and f is a convolution:

$$f(x) = K(\,\cdot\,,0) * x$$

The case of permutation invariance

We now consider permutation of indices: $\mathcal{S} = S_n$ and $\tau_\sigma(x)_i = x_{\sigma(i)}$.

Permutation equivariant affine layers

- If we use the same method, $f(x) = \sum_j M_{ij} x_j$, we get $M_{ij} = M_{kl}$ if $i \neq j$ and $k \neq l$
- This is quite restrictive, as we only have two parameters per layer...

The case of permutation invariance

We now consider permutation of indices: $\mathcal{S} = S_n$ and $\tau_\sigma(x)_i = x_{\sigma(i)}$.

Permutation equivariant affine layers

- If we use the same method, $f(x) = \sum_j M_{ij} x_j$, we get $M_{ij} = M_{kl}$ if $i \neq j$ and $k \neq l$
- This is quite restrictive, as we only have two parameters per layer...

DeepSet (Zaheer et.al., 2017)

- Instead, we put the complexity in the activation:

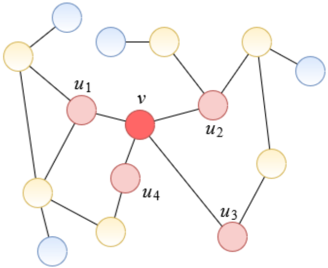
$$g_\theta(x) = \psi\left(\sum_i \phi(x_i)\right)$$

- The functions ϕ and ψ are usually MLPs and contain the parameters of the model.
- This is sufficient to represent any permutation invariant function.

Graph neural networks (GNN)

Message passing schemes

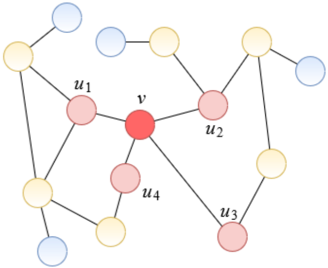
- Relies on the transfer of messages between neighbors
- Composed of three steps:
  - **Initialization:** Graph $G = (V, E)$, node attributes $u_{i,0} \in \mathbb{R}^d$.
  - **Aggregation:** $u_{i,l+1} = \phi_l(u_{i,l}, \{u_{j,l} \mid \{i,j\} \in E\})$.
  - **Readout:** $u_G = \psi(\{u_{i,L} \mid i \in V\})$.
- The functions ϕ_l and ψ are permutation invariant neural networks (e.g. DeepSet or simple affine functions).



Graph neural networks (GNN)

Message passing schemes

- Relies on the transfer of messages between neighbors
- Composed of three steps:
  - **Initialization:** Graph $G = (V, E)$, node attributes $u_{i,0} \in \mathbb{R}^d$.
  - **Aggregation:** $u_{i,l+1} = \phi_l(u_{i,l}, \{u_{j,l} \mid \{i,j\} \in E\})$.
  - **Readout:** $u_G = \psi(\{u_{i,L} \mid i \in V\})$.
- The functions ϕ_l and ψ are permutation invariant neural networks (e.g. DeepSet or simple affine functions).
- Quite large framework... but unfortunately not expressive enough!
- Incapable of counting triangles (see exercise).



Recap

ReLU networks

- ReLU networks are exactly the continuous piecewise linear functions.
- The number of regions can grow exponentially in the depth.

Group invariances

- MLPs + translation invariance = CNNs.
- Group invariance can often be imposed by restricting affine layers to be equivariant.

Next lesson

- Approximation capabilities of MLPs.