

On the Happy Marriage of Kernel Methods and Deep Learning

Julien Mairal

Inria Grenoble

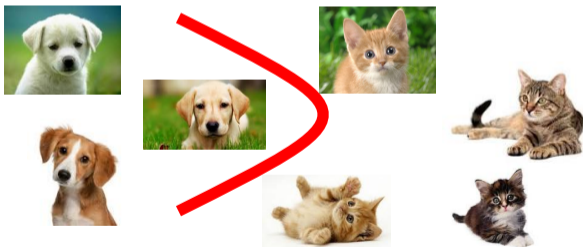
MaLGa Seminar



Context of supervised learning

The goal is to learn a **prediction function** $f : \mathcal{X} \rightarrow \mathcal{Y}$ given labeled training data $(x_i, y_i)_{i=1, \dots, n}$ with x_i in \mathcal{X} , and y_i in \mathcal{Y} :

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}} .$$



[Vapnik, 1995, Bottou, Curtis, and Nocedal, 2016]...

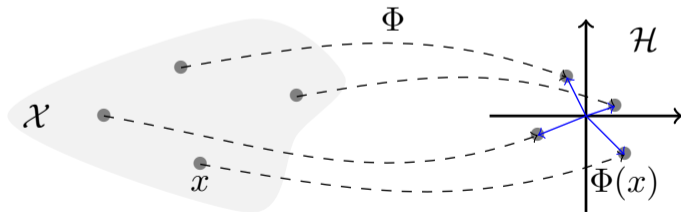
Kernel Methods 1/2

In the context of supervised learning with labels in \mathbb{R} ,

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2.$$

- **map** data x in \mathcal{X} to a Hilbert space and work with **linear forms**:

$$\Phi : \mathcal{X} \rightarrow \mathcal{H} \quad \text{and} \quad f(x) = \langle \Phi(x), f \rangle_{\mathcal{H}}.$$



[Shawe-Taylor and Cristianini, 2004, Schölkopf and Smola, 2002]...

Kernel Methods 2/2

In the context of supervised learning with labels in \mathbb{R} ,

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2.$$

- $f(x) = \langle \Phi(x), f \rangle_{\mathcal{H}}$ but $\Phi(x)$ may be very high- or infinite-dimensional.
- then, only manipulate **inner-products** $K(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}$ (kernel trick).
- Alternatively, compute a finite-dimensional approximate embedding $f(x) \approx w^{\top} \Psi(x)$;
- **regularize** with $\|\cdot\|_{\mathcal{H}}$ (encourages smoothness);

Kernel Methods 2/2

In the context of supervised learning with labels in \mathbb{R} ,

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2.$$

- $f(x) = \langle \Phi(x), f \rangle_{\mathcal{H}}$ but $\Phi(x)$ may be very high- or infinite-dimensional.
- then, only manipulate **inner-products** $K(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}$ (kernel trick).
- Alternatively, compute a finite-dimensional approximate embedding $f(x) \approx w^{\top} \Psi(x)$;
- **regularize** with $\|\cdot\|_{\mathcal{H}}$ (encourages smoothness);

If you want to know more (24 hours course)

<http://members.cbio.mines-paristech.fr/~jvert/svn/kernelcourse/slides/master2017/master2017.pdf>

Relation with deep learning?

A functional space viewpoint: kernels for deep networks

- View deep networks as functions in some functional space;
- Non-parametric models, natural measures of complexity (e.g., norms);
- Linearization $f(x) = \langle f, \Phi(x) \rangle$ decouples learning f from data representation $\Phi(x)$.

Relation with deep learning?

A functional space viewpoint: kernels for deep networks

- View deep networks as functions in some functional space;
- Non-parametric models, natural measures of complexity (e.g., norms);
- Linearization $f(x) = \langle f, \Phi(x) \rangle$ decouples learning f from data representation $\Phi(x)$.

What is an appropriate functional space?

Relation with deep learning?

A functional space viewpoint: kernels for deep networks

- View deep networks as functions in some functional space;
- Non-parametric models, natural measures of complexity (e.g., norms);
- Linearization $f(x) = \langle f, \Phi(x) \rangle$ decouples learning f from data representation $\Phi(x)$.

What is an appropriate functional space?

Deep learning for kernels

- Scalable learning with finite-dimensional embeddings;
- Deep networks with a geometric interpretation and regularization principles;
- End-to-end learning with kernels?

Relation with deep learning?

A functional space viewpoint: kernels for deep networks

- View deep networks as functions in some functional space;
- Non-parametric models, natural measures of complexity (e.g., norms);
- Linearization $f(x) = \langle f, \Phi(x) \rangle$ decouples learning f from data representation $\Phi(x)$.

What is an appropriate functional space?

Deep learning for kernels

- Scalable learning with finite-dimensional embeddings;
- Deep networks with a geometric interpretation and regularization principles;
- End-to-end learning with kernels?

How do we proceed?

Graph Modeling

- D. Chen, L. Jacob and J. Mairal. Convolutional Kernel Networks for Graph-Structured Data. International Conference on Machine Learning (ICML). 2020.

Learning graph representations

State-of-the-art models for representing graphs:

- **Deep learning for graphs**: graph neural networks (GNNs);
- **Graph kernels**: Weisfeiler-Lehman (WL) graph kernels;
- **Hybrid models** attempt to bridge both worlds: graph neural tangent kernels (GNTK).

Learning graph representations

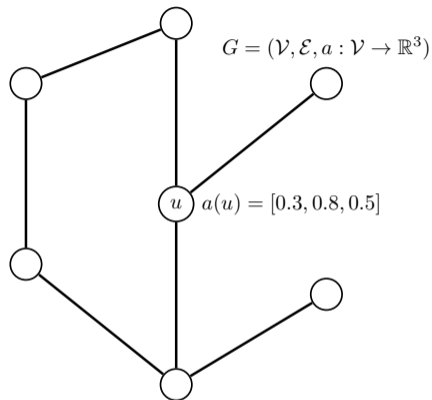
State-of-the-art models for representing graphs:

- **Deep learning for graphs**: graph neural networks (GNNs);
- **Graph kernels**: Weisfeiler-Lehman (WL) graph kernels;
- **Hybrid models** attempt to bridge both worlds: graph neural tangent kernels (GNTK).

Our model:

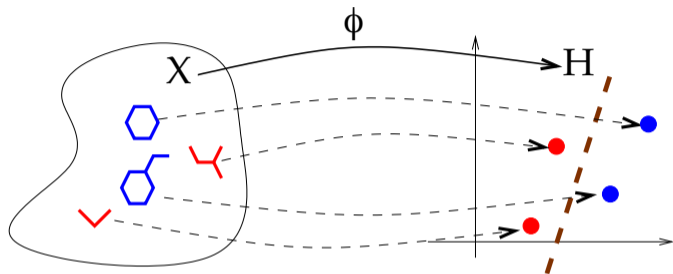
- A new type of **multilayer** graph kernel: more **expressive** than WL kernels;
- Learning easy-to-regularize and scalable **unsupervised** graph representations;
- Learning **supervised** graph representations like GNNs.

Graphs with node attributes



- A graph is defined as a triplet $(\mathcal{V}, \mathcal{E}, a)$;
- \mathcal{V} and \mathcal{E} correspond to the set of vertices and edges;
- $a : \mathcal{V} \rightarrow \mathbb{R}^d$ is a function assigning attributes to each node.

Graph kernel mappings

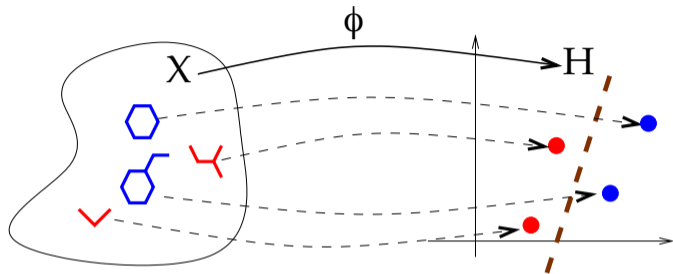


- Map each graph G in \mathcal{X} to a vector $\Phi(G)$ in \mathcal{H} , which lends itself to learning tasks.
- A large class of graph kernel mappings can be written in the form

$$\Phi(G) := \sum_{u \in \mathcal{V}} \varphi_{\text{base}}(\ell_G(u)) \quad \text{where } \varphi_{\text{base}} \text{ embeds some local patterns } \ell_G(u) \text{ to } \mathcal{H}.$$

[Shervashidze et al., 2011, Lei et al., 2017, Kriege et al., 2019]

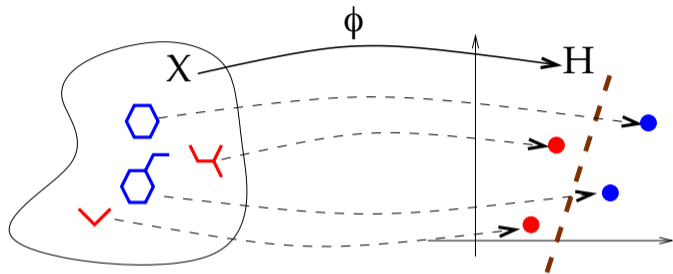
Graph kernel mappings



- Map each graph G in \mathcal{X} to a vector $\Phi(G)$ in \mathcal{H} , which lends itself to learning tasks.
- A large class of graph kernel mappings can be written in the form

$$K(G, G') = \left\langle \underbrace{\sum_{u \in \mathcal{V}} \varphi_{\text{base}}(\ell_G(u))}_{\Phi(G)}, \underbrace{\sum_{u' \in \mathcal{V}'} \varphi_{\text{base}}(\ell_{G'}(u'))}_{\Phi(G')} \right\rangle.$$

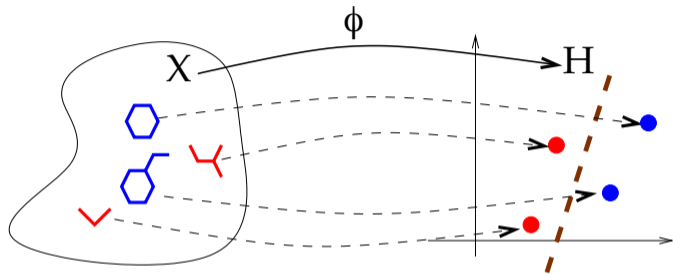
Graph kernel mappings



- Map each graph G in \mathcal{X} to a vector $\Phi(G)$ in \mathcal{H} , which lends itself to learning tasks.
- A large class of graph kernel mappings can be written in the form

$$K(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \langle \varphi_{\text{base}}(\ell_G(u)), \varphi_{\text{base}}(\ell_{G'}(u')) \rangle.$$

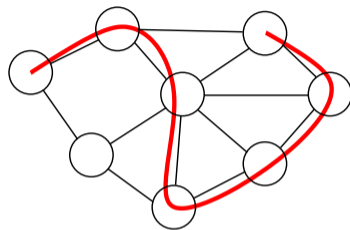
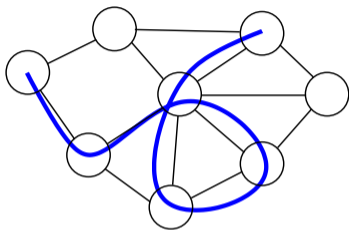
Graph kernel mappings



- Map each graph G in \mathcal{X} to a vector $\Phi(G)$ in \mathcal{H} , which lends itself to learning tasks.
- A large class of graph kernel mappings can be written in the form

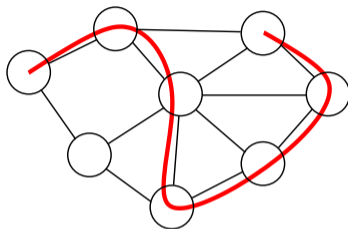
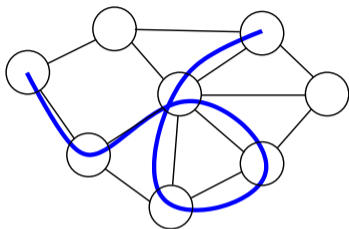
$$K(G, G') = \sum_{u \in \mathcal{V}} \sum_{u' \in \mathcal{V}'} \kappa_{\text{base}}(\ell_G(u), \ell_{G'}(u')).$$

Basic kernels: walk and path kernel mappings



- Path kernels are more **expressive** than walk kernels, but less preferred for **computational** reasons.

Basic kernels: walk and path kernel mappings

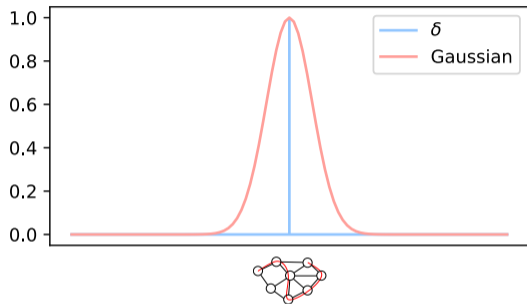


- $\mathcal{P}_k(G, u) :=$ paths of length k from node u in G . The k -**path** mapping is

$$\varphi_{\text{path}}(u) := \sum_{p \in \mathcal{P}_k(G, u)} \delta_{a(p)} \quad \Longrightarrow \quad \Phi(G) = \sum_{u \in \mathcal{V}} \sum_{p \in \mathcal{P}_k(G, u)} \delta_{a(p)}.$$

- $a(p)$: concatenated attributes in p ; δ : the Dirac function;
- $\Phi(G)$ can be interpreted as a **histogram** of paths occurrences;

A relaxed path kernel

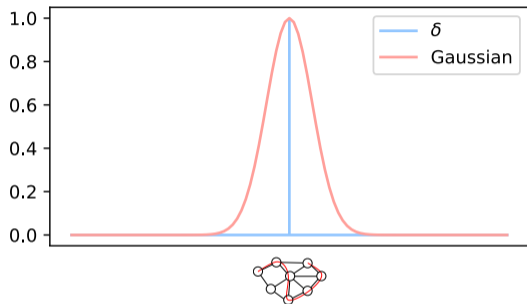


$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G, u)} \delta_{a(p)}(\cdot)$$

Issues of the path kernel mapping:

- δ allows hard comparison between paths thus only works for discrete attributes;
- δ is not differentiable, which cannot be “optimized” with back-propagation.

A relaxed path kernel



$$\begin{aligned}\varphi_{\text{path}}(u) &= \sum_{p \in \mathcal{P}_k(G, u)} \delta_{a(p)}(\cdot) \\ \implies & \sum_{p \in \mathcal{P}_k(G, u)} e^{-\frac{\alpha}{2} \|a(p) - \cdot\|^2}.\end{aligned}$$

Issues of the path kernel mapping:

- δ allows hard comparison between paths thus only works for discrete attributes;
- δ is not differentiable, which cannot be “optimized” with back-propagation.

Relax it with a “soft” and differentiable mapping

- interpreted as the sum of Gaussians centered at each path from u .

One-layer GCKN: a closer look at the relaxed path kernel

- We define the one-layer GCKN as the relaxed path kernel mapping

$$\varphi_1(u) := \sum_{p \in \mathcal{P}_k(G, u)} e^{-\frac{\alpha_1}{2} \|a(p) - \cdot\|^2} = \sum_{p \in \mathcal{P}_k(G, u)} \varphi_{\text{RBF}}(a(p)) \in \mathcal{H}_1.$$

- This formula can be divided into **3 steps**:
 - path extraction: enumerating all $\mathcal{P}_k(G, u)$;
 - kernel mapping: evaluating Gaussian embedding φ_{RBF} of path features;
 - path aggregation: aggregating the path embeddings.

One-layer GCKN: a closer look at the relaxed path kernel

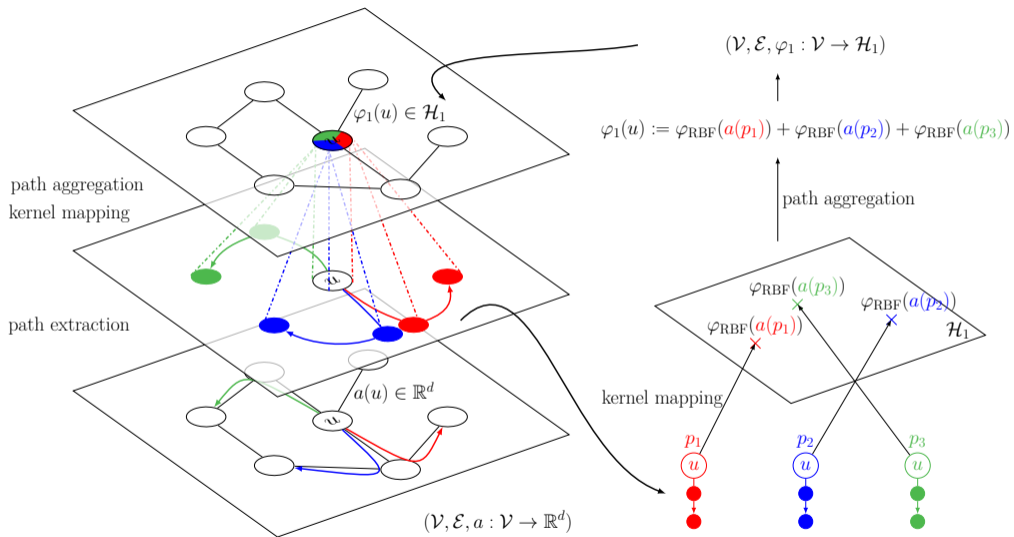
- We define the one-layer GCKN as the relaxed path kernel mapping

$$\varphi_1(u) := \sum_{p \in \mathcal{P}_k(G, u)} e^{-\frac{\alpha_1}{2} \|a(p) - \cdot\|^2} = \sum_{p \in \mathcal{P}_k(G, u)} \varphi_{\text{RBF}}(a(p)) \in \mathcal{H}_1.$$

- This formula can be divided into **3 steps**:
 - path extraction: enumerating all $\mathcal{P}_k(G, u)$;
 - kernel mapping: evaluating Gaussian embedding φ_{RBF} of path features;
 - path aggregation: aggregating the path embeddings.
- We obtain a new graph with the same topology but different features

$$(\mathcal{V}, \mathcal{E}, a) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_1).$$

Construction of one-layer GCKN



From one-layer to multilayer GCKN

- We can repeat applying φ_{path} to the new graph

$$(\mathcal{V}, \mathcal{E}, a) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_1) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_2) \xrightarrow{\varphi_{\text{path}}} \dots \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_j).$$

- Final graph representation at layer j , $\Phi(G) = \sum_{u \in \mathcal{V}} \varphi_j(u)$.

From one-layer to multilayer GCKN

- We can repeat applying φ_{path} to the new graph

$$(\mathcal{V}, \mathcal{E}, a) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_1) \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_2) \xrightarrow{\varphi_{\text{path}}} \dots \xrightarrow{\varphi_{\text{path}}} (\mathcal{V}, \mathcal{E}, \varphi_j).$$

- Final graph representation at layer j , $\Phi(G) = \sum_{u \in \mathcal{V}} \varphi_j(u)$.
- Why is the multilayer model interesting ?
 - applying φ_{path} once can capture **paths**: GCKN-path;
 - applying twice can capture **subtrees**: GCKN-subtree;
 - applying more times may capture **higher-order structures**?
 - **Long paths** cannot be enumerated due to computational complexity, yet multilayer model can capture **long-range substructures**.

Scalable approximation of Gaussian kernel mapping

$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G, u)} \varphi_{\text{RBF}}(a(p)).$$

- $\varphi_{\text{RBF}}(a(p)) = e^{-\frac{\alpha}{2} \|a(p) - \cdot\|^2} \in \mathcal{H}$ is infinite-dimensional;

[Chen et al., 2019a,b, Williams and Seeger, 2001]

Scalable approximation of Gaussian kernel mapping

$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G, u)} \varphi_{\text{RBF}}(a(p)).$$

- $\varphi_{\text{RBF}}(a(p)) = e^{-\frac{\alpha}{2} \|a(p) - \cdot\|^2} \in \mathcal{H}$ is infinite-dimensional;
- **Nystrom** provides a **finite-dimensional** approximation $\Psi(a(p))$ by orthogonally projecting $\varphi_{\text{RBF}}(a(p))$ onto some finite-dimensional subspace:

$\text{Span}(\varphi_{\text{RBF}}(z_1), \dots, \varphi_{\text{RBF}}(z_q))$ parametrized by $Z = \{z_1, \dots, z_q\}$,

where $z_j \in \mathbb{R}^{dk}$ can be interpreted as path features.

Scalable approximation of Gaussian kernel mapping

$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G, u)} \varphi_{\text{RBF}}(a(p)).$$

- $\varphi_{\text{RBF}}(a(p)) = e^{-\frac{\alpha}{2} \|a(p) - \cdot\|^2} \in \mathcal{H}$ is infinite-dimensional;
- **Nystrom** provides a **finite-dimensional** approximation $\Psi(a(p))$ by orthogonally projecting $\varphi_{\text{RBF}}(a(p))$ onto some finite-dimensional subspace:

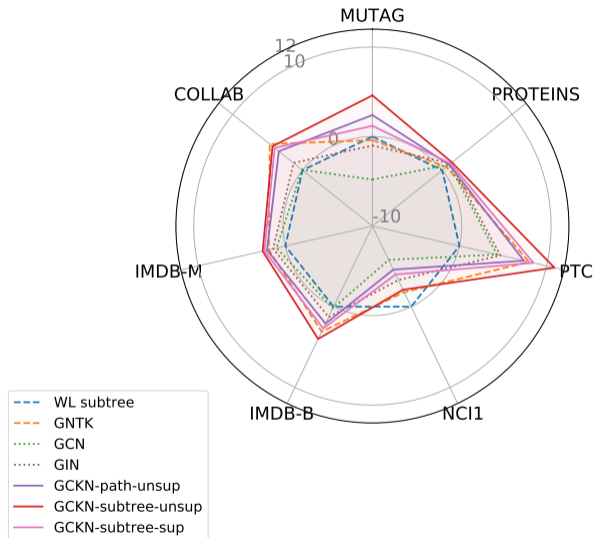
$\text{Span}(\varphi_{\text{RBF}}(z_1), \dots, \varphi_{\text{RBF}}(z_q))$ parametrized by $Z = \{z_1, \dots, z_q\}$,

where $z_j \in \mathbb{R}^{dk}$ can be interpreted as path features.

- The parameters Z can be learned by
 - (unsupervised) K-means on the set of path features;
 - (supervised) end-to-end learning with back-propagation.

[Chen et al., 2019a,b, Williams and Seeger, 2001]

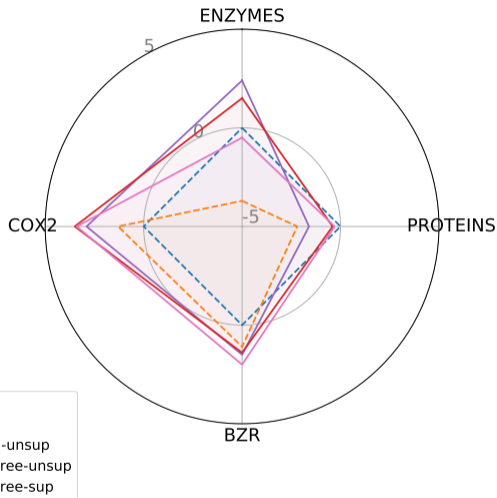
Experiments on graphs with discrete attributes



- Accuracy improvement with respect to the WL subtree kernel.
- GCKN-path already outperforms the baselines.
- Increasing number of layers brings larger improvement.
- Supervised learning does not improve performance, but leads to more compact representations.

[Shervashidze et al., 2011, Du et al., 2019, Xu et al., 2019, Kipf and Welling, 2017]

Experiments on graphs with continuous attributes

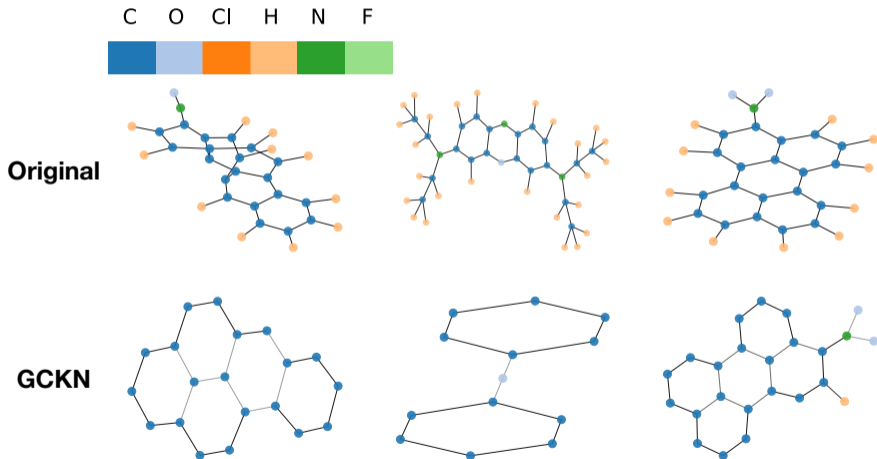


- Accuracy improvement with respect to the WWL kernel.
- Results similar to discrete case.
- Path features seem presumably predictive enough.

[Du et al., 2019, Togninalli et al., 2019]

Model interpretation for Mutagenicity prediction

- Idea: find the minimal connected component that preserves the prediction.



[Ying et al., 2019]

Take-home messages

- GCKN is a **multilayer kernel** for graphs based on **paths**, which allows to control the trade-off between **computation** and **expressiveness**.
- Its graph representations can be learned in both **supervised** and **unsupervised** fashions. Unsupervised models are **easy-to-regularize** and **scalable**.
- A straightforward model **interpretation** is also provided.
- **Our code is freely available at** <https://github.com/claying/GCKN>.

Take-home messages

- GCKN is a **multilayer kernel** for graphs based on **paths**, which allows to control the trade-off between **computation** and **expressiveness**.
- Its graph representations can be learned in both **supervised** and **unsupervised** fashions. Unsupervised models are **easy-to-regularize** and **scalable**.
- A straightforward model **interpretation** is also provided.
- **Our code is freely available at** <https://github.com/claying/GCKN>.

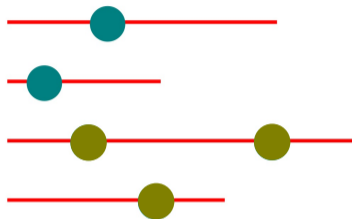
Future (on-going) work

- working with real people dealing with real data (protein folding prediction).

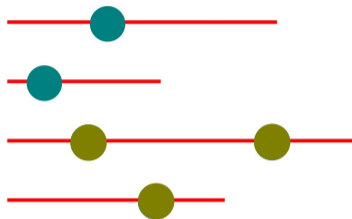
Biological Sequence Modeling

- D. Chen, L. Jacob and J. Mairal. Recurrent Kernel Networks. Adv. Neural Information Processing Systems (NeurIPS). 2019.
- D. Chen, L. Jacob and J. Mairal. Biological Sequence Modeling with Convolutional Kernel Networks. Bioinformatics. 2019.

Sequence modeling as a supervised learning problem



Sequence modeling as a supervised learning problem



- Biological sequences $x_1, \dots, x_n \in \mathcal{X}$ and their associated labels y_1, \dots, y_n .
- Goal: learning a **predictive** and **interpretable** function $f : \mathcal{X} \rightarrow \mathbb{R}$

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\mu \Omega(f)}_{\text{regularization}} .$$

- How do we define the functional space \mathcal{F} ?

String kernels

A classical approach for modeling biological sequences over alphabet \mathcal{A} relies on string kernels.

$$K(x, x') = \sum_{u \in \mathcal{A}^k} \delta_u(x) \delta_u(x'),$$

where u is a k -mer over an alphabet \mathcal{A} and $\delta_u(x)$ can be:

- the number of occurrences of u in x : **spectrum kernel** [Leslie et al., 2002];
- the number of occurrences of u in x up to m mismatches: **mismatch kernel** [Leslie and Kuang, 2004];
- the number of occurrences of u in x allowing gaps, with a weight decaying exponentially with the number of gaps : **substring kernel** [Lodhi et al., 2002].

What is $\Phi(x)$?

It can be interpreted as a histogram of pattern occurrences.

String kernels

A classical approach for modeling biological sequences over alphabet \mathcal{A} relies on string kernels.

$$K(x, x') = \sum_{u \in \mathcal{A}^k} \delta_u(x) \delta_u(x') = \langle \Phi(x), \Phi(x') \rangle,$$

where u is a k -mer over an alphabet \mathcal{A} and $\delta_u(x)$ can be:

- the number of occurrences of u in x : **spectrum kernel** [Leslie et al., 2002];
- the number of occurrences of u in x up to m mismatches: **mismatch kernel** [Leslie and Kuang, 2004];
- the number of occurrences of u in x allowing gaps, with a weight decaying exponentially with the number of gaps : **substring kernel** [Lodhi et al., 2002].

What is $\Phi(x)$?

It can be interpreted as a histogram of pattern occurrences.

Convolutional kernel networks for sequence modeling

Define a continuous relaxation of the mismatch kernel [Chen et al., 2019a, Morrow et al., 2017]

$$K_{\text{CKN}}(x, x') = \sum_{i=1}^{|x|-k+1} \sum_{j=1}^{|x'|-k+1} K_0(\underbrace{x_{[i:i+k]}, x'_{[j:j+k]}}_{\text{one k-mer}}).$$

- Use one-hot encoding

$$x_{[i:i+5]} := \text{TTGAG} \mapsto \begin{matrix} \text{A} \\ \text{T} \\ \text{C} \\ \text{G} \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

- K_0 is a Gaussian kernel over **one-hot** representations of k-mers (in $\mathbb{R}^{k \times d}$).

Convolutional kernel networks for sequence modeling

Define a continuous relaxation of the mismatch kernel [Chen et al., 2019a, Morrow et al., 2017]

$$K_{\text{CKN}}(x, x') = \sum_{i=1}^{|x|-k+1} \sum_{j=1}^{|x'|-k+1} \langle \varphi_0(\underbrace{x_{[i:i+k]}}_{\text{one k-mer}}), \varphi_0(x'_{[j:j+k]}) \rangle.$$

- Use one-hot encoding

$$x_{[i:i+5]} := \text{TTGAG} \mapsto \begin{matrix} \text{A} \\ \text{T} \\ \text{C} \\ \text{G} \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

- K_0 is a Gaussian kernel over **one-hot** representations of k-mers (in $\mathbb{R}^{k \times d}$).

Convolutional kernel networks for sequence modeling

Define a continuous relaxation of the mismatch kernel [Chen et al., 2019a, Morrow et al., 2017]

$$K_{\text{CKN}}(x, x') = \left\langle \underbrace{\sum_{i=1}^{|x|-k+1} \varphi_0(x_{[i:i+k]})}_{\Phi(x)}, \underbrace{\sum_{j=1}^{|x'|-k+1} \varphi_0(x'_{[j:j+k]})}_{\Phi(x')} \right\rangle.$$

- Use one-hot encoding

$$x_{[i:i+5]} := \text{TTGAG} \mapsto \begin{matrix} \text{A} \\ \text{T} \\ \text{C} \\ \text{G} \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

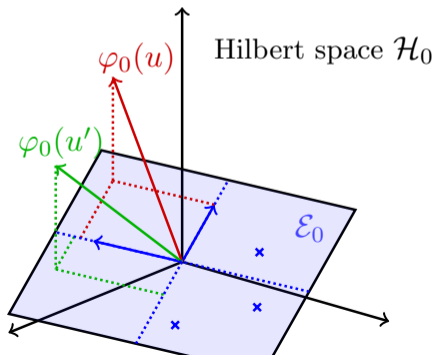
- K_0 is a Gaussian kernel over **one-hot** representations of k-mers (in $\mathbb{R}^{k \times d}$).

Scalable Approximation of Kernel Mapping (with more details this time)

$$K_0(u, u') = \langle \varphi_0(u), \varphi_0(u') \rangle_{\mathcal{H}_0} \approx \langle \psi_0(u), \psi_0(u') \rangle_{\mathbb{R}^q}.$$

- **Nyström** provides a **finite-dimensional** approximation $\psi_0(u)$ in \mathbb{R}^q by orthogonally projecting $\varphi_0(u)$ onto some finite-dimensional subspace:

$$\mathcal{E}_0 = \text{Span}(\varphi_0(z_1), \dots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \dots, z_q\}.$$



Scalable Approximation of Kernel Mapping (with more details this time)

$$K_0(u, u') = \langle \varphi_0(u), \varphi_0(u') \rangle_{\mathcal{H}_0} \approx \langle \psi_0(u), \psi_0(u') \rangle_{\mathbb{R}^q}.$$

- **Nyström** provides a **finite-dimensional** approximation $\psi_0(u)$ in \mathbb{R}^q by orthogonally projecting $\varphi_0(u)$ onto some finite-dimensional subspace:

$$\mathcal{E}_0 = \text{Span}(\varphi_0(z_1), \dots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \dots, z_q\}.$$

- **General case:**

$$\psi_0(u) = [K_0(z_i, z_j)]_{ij}^{-1/2} [K_0(z_1, u), \dots, K_0(z_q, u)]^T = K_0(Z, Z)^{-1/2} K_0(Z, u).$$

Scalable Approximation of Kernel Mapping (with more details this time)

$$K_0(u, u') = \langle \varphi_0(u), \varphi_0(u') \rangle_{\mathcal{H}_0} \approx \langle \psi_0(u), \psi_0(u') \rangle_{\mathbb{R}^q}.$$

- **Nyström** provides a **finite-dimensional** approximation $\psi_0(u)$ in \mathbb{R}^q by orthogonally projecting $\varphi_0(u)$ onto some finite-dimensional subspace:

$$\mathcal{E}_0 = \text{Span}(\varphi_0(z_1), \dots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \dots, z_q\}.$$

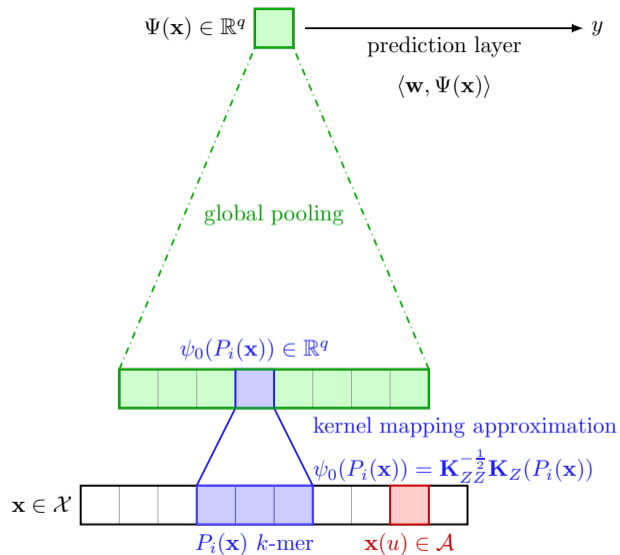
- Case of **dot-product kernels** $K_0(u, u') = \kappa(\langle u, u' \rangle)$:

$$\psi_0(u) = \kappa(Z^\top Z)^{-1/2} \kappa(Z^\top u).$$

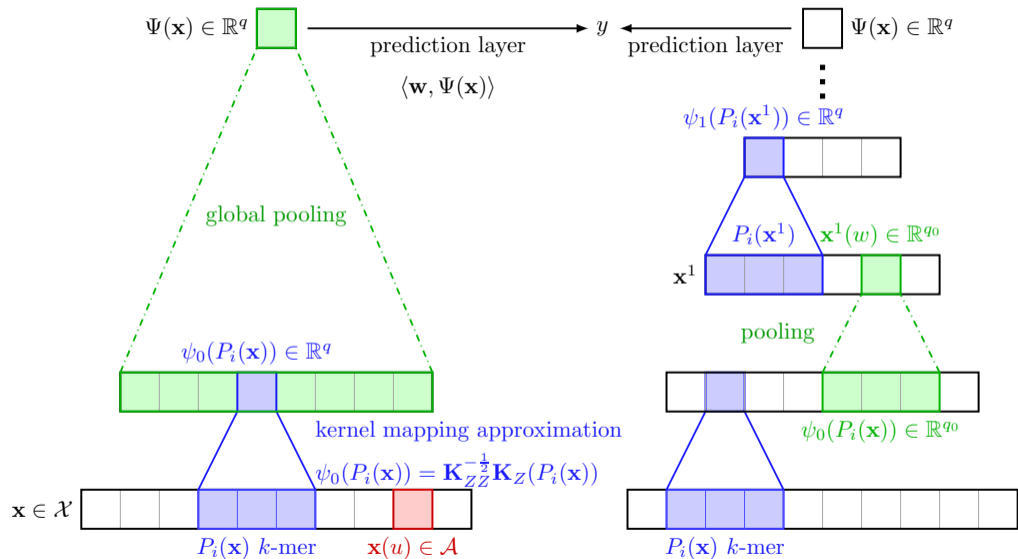
linear operation - pointwise nonlinearity - linear operation (subject to interpretation)

Ex: $\kappa(\beta) = e^{\beta-1}$, polynomial, inverse polynomial, arc-cosine kernels....

Single-Layer CKN for sequence modeling



Multilayer CKN for sequence modeling



How to learn the anchor points Z ?

with no supervision?

we learn one layer at a time, starting from the bottom one.

- we **extract a large number**—say 100 000 k-mers from the previous layer computed on a sequence database;
- perform a **K-means algorithm** to learn the anchor points;
- **compute the projection matrix** $\kappa(Z^\top Z)^{-1/2}$ (case of a dot-product kernel).

How to learn the anchor points Z ?

with no supervision?

we learn one layer at a time, starting from the bottom one.

- we **extract a large number**—say 100 000 k-mers from the previous layer computed on a sequence database;
- perform a **K-means algorithm** to learn the anchor points;
- **compute the projection matrix** $\kappa(Z^\top Z)^{-1/2}$ (case of a dot-product kernel).

with supervision?

- by using **back-propagation** on a supervised loss function;
- all it requires is differentiating $\kappa(Z^\top Z)^{-1/2}$ which requires an eigenvalue decomposition;
- use the unsupervised learning procedure as initialization.

From k-mers to gapped k-mers

k-mers with gaps

- For a sequence $x = x_1 \dots x_n \in \mathcal{X}$ of length n and a sequence of ordered indices $\mathbf{i} = (i_1, \dots, i_k)$ in $I(k, n)$, we define a k-substring as:

$$x[\mathbf{i}] = x_{i_1} x_{i_2} \dots x_{i_k}.$$

- We introduce the quantity

$$\text{gaps}(\mathbf{i}) = \text{number of gaps in index sequence.}$$

- Example: $x = \text{ABRACADABRA}$

$$\mathbf{i} = (4, 5, 8, 9, 11) \quad x[\mathbf{i}] = \text{RADAR} \quad \text{gaps}(\mathbf{i}) = 3.$$

Recurrent kernel networks

Comparing all the k-mers between a pair of sequences (single layer models)

$$K_{\text{CKN}}(x, x') = \sum_{i=1}^{|x|-k+1} \sum_{j=1}^{|x'|-k+1} K_0 \left(x_{[i:i+k]}, x'_{[j:j+k]} \right).$$

- The kernel mapping is $\Phi(x) = \sum_{i=1}^{|x|-k+1} \varphi_0(x_{[i:i+k]})$.

Recurrent kernel networks

Comparing all the **gapped** k-mers between a pair of sequences (single layer models)

$$K_{\text{RKN}}(x, x') = \sum_{\mathbf{i} \in I(k, |x|)} \sum_{\mathbf{j} \in I(k, |x'|)} \lambda^{\text{gaps}(\mathbf{i})} \lambda^{\text{gaps}(\mathbf{j})} K_0(x_{[\mathbf{i}]}, x'_{[\mathbf{j}]}) .$$

- The kernel mapping is $\Phi(x) = \sum_{\mathbf{i} \in I(k, |x|)} \lambda^{\text{gaps}(\mathbf{i})} \varphi_0(x_{[\mathbf{i}]})$.
- This is a differentiable relaxation of the substring kernel.

But enumerating all possible substrings is costly...

Approximation and recursive computation of RKN

Approximate feature map of RKN kernel

The approximate feature map of K_{RKN} via Nyström approximation is

$$\Psi(x) = \sum_{\mathbf{i} \in I(k,t)} \lambda^{\text{gaps}(\mathbf{i})} \psi_0(x_{[\mathbf{i}]}) \in \mathbb{R}^q,$$

where, as usual with a dot-product kernel, $\psi_0(x_{[\mathbf{i}]}) = \kappa(Z^\top Z)^{-1/2} \kappa(Z^\top x_{[\mathbf{i}]})$.

- The sum can be computed by using **dynamic programming** [Lodhi et al., 2002],
- which leads to a **particular** recurrent neural network [see Lei et al., 2017].

A feature map for the single-layer RKN

When K_0 is a Gaussian kernel, the feature map of RKN is a mixture of Gaussians centered at $x_{[i]}$, weighted by the corresponding penalization $\lambda^{\text{gap}(i)}$.

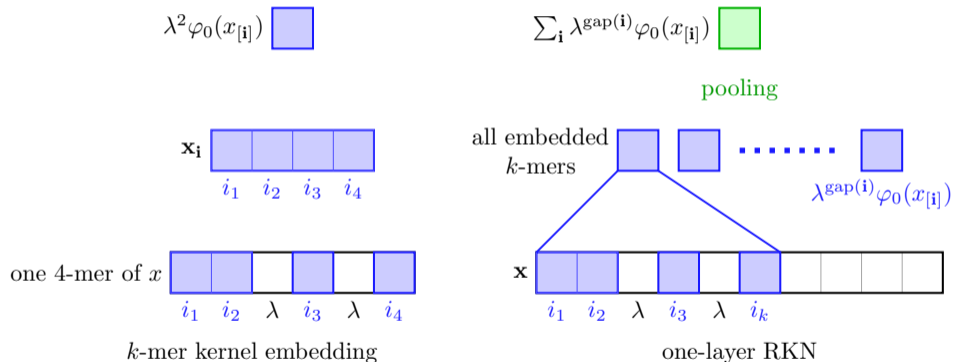


Figure: Example of K_{RKN} for $k = 4$

Results

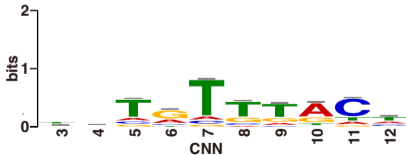
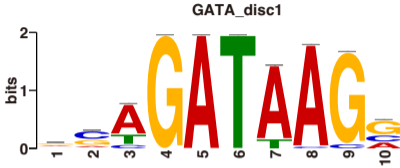
Protein fold classification on SCOP 2.06 [Hou et al., 2017] (using more informative sequence features including PSSM, secondary structure and solvent accessibility)

Method	#Params	Accuracy		Level-stratified accuracy (top1/top5)		
		top 1	top 5	family	superfamily	fold
PSI-BLAST	-	84.53	86.48	82.20/84.50	86.90/88.40	18.90/35.100
DeepSF	920k	73.00	90.25	75.87/91.77	72.23/90.08	51.35/67.57
CKN (128 filters)	211k	76.30	92.17	83.30/94.22	74.03/91.83	43.78/67.03
CKN (512 filters)	843k	84.11	94.29	90.24/95.77	82.33/94.20	45.41/69.19
RKN (128 filters)	211k	77.82	92.89	76.91/93.13	78.56/92.98	60.54/83.78
RKN (512 filters)	843k	85.29	94.95	84.31/94.80	85.99/95.22	71.35/84.86

Note: More experiments with statistical tests have been conducted in our paper.

[Hou et al., 2017, Chen et al., 2019a]

Logos, by finding pre-image of each filter



Results

Protein fold recognition on SCOP 1.67 (widely used in the past)

Method	pooling	one-hot		BLOSUM62	
		auROC	auROC50	auROC	auROC50
SVM-pairwise		0.724	0.359		
Mismatch		0.814	0.467		
LA-kernel		–	–	0.834	0.504
LSTM		0.830	0.566	–	–
CKN		0.837	0.572	0.866	0.621
RKN	mean	0.829	0.541	0.840	0.571
RKN	max	0.844	0.587	0.871	0.629
RKN (unsup)	mean	0.805	0.504	0.833	0.570

[Liao and Noble, 2003, Leslie et al., 2003, Vert et al., 2004, Hochreiter et al., 2007, Chen et al., 2019a]

Take-home messages

- CKN and RKNs are **multilayer kernels** for sequences, achieving state-of-the-art results for biological sequence modeling (see other tasks in papers).
- RKN is able to model gaps with a recurrent neural network structure.
- These models can be used without supervision, providing effective, but **high-dimensional** embeddings.
- With supervision, models trained with backpropagation are much more compact.
- For biological sequences, best results were obtained with a single layer.

Take-home messages

- CKN and RKNs are **multilayer kernels** for sequences, achieving state-of-the-art results for biological sequence modeling (see other tasks in papers).
- RKN is able to model gaps with a recurrent neural network structure.
- These models can be used without supervision, providing effective, but **high-dimensional** embeddings.
- With supervision, models trained with backpropagation are much more compact.
- For biological sequences, best results were obtained with a single layer.

Our code in Pytorch is freely available at
<https://gitlab.inria.fr/dchen/CKN-seq>
<https://github.com/claying/RKN>

Image Modeling

- J. Mairal. End-to-End Kernel Learning with Supervised Convolutional Kernel Networks. Adv. Neural Information Processing Systems (NIPS), 2016.
- J. Mairal, P. Koniusz, Z. Harchaoui and C. Schmid. Convolutional Kernel Networks. Adv. Neural Information Processing Systems (NIPS). 2014.

Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location $(d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location $(d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

Building map x_k in $L^2(\Omega, \mathcal{H}_k)$ from x_{k-1} in $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \rightarrow \mathcal{H}_k$: **feature map** at layer k

$$P_k x_{k-1}.$$

- P_k : **patch extraction** operator, extract small patch of feature map x_{k-1} around each point u ($P_k x_{k-1}(u)$ is a patch centered at u).

Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location $(d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

Building map x_k in $L^2(\Omega, \mathcal{H}_k)$ from x_{k-1} in $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \rightarrow \mathcal{H}_k$: **feature map** at layer k

$$M_k P_k x_{k-1}.$$

- P_k : **patch extraction** operator, extract small patch of feature map x_{k-1} around each point u ($P_k x_{k-1}(u)$ is a patch centered at u).
- M_k : **non-linear mapping** operator, maps each patch to a new Hilbert space \mathcal{H}_k with a **pointwise** non-linear function $\varphi_k(\cdot)$.

Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location $(d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

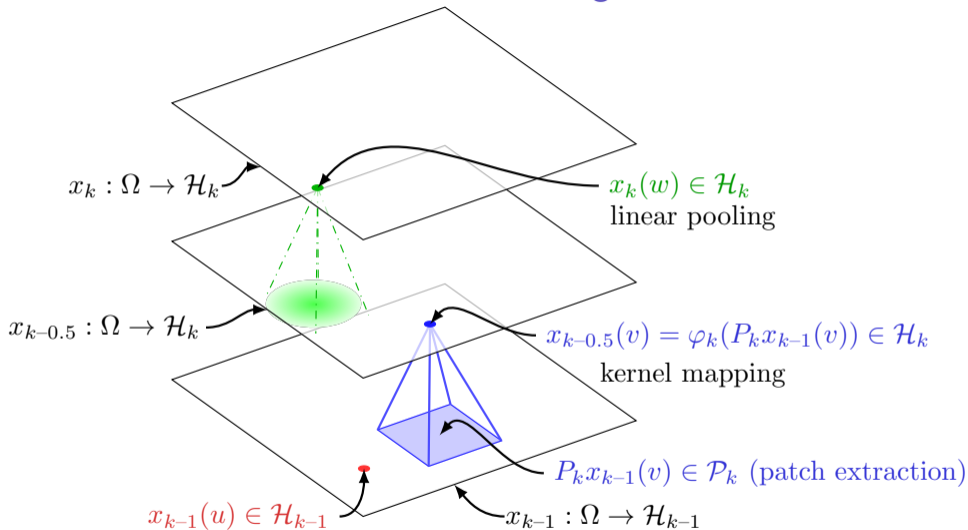
Building map x_k in $L^2(\Omega, \mathcal{H}_k)$ from x_{k-1} in $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \rightarrow \mathcal{H}_k$: **feature map** at layer k

$$x_k = A_k M_k P_k x_{k-1}.$$

- P_k : **patch extraction** operator, extract small patch of feature map x_{k-1} around each point u ($P_k x_{k-1}(u)$ is a patch centered at u).
- M_k : **non-linear mapping** operator, maps each patch to a new Hilbert space \mathcal{H}_k with a **pointwise** non-linear function $\varphi_k(\cdot)$.
- A_k : (linear) **pooling** operator at scale σ_k .

Construction of the RKHS for continuous signals



Construction of the RKHS for continuous signals

Kernel mapping for patches

- We use a homogeneous dot-product kernel for image patches

$$K(z, z') = \|z\| \|z'\| \kappa \left(\frac{\langle z, z' \rangle}{\|z\| \|z'\|} \right).$$

Multilayer representation

$$\Phi_n(x) = A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 x_0 \in L^2(\Omega, \mathcal{H}_n).$$

- σ_k grows exponentially in practice (i.e., fixed with subsampling).

Construction of the RKHS for continuous signals

Kernel mapping for patches

- We use a homogeneous dot-product kernel for image patches

$$K(z, z') = \|z\| \|z'\| \kappa \left(\frac{\langle z, z' \rangle}{\|z\| \|z'\|} \right).$$

Multilayer representation

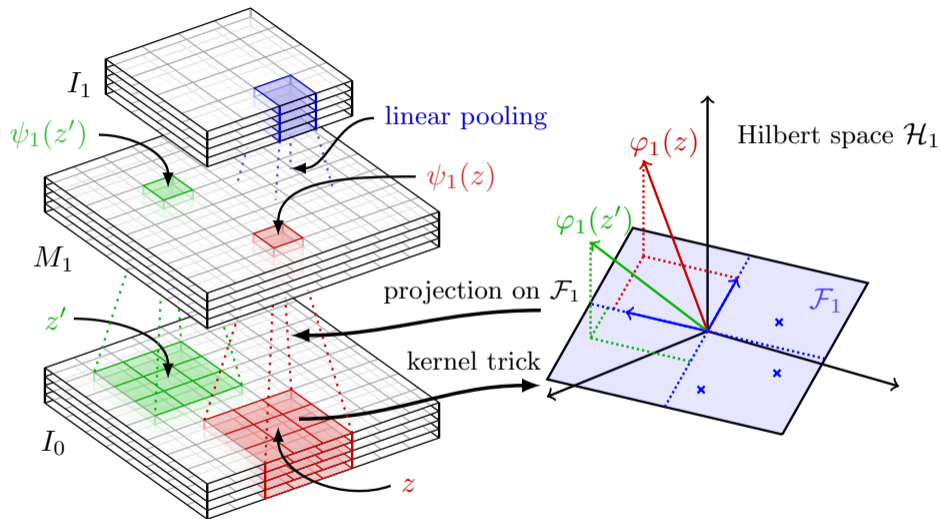
$$\Phi_n(x) = A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 x_0 \in L^2(\Omega, \mathcal{H}_n).$$

- σ_k grows exponentially in practice (i.e., fixed with subsampling).

Prediction layer

- e.g., linear $f(x) = \langle w, \Phi_n(x) \rangle$.
- “linear kernel” $\mathcal{K}(x, x') = \langle \Phi_n(x), \Phi_n(x') \rangle = \int_{\Omega} \langle x_n(u), x'_n(u) \rangle du$.

Convolutional Kernel Networks in practice



Learning mechanism of CKNs between layers 0 and 1.

Convolutional Kernel Networks in Practice

What is the difference with a CNN?

- Given a patch x , a CNN computes $\psi_{CNN}(x) = \sigma(Z^\top x)$ (+batch norm?)
- Given a patch x , a CKN computes $\psi_{CKN}(x) = \|x\| \kappa(Z^\top Z)^{-1/2} \kappa(Z^\top x / \|x\|)$.

Convolutional Kernel Networks in Practice

What is the difference with a CNN?

- Given a patch x , a CNN computes $\psi_{CNN}(x) = \sigma(Z^\top x)$ (+batch norm?)
- Given a patch x , a CKN computes $\psi_{CKN}(x) = \|x\| \kappa(Z^\top Z)^{-1/2} \kappa(Z^\top x / \|x\|)$.

Consequences

- we have a geometric interpretation in terms of subspace learning.
- it provides unsupervised learning mechanisms (kernel approximation with Nyström).
- supervised learning is still feasible (backpropagating through $\kappa(Z^\top Z)^{-1/2}$ is fun).
- the kernel interpretation provides regularization mechanisms.
- kernel representations can possibly be used in other contexts (statistical testing? kernel PCA? CCA? K-means?).

Experiments

- Briefly state-of-the-art for image retrieval [Paulin et al., 2015];
- Briefly state-of-the-art for image super-resolution [Mairal, 2016a];

Interesting findings from CIFAR-10

- about 92% with supervision, mild data augmentation, 14 layers, 256 anchor points per layers (no need for batch norm, vanilla SGD+momentum).
- about 86% **with no supervision** for a two-layer model with a huge number of anchor points (1024-16384) and no data augmentation.
- with no supervision, **the performance monotonically increases with the dimension** (better kernel approximation).
- computing the exact kernel does not make sense in practice for computational reasons, but it is feasible with lots of CPUs; it yields about 90% with three layers (unpublished, by A. Bietti), which is consistent with [Shankar et al., 2020].

Take-home messages

- unsupervised representations are shallow and high-dimensional;
- supervised representations may be deep and compact;
- **Our code is freely available at**
`https://gitlab.inria.fr/mairal/ckn-cudnn-matlab`.
- **and** `https://github.com/claying/CKN-Pytorch-image`.

Take-home messages

- unsupervised representations are shallow and high-dimensional;
- supervised representations may be deep and compact;
- **Our code is freely available at**
`https://gitlab.inria.fr/mairal/ckn-cudnn-matlab`.
- **and** `https://github.com/claying/CKN-Pytorch-image`.

Open

- how to close the gap between the approximate embedding and the exact kernel?

Theory for Deep Learning Models

- A. Bietti and J. Mairal. On the Inductive Bias of Neural Tangent Kernels. Adv. Neural Information Processing Systems (NeurIPS). 2019.
- A. Bietti and J. Mairal. Group Invariance, Stability to Deformations, and Complexity of Deep Convolutional Representations. Journal of Machine Learning Research (JMLR). 2019.

Kernels for deep models: deep kernel machines

Hierarchical kernels [Cho and Saul, 2009]

- Kernels can be constructed **hierarchically**

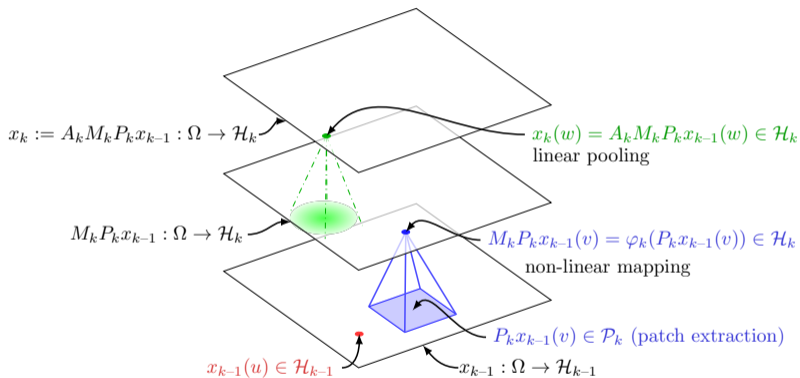
$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle \text{ with } \Phi(x) = \varphi_2(\varphi_1(x))$$

- e.g., dot-product kernels on the sphere

$$K(x, x') = \kappa_2(\langle \varphi_1(x), \varphi_1(x') \rangle) = \kappa_2(\kappa_1(x^\top x'))$$

Kernels for deep models: deep kernel machines

Convolutional kernels networks (CKNs) for images [Mairal et al., 2014, Mairal, 2016b]



- Good empirical performance with tractable approximations (Nyström)

Kernels for deep models: infinite-width networks

$$f_{\theta}(x) = \frac{1}{\sqrt{m}} \sum_{i=1}^m v_i \sigma(w_i^{\top} x), \quad m \rightarrow \infty$$

Random feature kernels [RF, Neal, 1996, Rahimi and Recht, 2007]

- $\theta = (v_i)_i$, fixed random weights $w_i \sim N(0, I)$

$$K_{RF}(x, x') = \mathbb{E}_{w \sim N(0, I)} [\sigma(w^{\top} x) \sigma(w^{\top} x')]$$

Kernels for deep models: infinite-width networks

$$f_{\theta}(x) = \frac{1}{\sqrt{m}} \sum_{i=1}^m v_i \sigma(w_i^{\top} x), \quad m \rightarrow \infty$$

Random feature kernels [RF, Neal, 1996, Rahimi and Recht, 2007]

- $\theta = (v_i)_i$, fixed random weights $w_i \sim N(0, I)$

$$K_{RF}(x, x') = \mathbb{E}_{w \sim N(0, I)} [\sigma(w^{\top} x) \sigma(w^{\top} x')]$$

Neural tangent kernels [NTK, Jacot et al., 2018]

- $\theta = (v_i, w_i)_i$, initialization $\theta_0 \sim N(0, I)$
- **Lazy training** [Chizat et al., 2019]: θ stays close to θ_0 when training with large m

$$f_{\theta}(x) \approx f_{\theta_0}(x) + \langle \theta - \theta_0, \nabla_{\theta} f_{\theta}(x) |_{\theta=\theta_0} \rangle.$$

Kernels for deep models: infinite-width networks

$$f_{\theta}(x) = \frac{1}{\sqrt{m}} \sum_{i=1}^m v_i \sigma(w_i^{\top} x), \quad m \rightarrow \infty$$

Random feature kernels [RF, Neal, 1996, Rahimi and Recht, 2007]

- $\theta = (v_i)_i$, fixed random weights $w_i \sim N(0, I)$

$$K_{RF}(x, x') = \mathbb{E}_{w \sim N(0, I)} [\sigma(w^{\top} x) \sigma(w^{\top} x')]$$

Neural tangent kernels [NTK, Jacot et al., 2018]

- $\theta = (v_i, w_i)_i$, initialization $\theta_0 \sim N(0, I)$
- **Lazy training** [Chizat et al., 2019]: θ stays close to θ_0 when training with large m

$$f_{\theta}(x) \approx f_{\theta_0}(x) + \langle \theta - \theta_0, \nabla_{\theta} f_{\theta}(x) |_{\theta=\theta_0} \rangle.$$

- Gradient descent for $m \rightarrow \infty \approx$ kernel ridge regression with **neural tangent kernel**

$$K_{NTK}(x, x') = \lim_{m \rightarrow \infty} \langle \nabla_{\theta} f_{\theta_0}(x), \nabla_{\theta} f_{\theta_0}(x') \rangle$$

Other relations between kernels and deep learning

- hierarchical kernel descriptors [Bo et al., 2011];
- other multilayer models [Bouvrie et al., 2009, Montavon et al., 2011, Anselmi et al., 2015];
- deep Gaussian processes [Damianou and Lawrence, 2013].
- multilayer PCA [Schölkopf et al., 1998].
- old kernels for images [Scholkopf, 1997], related to one-layer CKN.
- RBF networks [Broomhead and Lowe, 1988].
- ...

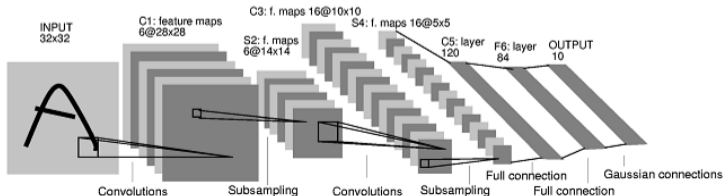
Objectives

Deep convolutional signal representations

- Are they **stable to deformations**?
- How can we achieve **invariance to transformation groups**?
- Do they **preserve signal information**?

Learning aspects

- Building a **functional space** for CNNs (or similar objects).
- Deriving a measure of **model complexity**.



Focus on convolutional kernel networks (CKNs)

What is the relation?

- it is possible to design functional spaces \mathcal{H} for deep neural networks [Mairal, 2016b].

$$f(x) = \sigma_k(W_k \sigma_{k-1}(W_{k-1} \dots \sigma_2(W_2 \sigma_1(W_1 x)) \dots)) = \langle f, \Phi(x) \rangle_{\mathcal{H}}.$$

- we call the construction “**convolutional kernel networks**” (in short, replace $u \mapsto \sigma(\langle a, u \rangle)$ by a kernel mapping $u \mapsto \varphi_k(u)$).

Why do we care?

- $\Phi(x)$ is related to the **network architecture** and is **independent of training data**. Is it stable? Does it lose signal information?

Focus on convolutional kernel networks (CKNs)

What is the relation?

- it is possible to design functional spaces \mathcal{H} for deep neural networks [Mairal, 2016b].

$$f(x) = \sigma_k(W_k \sigma_{k-1}(W_{k-1} \dots \sigma_2(W_2 \sigma_1(W_1 x)) \dots)) = \langle f, \Phi(x) \rangle_{\mathcal{H}}.$$

- we call the construction “**convolutional kernel networks**” (in short, replace $u \mapsto \sigma(\langle a, u \rangle)$ by a kernel mapping $u \mapsto \varphi_k(u)$).

Why do we care?

- $\Phi(x)$ is related to the **network architecture** and is **independent of training data**. Is it stable? Does it lose signal information?
- f is a **predictive model**. Can we control its stability?

$$|f(x) - f(x')| \leq \|f\|_{\mathcal{H}} \|\Phi(x) - \Phi(x')\|_{\mathcal{H}}.$$

Summary of the results from Bietti and Mairal [2019a]

Multi-layer construction of the RKHS \mathcal{H}

- Contains CNNs with smooth homogeneous activations functions.

Summary of the results from Bietti and Mairal [2019a]

Multi-layer construction of the RKHS \mathcal{H}

- Contains CNNs with smooth homogeneous activations functions.

Signal representation: Conditions for

- **Signal preservation** of the multi-layer kernel mapping Φ .
- **Stability to deformations and non-expansiveness** for Φ .
- Constructions to achieve **group invariance**.

Summary of the results from Bietti and Mairal [2019a]

Multi-layer construction of the RKHS \mathcal{H}

- Contains CNNs with smooth homogeneous activations functions.

Signal representation: Conditions for

- **Signal preservation** of the multi-layer kernel mapping Φ .
- **Stability to deformations and non-expansiveness** for Φ .
- Constructions to achieve **group invariance**.

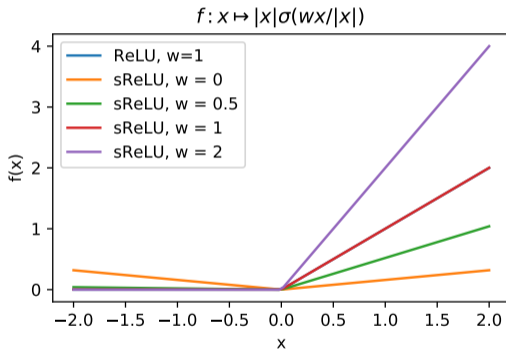
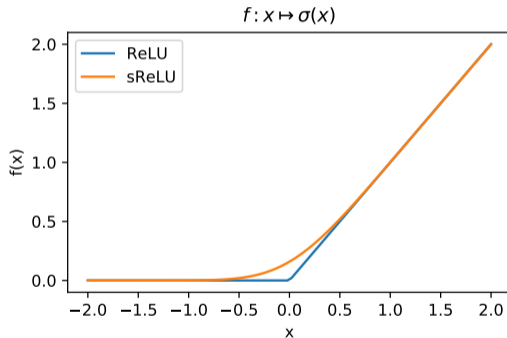
On learning

- Bounds on the RKHS norm $\|\cdot\|_{\mathcal{H}}$ to control **stability and generalization** of a predictive model f .

$$|f(x) - f(x')| \leq \|f\|_{\mathcal{H}} \|\Phi(x) - \Phi(x')\|_{\mathcal{H}}.$$

Smooth homogeneous activations functions

$$z \mapsto \text{ReLU}(w^\top z) \quad \implies \quad z \mapsto \|z\| \sigma(w^\top z / \|z\|).$$



Recap: Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location ($d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

Building map x_k in $L^2(\Omega, \mathcal{H}_k)$ from x_{k-1} in $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \rightarrow \mathcal{H}_k$: **feature map** at layer k

- P_k : **patch extraction** operator, extract small patch of feature map x_{k-1} around each point u ($P_k x_{k-1}(u)$ is a patch centered at u).
- M_k : **non-linear mapping** operator, maps each patch to a new Hilbert space \mathcal{H}_k with a **pointwise** non-linear function $\varphi_k(\cdot)$.
- A_k : (linear) **pooling** operator at scale σ_k .

Recap: Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location ($d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

Building map x_k in $L^2(\Omega, \mathcal{H}_k)$ from x_{k-1} in $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \rightarrow \mathcal{H}_k$: **feature map** at layer k

$$P_k x_{k-1}.$$

- P_k : **patch extraction** operator, extract small patch of feature map x_{k-1} around each point u ($P_k x_{k-1}(u)$ is a patch centered at u).
- M_k : **non-linear mapping** operator, maps each patch to a new Hilbert space \mathcal{H}_k with a **pointwise** non-linear function $\varphi_k(\cdot)$.
- A_k : (linear) **pooling** operator at scale σ_k .

Recap: Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location ($d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

Building map x_k in $L^2(\Omega, \mathcal{H}_k)$ from x_{k-1} in $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \rightarrow \mathcal{H}_k$: **feature map** at layer k

$$M_k P_k x_{k-1}.$$

- P_k : **patch extraction** operator, extract small patch of feature map x_{k-1} around each point u ($P_k x_{k-1}(u)$ is a patch centered at u).
- M_k : **non-linear mapping** operator, maps each patch to a new Hilbert space \mathcal{H}_k with a **pointwise** non-linear function $\varphi_k(\cdot)$.
- A_k : (linear) **pooling** operator at scale σ_k .

Recap: Construction of the RKHS for continuous signals

Initial map x_0 in $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \rightarrow \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location ($d = 2$ for images).
• $x_0(u) \in \mathcal{H}_0$: input value at location u ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

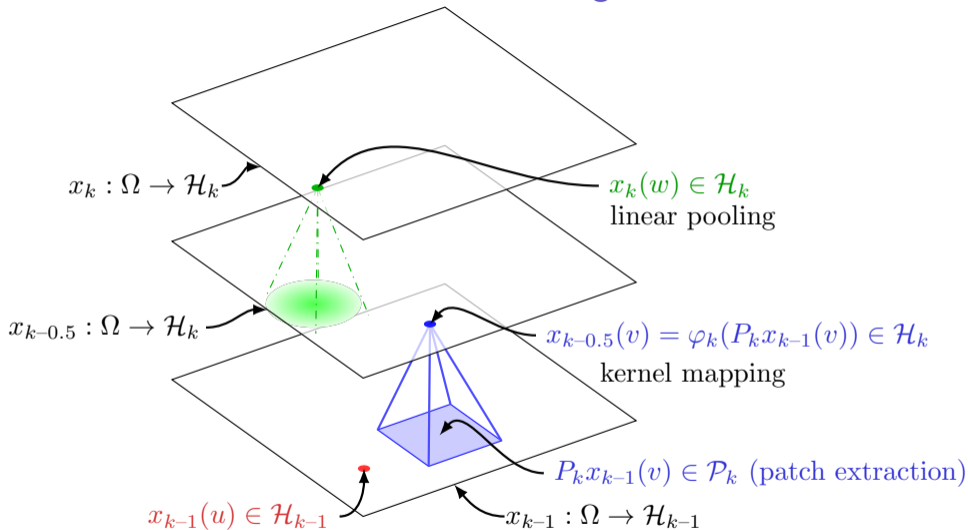
Building map x_k in $L^2(\Omega, \mathcal{H}_k)$ from x_{k-1} in $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \rightarrow \mathcal{H}_k$: **feature map** at layer k

$$x_k = A_k M_k P_k x_{k-1}.$$

- P_k : **patch extraction** operator, extract small patch of feature map x_{k-1} around each point u ($P_k x_{k-1}(u)$ is a patch centered at u).
- M_k : **non-linear mapping** operator, maps each patch to a new Hilbert space \mathcal{H}_k with a **pointwise** non-linear function $\varphi_k(\cdot)$.
- A_k : (linear) **pooling** operator at scale σ_k .

Construction of the RKHS for continuous signals



Construction of the RKHS for continuous signals

Multilayer representation

$$\Phi_n(x) = A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 x_0 \in L^2(\Omega, \mathcal{H}_n).$$

- σ_k grows exponentially in practice (i.e., fixed with subsampling).

Construction of the RKHS for continuous signals

Multilayer representation

$$\Phi_n(x) = A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 x_0 \in L^2(\Omega, \mathcal{H}_n).$$

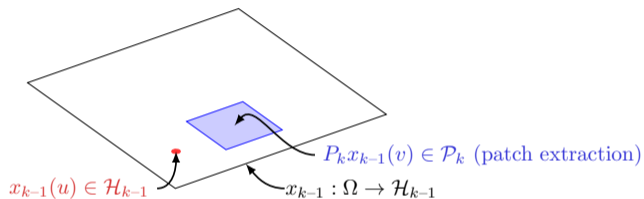
- σ_k grows exponentially in practice (i.e., fixed with subsampling).

Prediction layer

- e.g., linear $f(x) = \langle w, \Phi_n(x) \rangle$.
- “linear kernel” $\mathcal{K}(x, x') = \langle \Phi_n(x), \Phi_n(x') \rangle = \int_{\Omega} \langle x_n(u), x'_n(u) \rangle du$.

Patch extraction operator P_k

$$P_k x_{k-1}(u) := (x_{k-1}(u + v))_{v \in S_k} \in \mathcal{P}_k = \mathcal{H}_{k-1}^{S_k}$$



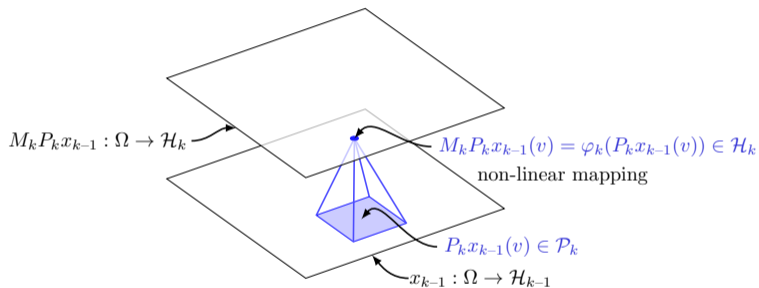
Patch extraction operator P_k

$$P_k x_{k-1}(u) := (x_{k-1}(u + v))_{v \in S_k} \in \mathcal{P}_k = \mathcal{H}_{k-1}^{S_k}$$

- S_k : patch shape, e.g. box

Non-linear mapping operator M_k

$$M_k P_k x_{k-1}(u) := \varphi_k(P_k x_{k-1}(u)) \in \mathcal{H}_k$$



Non-linear mapping operator M_k

$$M_k P_k x_{k-1}(u) := \varphi_k(P_k x_{k-1}(u)) \in \mathcal{H}_k$$

Kernel mapping of **homogeneous dot-product kernels**:

$$K_k(z, z') = \|z\| \|z'\| \kappa_k \left(\frac{\langle z, z' \rangle}{\|z\| \|z'\|} \right) = \langle \varphi_k(z), \varphi_k(z') \rangle.$$

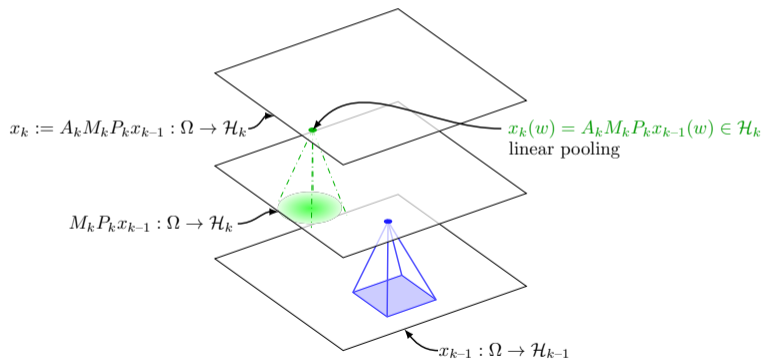
$$\kappa_k(u) = \sum_{j=0}^{\infty} b_j u^j \text{ with } b_j \geq 0, \kappa_k(1) = 1$$

Examples

- $\kappa_{\text{exp}}(\langle z, z' \rangle) = e^{\langle z, z' \rangle - 1}$ (Gaussian kernel on the sphere)
- $\kappa_{\text{inv-poly}}(\langle z, z' \rangle) = \frac{1}{2 - \langle z, z' \rangle}$

Pooling operator A_k

$$x_k(u) = A_k M_k P_k x_{k-1}(u) = \int_{\mathbb{R}^d} h_{\sigma_k}(u - v) M_k P_k x_{k-1}(v) dv \in \mathcal{H}_k$$



Pooling operator A_k

$$x_k(u) = A_k M_k P_k x_{k-1}(u) = \int_{\mathbb{R}^d} h_{\sigma_k}(u - v) M_k P_k x_{k-1}(v) dv \in \mathcal{H}_k$$

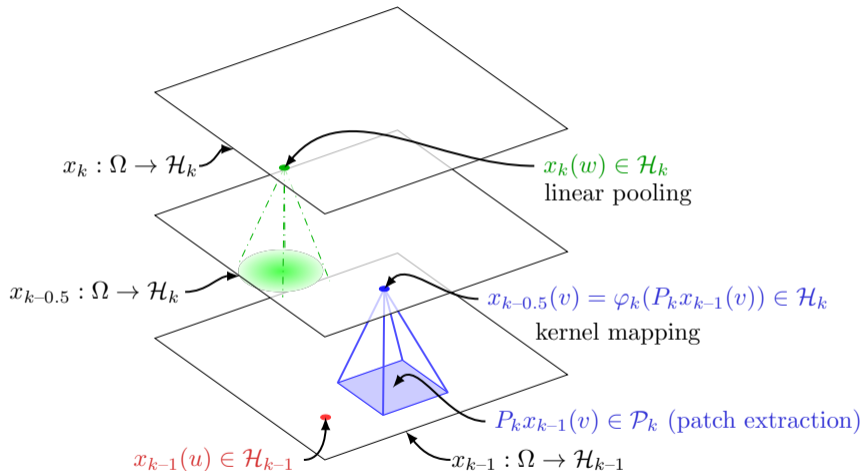
- h_{σ_k} : pooling filter at scale σ_k
- $h_{\sigma_k}(u) := \sigma_k^{-d} h(u/\sigma_k)$ with $h(u)$ **Gaussian**
- **linear, non-expansive operator**: $\|A_k\| \leq 1$

Pooling operator A_k

$$x_k(u) = A_k M_k P_k x_{k-1}(u) = \int_{\mathbb{R}^d} h_{\sigma_k}(u - v) M_k P_k x_{k-1}(v) dv \in \mathcal{H}_k$$

- h_{σ_k} : pooling filter at scale σ_k
- $h_{\sigma_k}(u) := \sigma_k^{-d} h(u/\sigma_k)$ with $h(u)$ **Gaussian**
- **linear, non-expansive operator**: $\|A_k\| \leq 1$
- In practice: **discretization**, sampling at resolution σ_k after pooling
- “Preserves information” when **subsampling** \leq **patch size**

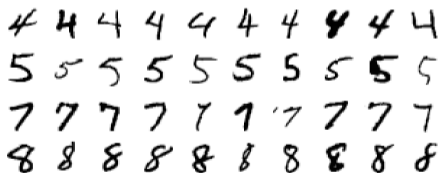
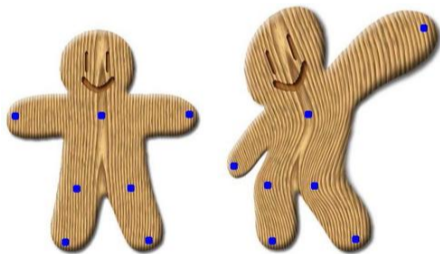
Recap: P_k, M_k, A_k



Stability to deformations

Deformations

- $\tau : \Omega \rightarrow \Omega$: C^1 -diffeomorphism
- $L_\tau x(u) = x(u - \tau(u))$: action operator
- Much richer group of transformations than translations



- Studied for wavelet-based scattering transform [Mallat, 2012, Bruna and Mallat, 2013]

Stability to deformations

Deformations

- $\tau : \Omega \rightarrow \Omega$: C^1 -diffeomorphism
- $L_\tau x(u) = x(u - \tau(u))$: action operator
- Much richer group of transformations than translations

Definition of stability

- Representation $\Phi(\cdot)$ is **stable** [Mallat, 2012] if:

$$\|\Phi(L_\tau x) - \Phi(x)\| \leq (C_1 \|\nabla \tau\|_\infty + C_2 \|\tau\|_\infty) \|x\|$$

- $\|\nabla \tau\|_\infty = \sup_u \|\nabla \tau(u)\|$ controls deformation
- $\|\tau\|_\infty = \sup_u |\tau(u)|$ controls translation
- $C_2 \rightarrow 0$: translation invariance

Smoothness and stability with kernels

Geometry of the kernel mapping: $f(x) = \langle f, \Phi(x) \rangle$

$$|f(x) - f(x')| \leq \|f\|_{\mathcal{H}} \cdot \|\Phi(x) - \Phi(x')\|_{\mathcal{H}}$$

- $\|f\|_{\mathcal{H}}$ controls **complexity** of the model
- $\Phi(x)$ encodes CNN **architecture** independently of the model (smoothness, invariance, stability to deformations)

Smoothness and stability with kernels

Geometry of the kernel mapping: $f(x) = \langle f, \Phi(x) \rangle$

$$|f(x) - f(x')| \leq \|f\|_{\mathcal{H}} \cdot \|\Phi(x) - \Phi(x')\|_{\mathcal{H}}$$

- $\|f\|_{\mathcal{H}}$ controls **complexity** of the model
- $\Phi(x)$ encodes CNN **architecture** independently of the model (smoothness, invariance, stability to deformations)

Useful kernels in practice:

- Convolutional kernel networks [**CKNs**, Mairal, 2016b] with efficient approximations
- Extends to neural tangent kernels [**NTKs**, Jacot et al., 2018] of infinitely wide CNNs [Bietti and Mairal, 2019b]

Recap: multilayer construction

Multilayer representation

$$\Phi(x_0) = A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 x_0 \in L^2(\Omega, \mathcal{H}_n).$$

- S_k, σ_k grow exponentially in practice (i.e., fixed with subsampling).

Recap: multilayer construction

Multilayer representation

$$\Phi(x_0) = A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 x_0 \in L^2(\Omega, \mathcal{H}_n).$$

- S_k, σ_k grow exponentially in practice (i.e., fixed with subsampling).

Assumption on x_0

- x_0 is typically a **discrete** signal acquired with physical device.
- Natural assumption: $x_0 = A_0 x$, with x the original continuous signal, A_0 local integrator with scale σ_0 (**anti-aliasing**).

Recap: multilayer construction

Multilayer representation

$$\Phi(x_0) = A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 x_0 \in L^2(\Omega, \mathcal{H}_n).$$

- S_k, σ_k grow exponentially in practice (i.e., fixed with subsampling).

Assumption on x_0

- x_0 is typically a **discrete** signal acquired with physical device.
- Natural assumption: $x_0 = A_0 x$, with x the original continuous signal, A_0 local integrator with scale σ_0 (**anti-aliasing**).

Final kernel

$$K_{CKN}(x, x') = \langle \Phi(x), \Phi(x') \rangle_{L^2(\Omega)} = \int_{\Omega} \langle x_n(u), x'_n(u) \rangle du$$

Warmup: translation invariance

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve translation invariance?

- Translation: $L_c x(u) = x(u - c)$.

Warmup: translation invariance

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve translation invariance?

- Translation: $L_c x(u) = x(u - c)$.
- *Equivariance* - all operators commute with L_c : $\square L_c = L_c \square$.

$$\begin{aligned} \|\Phi_n(L_c x) - \Phi_n(x)\| &= \|L_c \Phi_n(x) - \Phi_n(x)\| \\ &\leq \|L_c A_n - A_n\| \cdot \|M_n P_n \Phi_{n-1}(x)\| \\ &\leq \|L_c A_n - A_n\| \|x\|. \end{aligned}$$

Warmup: translation invariance

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve translation invariance?

- Translation: $L_c x(u) = x(u - c)$.
- *Equivariance* - all operators commute with L_c : $\square L_c = L_c \square$.

$$\begin{aligned} \|\Phi_n(L_c x) - \Phi_n(x)\| &= \|L_c \Phi_n(x) - \Phi_n(x)\| \\ &\leq \|L_c A_n - A_n\| \cdot \|M_n P_n \Phi_{n-1}(x)\| \\ &\leq \|L_c A_n - A_n\| \|x\|. \end{aligned}$$

- Mallat [2012]: $\|L_\tau A_n - A_n\| \leq \frac{C_2}{\sigma_n} \|\tau\|_\infty$ (operator norm).

Warmup: translation invariance

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve translation invariance?

- Translation: $L_c x(u) = x(u - c)$.
- *Equivariance* - all operators commute with L_c : $\square L_c = L_c \square$.

$$\begin{aligned} \|\Phi_n(L_c x) - \Phi_n(x)\| &= \|L_c \Phi_n(x) - \Phi_n(x)\| \\ &\leq \|L_c A_n - A_n\| \cdot \|M_n P_n \Phi_{n-1}(x)\| \\ &\leq \|L_c A_n - A_n\| \|x\|. \end{aligned}$$

- Mallat [2012]: $\|L_c A_n - A_n\| \leq \frac{C_2}{\sigma_n} c$ (operator norm).
- **Scale σ_n of the last layer controls translation invariance.**

Stability to deformations

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve stability to deformations?

- Patch extraction P_k and pooling A_k **do not commute** with L_τ !

Stability to deformations

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve stability to deformations?

- Patch extraction P_k and pooling A_k **do not commute** with L_τ !
- $\|A_k L_\tau - L_\tau A_k\| \leq C_1 \|\nabla \tau\|_\infty$ [from Mallat, 2012].

Stability to deformations

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve stability to deformations?

- Patch extraction P_k and pooling A_k **do not commute** with L_τ !
- $\|[A_k, L_\tau]\| \leq C_1 \|\nabla \tau\|_\infty$ [from Mallat, 2012].

Stability to deformations

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve stability to deformations?

- Patch extraction P_k and pooling A_k **do not commute** with L_τ !
- $\|[A_k, L_\tau]\| \leq C_1 \|\nabla\tau\|_\infty$ [from Mallat, 2012].
- But: $[P_k, L_\tau]$ is **unstable** at high frequencies!

Stability to deformations

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve stability to deformations?

- Patch extraction P_k and pooling A_k **do not commute** with L_τ !
- $\|[A_k, L_\tau]\| \leq C_1 \|\nabla\tau\|_\infty$ [from Mallat, 2012].
- But: $[P_k, L_\tau]$ is **unstable** at high frequencies!
- Adapt to **current layer resolution**, patch size controlled by σ_{k-1} :

$$\|[P_k A_{k-1}, L_\tau]\| \leq C_{1,\kappa} \|\nabla\tau\|_\infty \quad \sup_{u \in S_k} |u| \leq \kappa \sigma_{k-1}$$

Stability to deformations

Representation

$$\Phi_n(x) \triangleq A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x.$$

How to achieve stability to deformations?

- Patch extraction P_k and pooling A_k **do not commute** with L_τ !
- $\|[A_k, L_\tau]\| \leq C_1 \|\nabla\tau\|_\infty$ [from Mallat, 2012].
- But: $[P_k, L_\tau]$ is **unstable** at high frequencies!
- Adapt to **current layer resolution**, patch size controlled by σ_{k-1} :

$$\|[P_k A_{k-1}, L_\tau]\| \leq C_{1,\kappa} \|\nabla\tau\|_\infty \quad \sup_{u \in S_k} |u| \leq \kappa \sigma_{k-1}$$

- $C_{1,\kappa}$ grows as $\kappa^{d+1} \implies$ more stable with **small patches** (e.g., 3x3, VGG et al.).

Stability to deformations

Theorem (Stability of CKN [Bietti and Mairal, 2019a])

Let $\Phi_n(x) = \Phi(A_0 x)$ and assume $\|\nabla\tau\|_\infty \leq 1/2$,

$$\|\Phi_n(L_\tau x) - \Phi_n(x)\| \leq \left(C_\beta (n+1) \|\nabla\tau\|_\infty + \frac{C}{\sigma_n} \|\tau\|_\infty \right) \|x\|$$

- Translation invariance: large σ_n
 - Stability: small patch sizes ($\beta \approx$ patch size, $C_\beta = O(\beta^3)$ for images)
 - Signal preservation: subsampling factor \approx patch size
- \implies need several layers with small patches $n = O(\log(\sigma_n/\sigma_0)/\log \beta)$

Stability to deformations

Theorem (Stability of CKN [Bietti and Mairal, 2019a])

Let $\Phi_n(x) = \Phi(A_0 x)$ and assume $\|\nabla\tau\|_\infty \leq 1/2$,

$$\|\Phi_n(L_\tau x) - \Phi_n(x)\| \leq \left(C_\beta (n+1) \|\nabla\tau\|_\infty + \frac{C}{\sigma_n} \|\tau\|_\infty \right) \|x\|$$

- Translation invariance: large σ_n
- Stability: small patch sizes ($\beta \approx$ patch size, $C_\beta = O(\beta^3)$ for images)
- Signal preservation: subsampling factor \approx patch size
 \implies need several layers with small patches $n = O(\log(\sigma_n/\sigma_0)/\log \beta)$
- Achieved by controlling norm of **commutator** $[L_\tau, P_k A_{k-1}]$
 - Extend result by Mallat [2012] for controlling $\|[L_\tau, A]\|$
 - Need patches S_k adapted to resolution σ_{k-1} : $\text{diam } S_k \leq \beta \sigma_{k-1}$

Beyond the translation group

Can we achieve invariance to other groups?

- Group action: $L_g x(u) = x(g^{-1}u)$ (e.g., rotations, reflections).
- Feature maps $x(u)$ defined on $u \in G$ (G : locally compact group).

Beyond the translation group

Can we achieve invariance to other groups?

- Group action: $L_g x(u) = x(g^{-1}u)$ (e.g., rotations, reflections).
- Feature maps $x(u)$ defined on $u \in G$ (G : locally compact group).

Recipe: Equivariant inner layers + global pooling in last layer

- **Patch extraction:**

$$Px(u) = (x(uv))_{v \in S}.$$

- **Non-linear mapping:** equivariant because pointwise!
- **Pooling** (μ : left-invariant Haar measure):

$$Ax(u) = \int_G x(uv)h(v)d\mu(v) = \int_G x(v)h(u^{-1}v)d\mu(v).$$

related work [Sifre and Mallat, 2013, Cohen and Welling, 2016, Raj et al., 2016]...

Stability to deformations for convolutional NTK

Theorem (Stability of NTK [Bietti and Mairal, 2019b])

Let $\Phi_n(x) = \Phi^{NTK}(A_0x)$, and assume $\|\nabla\tau\|_\infty \leq 1/2$

$$\begin{aligned} & \|\Phi_n(L_\tau x) - \Phi_n(x)\| \\ & \leq \left(C_\beta n^{7/4} \|\nabla\tau\|_\infty^{1/2} + C'_\beta n^2 \|\nabla\tau\|_\infty + \sqrt{n+1} \frac{C}{\sigma_n} \|\tau\|_\infty \right) \|x\|, \end{aligned}$$

Comparison with random feature CKN on deformed MNIST digits:



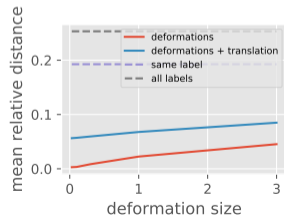
Stability to deformations for convolutional NTK

Theorem (Stability of NTK [Bietti and Mairal, 2019b])

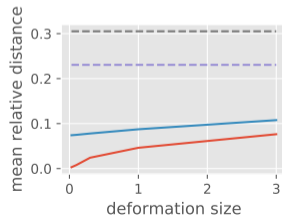
Let $\Phi_n(x) = \Phi^{NTK}(A_0x)$, and assume $\|\nabla\tau\|_\infty \leq 1/2$

$$\begin{aligned} & \|\Phi_n(L_\tau x) - \Phi_n(x)\| \\ & \leq \left(C_\beta n^{7/4} \|\nabla\tau\|_\infty^{1/2} + C'_\beta n^2 \|\nabla\tau\|_\infty + \sqrt{n+1} \frac{C}{\sigma_n} \|\tau\|_\infty \right) \|x\|, \end{aligned}$$

Comparison with random feature CKN on deformed MNIST digits:



(a) CKN



(b) NTK

Discretization and signal preservation: example in 1D

- Discrete signal \bar{x}_k in $\ell^2(\mathbb{Z}, \mathcal{H}_k)$ vs continuous ones x_k in $L^2(\mathbb{R}, \mathcal{H}_k)$.
- \bar{x}_k : subsampling factor s_k after pooling with scale $\sigma_k \approx s_k$:

$$\bar{x}_k[n] = \bar{A}_k \bar{M}_k \bar{P}_k \bar{x}_{k-1}[ns_k].$$

Discretization and signal preservation: example in 1D

- Discrete signal \bar{x}_k in $\ell^2(\mathbb{Z}, \mathcal{H}_k)$ vs continuous ones x_k in $L^2(\mathbb{R}, \mathcal{H}_k)$.
- \bar{x}_k : subsampling factor s_k after pooling with scale $\sigma_k \approx s_k$:

$$\bar{x}_k[n] = \bar{A}_k \bar{M}_k \bar{P}_k \bar{x}_{k-1}[ns_k].$$

- **Claim:** We can recover \bar{x}_{k-1} from \bar{x}_k if factor $s_k \leq$ **patch size**.

Discretization and signal preservation: example in 1D

- Discrete signal \bar{x}_k in $\ell^2(\mathbb{Z}, \mathcal{H}_k)$ vs continuous ones x_k in $L^2(\mathbb{R}, \mathcal{H}_k)$.
- \bar{x}_k : subsampling factor s_k after pooling with scale $\sigma_k \approx s_k$:

$$\bar{x}_k[n] = \bar{A}_k \bar{M}_k \bar{P}_k \bar{x}_{k-1}[ns_k].$$

- **Claim:** We can recover \bar{x}_{k-1} from \bar{x}_k if factor $s_k \leq$ **patch size**.
- **How?** Recover patches with **linear functions** (contained in \mathcal{H}_k)

$$\langle f_w, \bar{M}_k \bar{P}_k \bar{x}_{k-1}(u) \rangle = f_w(\bar{P}_k \bar{x}_{k-1}(u)) = \langle w, \bar{P}_k \bar{x}_{k-1}(u) \rangle,$$

and

$$\bar{P}_k \bar{x}_{k-1}(u) = \sum_{w \in B} \langle f_w, \bar{M}_k \bar{P}_k \bar{x}_{k-1}(u) \rangle w.$$

Discretization and signal preservation: example in 1D

- Discrete signal \bar{x}_k in $\ell^2(\mathbb{Z}, \mathcal{H}_k)$ vs continuous ones x_k in $L^2(\mathbb{R}, \mathcal{H}_k)$.
- \bar{x}_k : subsampling factor s_k after pooling with scale $\sigma_k \approx s_k$:

$$\bar{x}_k[n] = \bar{A}_k \bar{M}_k \bar{P}_k \bar{x}_{k-1}[ns_k].$$

- **Claim:** We can recover \bar{x}_{k-1} from \bar{x}_k if factor $s_k \leq$ **patch size**.
- **How?** Recover patches with **linear functions** (contained in \mathcal{H}_k)

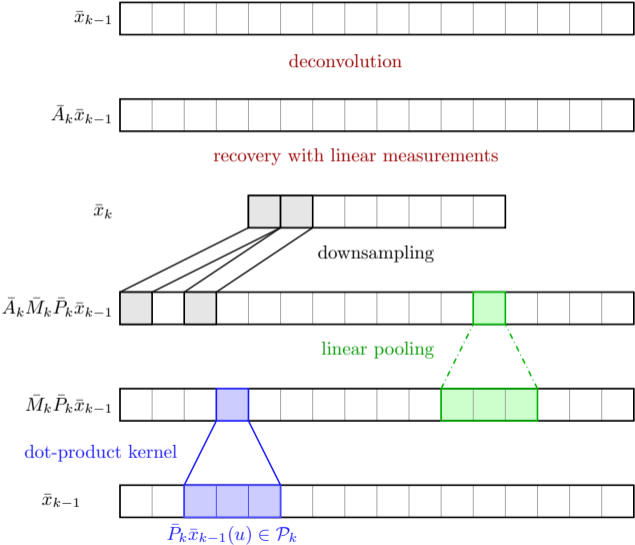
$$\langle f_w, \bar{M}_k \bar{P}_k \bar{x}_{k-1}(u) \rangle = f_w(\bar{P}_k \bar{x}_{k-1}(u)) = \langle w, \bar{P}_k \bar{x}_{k-1}(u) \rangle,$$

and

$$\bar{P}_k \bar{x}_{k-1}(u) = \sum_{w \in B} \langle f_w, \bar{M}_k \bar{P}_k \bar{x}_{k-1}(u) \rangle w.$$

Warning: no claim that recovery is practical and/or stable.

Discretization and signal preservation: example in 1D



RKHS of patch kernels K_k

$$K_k(z, z') = \|z\| \|z'\| \kappa\left(\frac{\langle z, z' \rangle}{\|z\| \|z'\|}\right), \quad \kappa(u) = \sum_{j=0}^{\infty} b_j u^j.$$

What does the RKHS contain?

Homogeneous version of [Zhang et al., 2016, 2017]

RKHS of patch kernels K_k

$$K_k(z, z') = \|z\| \|z'\| \kappa\left(\frac{\langle z, z' \rangle}{\|z\| \|z'\|}\right), \quad \kappa(u) = \sum_{j=0}^{\infty} b_j u^j.$$

What does the RKHS contain?

- RKHS contains **homogeneous functions**:

$$f : z \mapsto \|z\| \sigma(\langle g, z \rangle / \|z\|).$$

Homogeneous version of [Zhang et al., 2016, 2017]

RKHS of patch kernels K_k

$$K_k(z, z') = \|z\| \|z'\| \kappa\left(\frac{\langle z, z' \rangle}{\|z\| \|z'\|}\right), \quad \kappa(u) = \sum_{j=0}^{\infty} b_j u^j.$$

What does the RKHS contain?

- RKHS contains **homogeneous functions**:

$$f : z \mapsto \|z\| \sigma(\langle g, z \rangle / \|z\|).$$

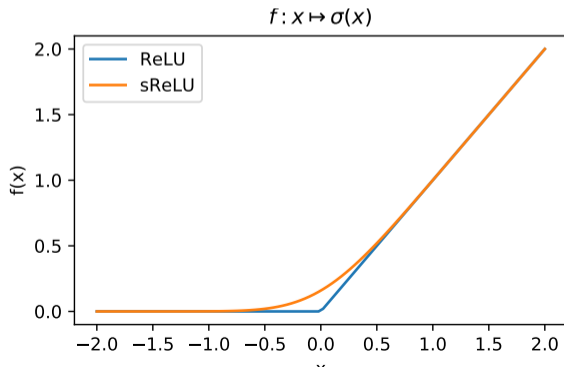
- **Smooth activations**: $\sigma(u) = \sum_{j=0}^{\infty} a_j u^j$ with $a_j \geq 0$.
- Norm: $\|f\|_{\mathcal{H}_k}^2 \leq C_{\sigma}^2 (\|g\|^2) = \sum_{j=0}^{\infty} \frac{a_j^2}{b_j} \|g\|^2 < \infty$.

Homogeneous version of [Zhang et al., 2016, 2017]

RKHS of patch kernels K_k

Examples:

- $\sigma(u) = u$ (linear): $C_\sigma^2(\lambda^2) = O(\lambda^2)$.
- $\sigma(u) = u^p$ (polynomial): $C_\sigma^2(\lambda^2) = O(\lambda^{2p})$.
- $\sigma \approx \sin$, sigmoid, smooth ReLU: $C_\sigma^2(\lambda^2) = O(e^{c\lambda^2})$.



Constructing a CNN in the RKHS $\mathcal{H}_{\mathcal{K}}$

Some CNNs live in the RKHS: “linearization” principle

$$f(x) = \sigma_k(W_k \sigma_{k-1}(W_{k-1} \dots \sigma_2(W_2 \sigma_1(W_1 x)) \dots)) = \langle f, \Phi(x) \rangle_{\mathcal{H}}.$$

Constructing a CNN in the RKHS $\mathcal{H}_{\mathcal{K}}$

Some CNNs live in the RKHS: “linearization” principle

$$f(x) = \sigma_k(W_k \sigma_{k-1}(W_{k-1} \dots \sigma_2(W_2 \sigma_1(W_1 x)) \dots)) = \langle f, \Phi(x) \rangle_{\mathcal{H}}.$$

- Consider a CNN with filters $W_k^{ij}(u), u \in S_k$.
 - k : layer;
 - i : index of filter;
 - j : index of input channel.
- “Smooth homogeneous” activations σ .
- The CNN can be constructed hierarchically in $\mathcal{H}_{\mathcal{K}}$.
- Norm (linear layers):

$$\|f_{\sigma}\|^2 \leq \|W_{n+1}\|_2^2 \cdot \|W_n\|_2^2 \cdot \|W_{n-1}\|_2^2 \dots \|W_1\|_2^2.$$

- Linear layers: product of spectral norms.

Link with generalization

Direct application of classical generalization bounds

- Simple bound on Rademacher complexity for linear/kernel methods:

$$\mathcal{F}_B = \{f \in \mathcal{H}_{\mathcal{K}}, \|f\| \leq B\} \implies \text{Rad}_N(\mathcal{F}_B) \leq O\left(\frac{BR}{\sqrt{N}}\right).$$

Link with generalization

Direct application of classical generalization bounds

- Simple bound on Rademacher complexity for linear/kernel methods:

$$\mathcal{F}_B = \{f \in \mathcal{H}_{\mathcal{K}}, \|f\| \leq B\} \implies \text{Rad}_N(\mathcal{F}_B) \leq O\left(\frac{BR}{\sqrt{N}}\right).$$

- Leads to margin bound $O(\|\hat{f}_N\|R/\gamma\sqrt{N})$ for a learned CNN \hat{f}_N with margin (confidence) $\gamma > 0$.
- Related to recent generalization bounds for neural networks based on **product of spectral norms** [e.g., Bartlett et al., 2017, Neyshabur et al., 2018].

[see, e.g., Boucheron et al., 2005, Shalev-Shwartz and Ben-David, 2014]...

Deep convolutional representations: conclusions

Study of generic properties of signal representation

- **Deformation stability** with small patches, adapted to resolution.
- **Signal preservation** when subsampling \leq patch size.
- **Group invariance** by changing patch extraction and pooling.

Deep convolutional representations: conclusions

Study of generic properties of signal representation

- **Deformation stability** with small patches, adapted to resolution.
- **Signal preservation** when subsampling \leq patch size.
- **Group invariance** by changing patch extraction and pooling.

Applies to learned models

- Same quantity $\|f\|$ controls stability and generalization.
- “higher capacity” is needed to discriminate small deformations.

Deep convolutional representations: conclusions

Study of generic properties of signal representation

- **Deformation stability** with small patches, adapted to resolution.
- **Signal preservation** when subsampling \leq patch size.
- **Group invariance** by changing patch extraction and pooling.

Applies to learned models

- Same quantity $\|f\|$ controls stability and generalization.
- “higher capacity” is needed to discriminate small deformations.

Questions:

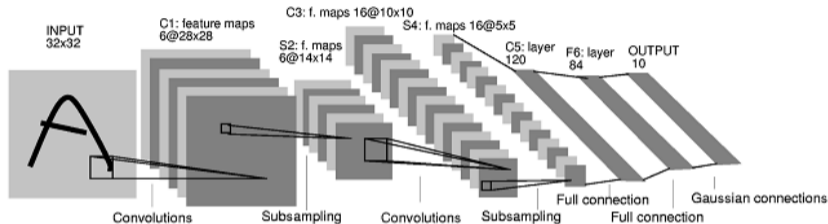
- Better regularization?
- How does SGD control capacity in CNNs?
- What about networks with no pooling layers? ResNet?

Robust Deep Learning Models with Kernels

- A. Bietti, G. Mialon, D. Chen, and J. Mairal. A Kernel Perspective for Regularizing Deep Neural Networks. International Conference on Machine Learning (ICML). 2019.

Convolutional Neural Networks

Picture from LeCun et al. [1998]

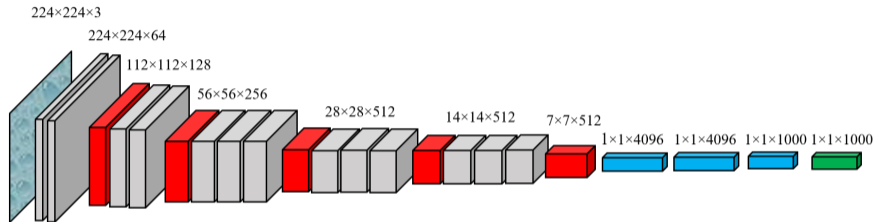


What are the main features of CNNs?

- they capture **compositional** and **multiscale** structures in images;
- they provide some **invariance**;
- they model the **local stationarity** of images at several scales;

Convolutional Neural Networks

[Simonyan and Zisserman, 2014]



What are the main features of CNNs?

- they capture **compositional** and **multiscale** structures in images;
- they provide some **invariance**;
- they model the **local stationarity** of images at several scales;

Convolutional neural networks for biological sequences

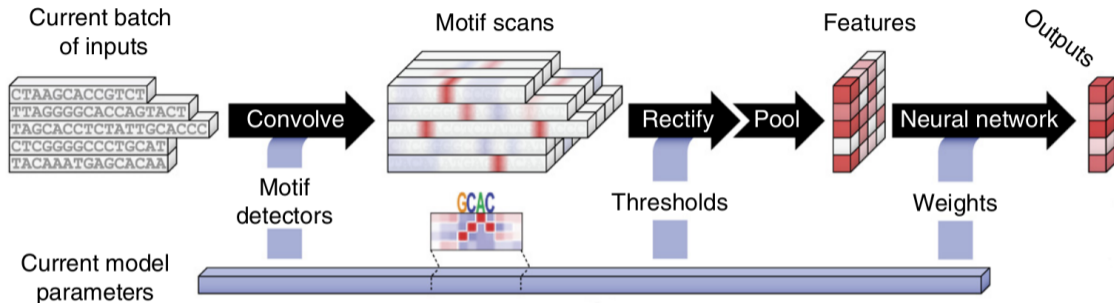


Figure: two-layer CNN architecture from Alipanahi et al. [2015]

- Sequences are represented by one-hot encoding ($A=(1,0,0,0), C=(0,1,0,0), \dots$).
- Single convolution layer followed by linear classifier.

Adversarial examples, Picture from Kurakin et al. [2016]

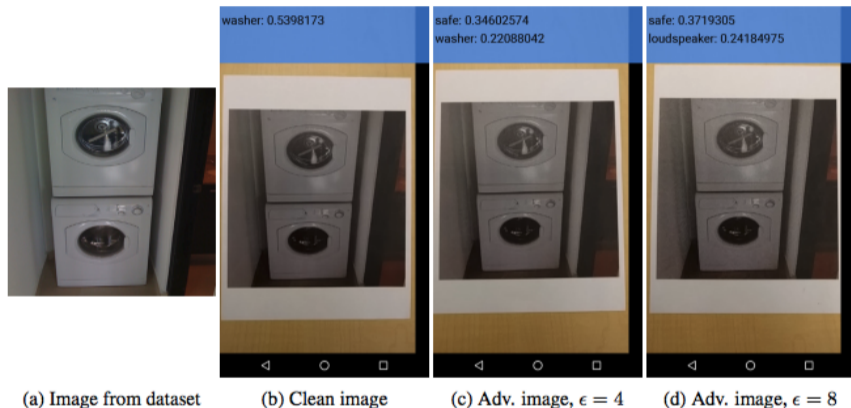
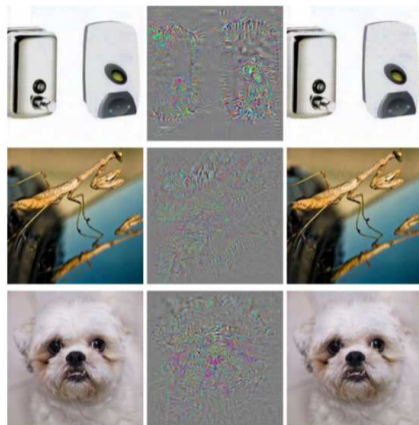


Figure: Adversarial examples are generated by computer; then printed on paper; a new picture taken on a smartphone fools the classifier.

Adversarial Examples



(b)

clean + noise \rightarrow “ostrich” [Szegedy et al., 2013].

Adversarial Examples



(a real ostrich)

Adversarial Examples



adversarial
perturbation



88% **tabby cat**

99% **guacamole**

<https://github.com/anishathalye/obfuscated-gradients>

Convolutional Neural Networks

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}} .$$

The issue of regularization

- today, heuristics are used (DropOut, weight decay, early stopping)...
- ...but they are not sufficient.
- how to **control variations of prediction functions**?

$|f(x) - f(x')|$ should be close if x and x' are “similar”.

- what does it mean for x and x' to be “similar”?
- what should be a good **regularization function** Ω ?

A kernel perspective: regularization

Assume we have an RKHS \mathcal{H} for deep networks:

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2.$$

$\|\cdot\|_{\mathcal{H}}$ encourages smoothness and stability w.r.t. the geometry induced by the kernel (which depends itself on the choice of architecture).

A kernel perspective: regularization

Assume we have an RKHS \mathcal{H} for deep networks:

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2.$$

$\|\cdot\|_{\mathcal{H}}$ encourages smoothness and stability w.r.t. the geometry induced by the kernel (which depends itself on the choice of architecture).

Problem

Multilayer kernels developed for deep networks are **typically intractable**.

One solution [Mairal, 2016a]

do kernel approximations at each layer, which leads to non-standard CNNs called convolutional kernel networks (CKNs).

A kernel perspective: regularization

Assume we have an RKHS \mathcal{H} for deep networks:

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2.$$

$\|\cdot\|_{\mathcal{H}}$ encourages smoothness and stability w.r.t. the geometry induced by the kernel (which depends itself on the choice of architecture).

Problem

Multilayer kernels developed for deep networks are **typically intractable**.

One solution [Mairal, 2016a]

do kernel approximations at each layer, which leads to non-standard CNNs called convolutional kernel networks (CKNs).

not the subject of this part.

A kernel perspective: regularization

Consider a classical CNN parametrized by θ , which live in the RKHS:

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n L(y_i, f_{\theta}(x_i)) + \frac{\lambda}{2} \|f_{\theta}\|_{\mathcal{H}}^2.$$

This is different than CKNs since f_{θ} admits a classical parametrization.

A kernel perspective: regularization

Consider a classical CNN parametrized by θ , which live in the RKHS:

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n L(y_i, f_{\theta}(x_i)) + \frac{\lambda}{2} \|f_{\theta}\|_{\mathcal{H}}^2.$$

This is different than CKNs since f_{θ} admits a classical parametrization.

Problem

$\|f_{\theta}\|_{\mathcal{H}}$ is **intractable**...

One solution [Bietti et al., 2019]

use approximations (lower- and upper-bounds), based on mathematical properties of $\|\cdot\|_{\mathcal{H}}$.

A kernel perspective: regularization

Consider a classical CNN parametrized by θ , which live in the RKHS:

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n L(y_i, f_{\theta}(x_i)) + \frac{\lambda}{2} \|f_{\theta}\|_{\mathcal{H}}^2.$$

This is different than CKNs since f_{θ} admits a classical parametrization.

Problem

$\|f_{\theta}\|_{\mathcal{H}}$ is **intractable**...

One solution [Bietti et al., 2019]

use approximations (lower- and upper-bounds), based on mathematical properties of $\|\cdot\|_{\mathcal{H}}$.

This is the subject of this part.

A kernel perspective: regularization

Another point of view: consider a classical CNN parametrized by θ , which live in the RKHS:

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n L(y_i, f_{\theta}(x_i)) + \frac{\lambda}{2} \|f_{\theta}\|_{\mathcal{H}}^2.$$

Upper-bounds

$$\|f_{\theta}\|_{\mathcal{H}} \leq \omega(\|W_k\|, \|W_{k-1}\|, \dots, \|W_1\|) \quad (\text{spectral norms}),$$

where the W_j 's are the convolution filters. The bound suggests controlling the spectral norm of the filters.

[Cisse et al., 2017, Miyato et al., 2018, Bartlett et al., 2017]...

A kernel perspective: regularization

Another point of view: consider a classical CNN parametrized by θ , which live in the RKHS:

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n L(y_i, f_{\theta}(x_i)) + \frac{\lambda}{2} \|f_{\theta}\|_{\mathcal{H}}^2.$$

Lower-bounds

$$\|f\|_{\mathcal{H}} = \sup_{\|u\|_{\mathcal{H}} \leq 1} \langle f, u \rangle_{\mathcal{H}} \geq \sup_{u \in U} \langle f, u \rangle_{\mathcal{H}} \quad \text{for } U \subseteq B_{\mathcal{H}}(1).$$

We design a set U that leads to a tractable approximation, but it requires **some knowledge** about the properties of \mathcal{H}, Φ .

A kernel perspective: regularization

Adversarial penalty

We know that Φ is **non-expansive** and $f(x) = \langle f, \Phi(x) \rangle$. Then,

$$U = \{\Phi(x + \delta) - \Phi(x) : x \in \mathcal{X}, \|\delta\|_2 \leq 1\}$$

leads to

$$\lambda \|f\|_\delta^2 = \sup_{x \in \mathcal{X}, \|\delta\|_2 \leq \lambda} f(x + \delta) - f(x).$$

The resulting strategy is related to **adversarial regularization** (but it is decoupled from the loss term and does not use labels).

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n L(y_i, f_\theta(x_i)) + \sup_{x \in \mathcal{X}, \|\delta\|_2 \leq \lambda} f_\theta(x + \delta) - f_\theta(x).$$

[Madry et al., 2018]

A kernel perspective: regularization

Adversarial penalty

We know that Φ is **non-expansive** and $f(x) = \langle f, \Phi(x) \rangle$. Then,

$$U = \{\Phi(x + \delta) - \Phi(x) : x \in \mathcal{X}, \|\delta\|_2 \leq 1\}$$

leads to

$$\lambda \|f\|_\delta^2 = \sup_{x \in \mathcal{X}, \|\delta\|_2 \leq \lambda} f(x + \delta) - f(x).$$

The resulting strategy is related to **adversarial regularization** (but it is decoupled from the loss term and does not use labels).

vs, for adversarial regularization,

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n \sup_{\|\delta\|_2 \leq \lambda} L(y_i, f_\theta(x_i + \delta)).$$

[Madry et al., 2018]

A kernel perspective: regularization

Gradient penalties

We know that Φ is non-expansive and $f(x) = \langle f, \Phi(x) \rangle$. Then,

$$U = \{\Phi(x + \delta) - \Phi(x) : x \in \mathcal{X}, \|\delta\|_2 \leq 1\}$$

leads to

$$\|\nabla f\| = \sup_{x \in \mathcal{X}} \|\nabla f(x)\|_2.$$

Related penalties have been used to stabilize the training of GANs and gradients of the **loss function** have been used to improve robustness.

[Gulrajani et al., 2017, Roth et al., 2017, 2018, Drucker and Le Cun, 1991, Lyu et al., 2015, Simon-Gabriel et al., 2018]

A kernel perspective: regularization

Adversarial deformation penalties

We know that Φ is **stable to deformations** and $f(x) = \langle f, \Phi(x) \rangle$. Then,

$$U = \{\Phi(L_\tau x) - \Phi(x) : x \in \mathcal{X}, \tau\}$$

leads to

$$\|f\|_\tau^2 = \sup_{\substack{x \in \mathcal{X} \\ \tau \text{ small deformation}}} f(L_\tau x) - f(x).$$

This is related to **data augmentation** and **tangent propagation**.

[Engstrom et al., 2017, Simard et al., 1998]

Experiments with Few labeled Samples

Table: Accuracies on CIFAR10 with 1 000 examples for standard architectures VGG-11 and ResNet-18. With / without data augmentation.

Method	1k VGG-11	1k ResNet-18
No weight decay	50.70 / 43.75	45.23 / 37.12
Weight decay	51.32 / 43.95	44.85 / 37.09
SN projection	54.14 / 46.70	47.12 / 37.28
PGD- ℓ_2	51.25 / 44.40	45.80 / 41.87
grad- ℓ_2	55.19 / 43.88	49.30 / 44.65
$\ f\ _{\delta}^2$ penalty	51.41 / 45.07	48.73 / 43.72
$\ \nabla f\ ^2$ penalty	54.80 / 46.37	48.99 / 44.97
PGD- ℓ_2 + SN proj	54.19 / 46.66	47.47 / 41.25
grad- ℓ_2 + SN proj	55.32 / 46.88	48.73 / 42.78
$\ f\ _{\delta}^2$ + SN proj	54.02 / 46.72	48.12 / 43.56
$\ \nabla f\ ^2$ + SN proj	55.24 / 46.80	49.06 / 44.92

Experiments with Few labeled Samples

Table: Accuracies with 300 or 1 000 examples from MNIST, using deformations. (*) indicates that random deformations were included as training examples,

Method	300 VGG	1k VGG
Weight decay	89.32	94.08
SN projection	90.69	95.01
grad- ℓ_2	93.63	96.67
$\ f\ _{\delta}^2$ penalty	94.17	96.99
$\ \nabla f\ ^2$ penalty	94.08	96.82
Weight decay (*)	92.41	95.64
grad- ℓ_2 (*)	95.05	97.48
$\ D_{\tau} f\ ^2$ penalty	94.18	96.98
$\ f\ _{\tau}^2$ penalty	94.42	97.13
$\ f\ _{\tau}^2 + \ \nabla f\ ^2$	94.75	97.40
$\ f\ _{\tau}^2 + \ f\ _{\delta}^2$	95.23	97.66
$\ f\ _{\tau}^2 + \ f\ _{\delta}^2$ (*)	95.53	97.56
$\ f\ _{\tau}^2 + \ f\ _{\delta}^2 + \text{SN proj}$	95.20	97.60
$\ f\ _{\tau}^2 + \ f\ _{\delta}^2 + \text{SN proj}$ (*)	95.40	97.77

Experiments with Few labeled Samples

Table: AUROC50 for protein homology detection tasks using CNN, with or without data augmentation (DA).

Method	No DA	DA
No weight decay	0.446	0.500
Weight decay	0.501	0.546
SN proj	0.591	0.632
PGD- l_2	0.575	0.595
grad- l_2	0.540	0.552
$\ f\ _{\delta}^2$	0.600	0.608
$\ \nabla f\ ^2$	0.585	0.611
PGD- l_2 + SN proj	0.596	0.627
grad- l_2 + SN proj	0.592	0.624
$\ f\ _{\delta}^2$ + SN proj	0.630	0.644
$\ \nabla f\ ^2$ + SN proj	0.603	0.625

Experiments with Few labeled Samples

Table: AUROC50 for protein homology detection tasks using CNN, with or without data augmentation (DA).

Method	No DA	DA
No weight decay	0.446	0.500
Weight decay	0.501	0.546
SN proj	0.591	0.632
PGD- l_2	0.575	0.595
grad- l_2	0.540	0.552
$\ f\ _{\delta}^2$	0.600	0.608
$\ \nabla f\ ^2$	0.585	0.611
PGD- l_2 + SN proj	0.596	0.627
grad- l_2 + SN proj	0.592	0.624
$\ f\ _{\delta}^2$ + SN proj	0.630	0.644
$\ \nabla f\ ^2$ + SN proj	0.603	0.625

Note: statistical tests have been conducted for all of these experiments (see paper).

Adversarial Robustness: Trade-offs

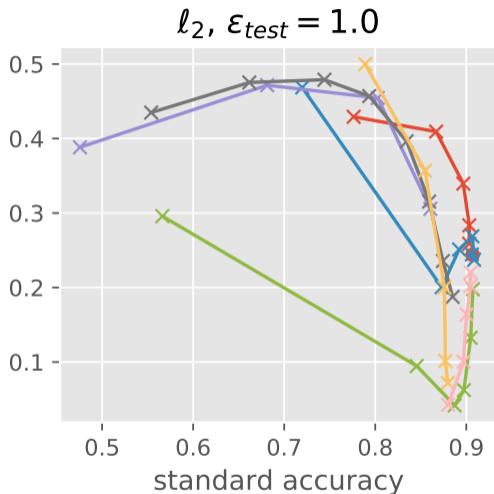
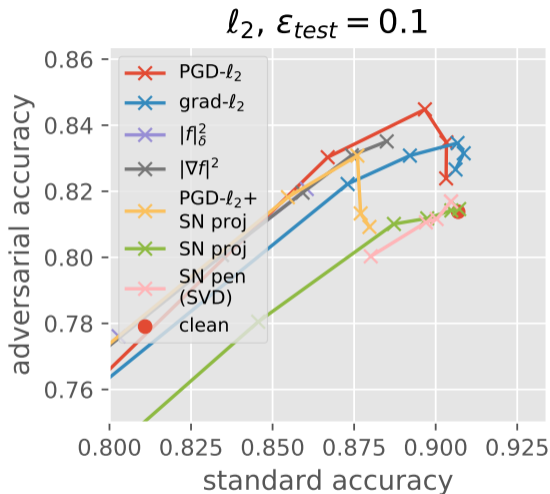


Figure: Robustness trade-off curves of different regularization methods for VGG11 on CIFAR10. Each plot shows test accuracy vs adversarial test accuracy. Different points on a curve correspond to

Conclusions from this work on regularization

What the kernel perspective brings us

- gives a **unified perspective on many regularization principles**.
- useful both for **generalization and robustness**.
- related to **robust optimization**.

Future work

- regularization based on kernel approximations.
- semi-supervised learning to exploit unlabeled data.
- relation with implicit regularization.

References I

- Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.
- Fabio Anselmi, Lorenzo Rosasco, Cheston Tan, and Tomaso Poggio. Deep convolutional networks are hierarchical kernel machines. *arXiv preprint arXiv:1508.01084*, 2015.
- Peter Bartlett, Dylan J Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1706.08498*, 2017.
- Alberto Bietti and Julien Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations. *Journal of Machine Learning Research (JMLR)*, 2019a.
- Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b.
- Alberto Bietti, Grégoire Mialon, Dexiong Chen, and Julien Mairal. A kernel perspective for regularizing deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

References II

- L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.
- Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.
- Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics*, 9:323–375, 2005.
- J. V. Bouvrie, L. Rosasco, and T. Poggio. On invariance in hierarchical models. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.
- David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document, 1988.
- Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE Transactions on pattern analysis and machine intelligence (PAMI)*, 35(8):1872–1886, 2013.
- Dexiong Chen, Laurent Jacob, and Julien Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, 35(18):3294–3302, 02 2019a.

References III

- Dexiong Chen, Laurent Jacob, and Julien Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b.
- Lenaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.
- Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.
- Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning (ICML)*, 2016.
- A. Damianou and N. Lawrence. Deep Gaussian processes. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.
- Harris Drucker and Yann Le Cun. Double backpropagation increasing generalization performance. In *International Joint Conference on Neural Networks (IJCNN)*, 1991.

References IV

- Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- Logan Engstrom, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 2017.
- Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.
- Sepp Hochreiter, Martin Heusel, and Klaus Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736, 2007.
- Jie Hou, Badri Adhikari, and Jianlin Cheng. DeepSF: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics*, 34(8):1295–1303, 12 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx780. URL <https://doi.org/10.1093/bioinformatics/btx780>.

References V

- Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- Nils M Kriege, Marion Neumann, Christopher Morris, Kristian Kersting, and Petra Mutzel. A unifying view of explicit and implicit feature maps of graph kernels. *Data Mining and Knowledge Discovery*, 33(6):1505–1547, 2019.
- Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *P. IEEE*, 86(11):2278–2324, 1998.
- Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

References VI

- C. Leslie, E. Eskin, J. Weston, and W.S. Noble. Mismatch String Kernels for SVM Protein Classification. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003. URL <http://www.cs.columbia.edu/~cleslie/papers/mismatch-short.pdf>.
- Christina Leslie and Rui Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5(Nov):1435–1455, 2004.
- Christina S Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, volume 7, pages 566–575. Hawaii, USA, 2002.
- Li Liao and William Stafford Noble. Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships. *Journal of computational biology*, 10(6):857–868, 2003.
- Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research (JMLR)*, 2:419–444, 2002.
- Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang. A unified gradient regularization family for adversarial examples. In *IEEE International Conference on Data Mining (ICDM)*, 2015.

References VII

- Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- J. Mairal. End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016a.
- Julien Mairal. End-to-End Kernel Learning with Supervised Convolutional Kernel Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016b.
- Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.
- Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65 (10):1331–1398, 2012.
- Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

References VIII

- GrÅšgoire Montavon, Mikio L Braun, and Klaus-Robert MÅÅ§ller. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 12(Sep):2563–2581, 2011.
- Alyssa Morrow, Vaishaal Shankar, Devin Petersohn, Anthony Joseph, Benjamin Recht, and Nir Yosef. Convolutional kitchen sinks for transcription factor binding site prediction. *arXiv preprint arXiv:1706.00125*, 2017.
- Radford M Neal. *Bayesian learning for neural networks*. Springer, 1996.
- Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- Mattis Paulin, Matthijs Douze, Zaid Harchaoui, Julien Mairal, Florent Perronin, and Cordelia Schmid. Local convolutional features with unsupervised training for image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

References IX

- Anant Raj, Abhishek Kumar, Youssef Mroueh, P Thomas Fletcher, and Bernhard Scholkopf. Local group invariant representations via orbit embeddings. *preprint arXiv:1612.01988*, 2016.
- Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.
- Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Adversarially robust training through structured gradient regularization. *arXiv preprint arXiv:1805.08736*, 2018.
- B. Scholkopf. *Support Vector Learning*. PhD thesis, Technischen Universität Berlin, 1997.
- Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.
- Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

References X

- Vaishaal Shankar, Alex Fang, Wenshuo Guo, Sara Fridovich-Keil, Ludwig Schmidt, Jonathan Ragan-Kelley, and Benjamin Recht. Neural kernels without tangents. *preprint arXiv:2003.02237*, 2020.
- John Shawe-Taylor and Nello Cristianini. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2004.
- Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12: 2539–2561, 2011.
- Laurent Sifre and Stéphane Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2013.
- Patrice Y Simard, Yann A LeCun, John S Denker, and Bernard Victorri. Transformation invariance in pattern recognition—tangent distance and tangent propagation. In *Neural networks: tricks of the trade*, pages 239–274. Springer, 1998.

References XI

- Carl-Johann Simon-Gabriel, Yann Ollivier, Bernhard Schölkopf, Léon Bottou, and David Lopez-Paz. Adversarial vulnerability of neural networks increases with input dimension. *arXiv preprint arXiv:1802.01421*, 2018.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.
- Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.
- Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein Weisfeiler-Lehman graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1995.
- Jean-Philippe Vert, Hiroto Saigo, and Tatsuya Akutsu. Convolution and local alignment kernels. *Kernel methods in computational biology*, pages 131–154, 2004.

References XII

- Christopher KI Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.
- Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
- Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- Y. Zhang, P. Liang, and M. J. Wainwright. Convexified convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 2017.
- Yuchen Zhang, Jason D Lee, and Michael I Jordan. ℓ_1 -regularized neural networks are improperly learnable in polynomial time. In *International Conference on Machine Learning (ICML)*, 2016.