On the Happy Marriage of Kernel Methods and Deep Learning

Julien Mairal
Inria Grenoble
Context of supervised learning

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

\[
\min_{f \in \mathcal{F}} \quad \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda \Omega(f).
\]

- empirical risk, data fit
- regularization

[Vapnik, 1995, Bottou, Curtis, and Nocedal, 2016]...
Kernel Methods 1/2

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

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

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

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

[Shawe-Taylor and Cristianini, 2004, Schölkopf and Smola, 2002]...
In the context of supervised learning with labels in $\mathbb{R}$,

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

- $f(x) = \langle \Phi(x), f \rangle_{\mathcal{H}}$ 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 $\|\|_{\mathcal{H}}$ (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)

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

Graph-structured data is everywhere

(a) molecules

(b) protein regulation

(c) social networks

(d) chemical pathways
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.
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; 
- $a : \mathcal{V} \to \mathbb{R}^d$ is a function assigning attributes to each node.
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

$$\Phi(G) := \sum_{u \in \mathcal{V}} \varphi_{\text{base}}(\ell_G(u))$$

where $\varphi_{\text{base}}$ embeds some local patterns $\ell_G(u)$ 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

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

1. Represent explicitly each graph $x$ by a vector of fixed dimension $(x)_2 \mathbb{R}^p$.
2. Use an algorithm for regression or pattern recognition in $\mathbb{R}^p$.

$\phi \in \mathcal{H}$

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

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

- 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

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

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

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

- \( a(p) \): concatenated attributes in \( p \); \( \delta \): 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:

- \( \delta \) allows hard comparison between paths thus only works for discrete attributes;
- \( \delta \) is not differentiable, which cannot be “optimized” with back-propagation.
A relaxed path kernel

\[ \varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G,u)} \delta_a(p)(\cdot) \]

\[ \implies \sum_{p \in \mathcal{P}_k(G,u)} e^{-\frac{\alpha}{2} \|a(p) - \cdot\|^2}. \]

Issues of the path kernel mapping:
- $\delta$ allows hard comparison between paths thus only works for discrete attributes;
- $\delta$ 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 $u$. 

Julien Mairal
From kernel methods to deep learning 12/107
One-layer GCKN: a closer look at the relaxed path kernel

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

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

- This formula can be divided into 3 steps:
  - path extraction: enumerating all \( \mathcal{P}_k(G,u) \);
  - kernel mapping: evaluating Gaussian embedding \( \varphi_{RBF} \) 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 \( \varphi_{\text{RBF}} \) 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

\[ \varphi_1(u) \in \mathcal{H}_1 \]

\[ \varphi_1(u) := \varphi_{RBF}(a(p_1)) + \varphi_{RBF}(a(p_2)) + \varphi_{RBF}(a(p_3)) \]

\[ (\mathcal{V}, \mathcal{E}, \varphi_1 : \mathcal{V} \rightarrow \mathcal{H}_1) \]

\[ (\mathcal{V}, \mathcal{E}, a : \mathcal{V} \rightarrow \mathbb{R}^d) \]

\[ \mathbb{R}^d \]

\[ \mathcal{H}_1 \]
From one-layer to multilayer GCKN

- We can repeat applying $\varphi_{\text{path}}$ to the new graph

\[(V, E, a) \xrightarrow{\varphi_{\text{path}}} (V, E, \varphi_1) \xrightarrow{\varphi_{\text{path}}} (V, E, \varphi_2) \xrightarrow{\varphi_{\text{path}}} \ldots \xrightarrow{\varphi_{\text{path}}} (V, E, \varphi_j).\]

- Final graph representation at layer $j$, $\Phi(G) = \sum_{u \in V} \varphi_j(u)$.
From one-layer to multilayer GCKN

- We can repeat applying $\varphi_{\text{path}}$ to the new graph

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

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

- Why is the multilayer model interesting?
  - applying $\varphi_{\text{path}}$ 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

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

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

[Chen et al., 2019a,b, Williams and Seeger, 2001]
Scalable approximation of Gaussian kernel mapping

\[ \varphi_{\text{path}}(u) = \sum_{p \in P_k(G,u)} \varphi_{\text{RBF}}(a(p)). \]

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

\[ \text{Span}(\varphi_{\text{RBF}}(z_1), \ldots, \varphi_{\text{RBF}}(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}, \]

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

[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;
- **Nyström** provides a finite-dimensional approximation \( \Psi(a(p)) \) by orthogonally projecting \( \varphi_{\text{RBF}}(a(p)) \) onto some finite-dimensional subspace:

\[ \text{Span}(\varphi_{\text{RBF}}(z_1), \ldots, \varphi_{\text{RBF}}(z_q)) \]

parametrized by \( Z = \{z_1, \ldots, z_q\} \),

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]
Comparison of GCKN and GNN

<table>
<thead>
<tr>
<th></th>
<th>GCKN</th>
<th>vs.</th>
<th>GNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{GCKN}(G)$</td>
<td>$\sum_{u \in G} \psi_k(u)$</td>
<td></td>
<td>$f_{GNN}(G) = \sum_{u \in G} f_k(u)$</td>
</tr>
<tr>
<td>$\psi_k(u)$</td>
<td>$\sum_{p \in \mathcal{P}<em>k(G,u)} \kappa(Z^T Z)^{-\frac{1}{2}} \kappa(Z^T \psi</em>{k-1}(p))$</td>
<td></td>
<td>$f_k(u) = \sum_{v \in \mathcal{N}(u)} \text{ReLU}(Z^T f_{k-1}(v))$</td>
</tr>
<tr>
<td>local path aggregation</td>
<td></td>
<td></td>
<td>neighborhood aggregation</td>
</tr>
<tr>
<td>projection in a known RKHS</td>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>supervised and unsupervised</td>
<td></td>
<td></td>
<td>supervised</td>
</tr>
</tbody>
</table>
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.

![Diagram of molecular structures with labels C, O, Cl, H, N, F.]

Original

GCKN

[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

Sequence modeling as a supervised learning problem

\[ \min_{f \in \mathcal{F}} \sum_{i=1}^{n} L(y_i, f(x_i)) + \mu \Omega(f) \]

empirical risk, data fit

regularization

How do we define the functional space \( \mathcal{F} \)?
Sequence modeling as a supervised learning problem

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

\[
\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \mu \Omega(f).
\]

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

Julien Mairal  From kernel methods to deep learning 23/107
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]

\[ K_{\text{CKN}}(x, x') = \sum_{i=1}^{\lfloor x \rfloor - k + 1} \sum_{j=1}^{\lfloor x' \rfloor - k + 1} K_0(x[i:i+k], x'[j:j+k]) \]

- Use one-hot encoding
  \[ x[i:i+5] := \text{TTGAG} \mapsto \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \]

- \( K_0 \) 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]

\[ K_{CKN}(x, x') = \sum_{i=1}^{\|x\| - k + 1} \sum_{j=1}^{\|x'\| - k + 1} \langle \varphi_0(x[i:i+k]), \varphi_0(x'[j:j+k]) \rangle. \]

- Use one-hot encoding

\[ x[i:i+5] := TTGAG \mapsto \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}. \]

- \( K_0 \) 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]

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

- Use one-hot encoding

\[
x[i:i+5] := \text{TGTAG} \mapsto \begin{bmatrix}
A & 0 & 0 & 0 & 1 & 0 \\
T & 1 & 1 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 0 & 0 \\
G & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}.
\]

- \(K_0\) 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:

\[ \mathcal{E}_0 = \text{Span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}. \]
Scalable Approximation of Kernel Mapping (with more details this time)

\[ K_0(u, u') = \langle \varphi_0(u), \varphi_0(u') \rangle_{H_0} \approx \langle \psi_0(u), \psi_0(u') \rangle_{R^q}. \]

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

  \[ E_0 = \text{Span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}. \]

- **General case:**

  \[ \psi_0(u) = [K_0(z_i, z_j)]_{i,j}^{-1/2} [K_0(z_1, u), \ldots, K_0(z_q, u)]^T = K_0(Z, Z)^{-1/2} K_0(Z, u). \]
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:

\[ \mathcal{E}_0 = \text{Span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}. \]

- 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

\[ \Psi(x) \in \mathbb{R}^q \]

Prediction layer

\[ \langle w, \Psi(x) \rangle \]

global pooling

Kernel mapping approximation

\[ \psi_0(P_i(x)) = \mathbf{K}_{Z}^{-\frac{1}{2}} \mathbf{K}_Z(P_i(x)) \]

\[ x \in \mathcal{X} \]

\[ P_i(x) \text{ } k\text{-mer} \]

\[ x(u) \in \mathcal{A} \]
Multilayer CKN for sequence modeling

\[ \psi_0(P_i(x)) \in \mathbb{R}^q \]

kernel mapping approximation
\[ \psi_0(P_i(x)) = K_{ZZ}^{-1/2}K_Z(P_i(x)) \]

global pooling
\[ \langle w, \psi_0(P_i(x)) \rangle \]

prediction layer
\[ \langle w, \psi_0(P_i(x)) \rangle \]

prediction layer
\[ \Psi(x) \in \mathbb{R}^q \]

y
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 $\kappa(Z^T Z)^{-1/2}$ (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 $x = x_1 \ldots x_n \in \mathcal{X}$ of length $n$ and a sequence of ordered indices $i = (i_1, \ldots, i_k)$ in $I(k, n)$, we define a k-substring as:

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

- We introduce the quantity

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

- Example: $x = \text{ABRA\textsc{CADABA}RA}$

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

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

\[
K_{CKN}(x, x') = \sum_{i=1}^{\lfloor x \rfloor - k + 1} \sum_{j=1}^{\lfloor x' \rfloor - k + 1} K_0(\mathbf{x}_{[i:i+k]}, \mathbf{x}'_{[j:j+k]}).
\]

- The kernel mapping is \( \Phi(x) = \sum_{i=1}^{\lfloor x \rfloor - k + 1} \varphi_0(\mathbf{x}_{[i:i+k]}) \).
Recurrent kernel networks

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

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

- The kernel mapping is \( \Phi(x) = \sum_{i \in I(k,|x|)} \lambda^{gaps(i)} \varphi_0(x[i]). \)
- 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 \( K_{\text{RKN}} \) via Nyström approximation is

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

where, as usual with a dot-product kernel, \( \psi_0(x_i) = \kappa(Z^\top Z)^{-1/2} \kappa(Z^\top x_i) \).

- 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 $\lambda^{\text{gaps}(i)}$.

$$\lambda^2 \varphi_0(x[i])$$

$$\sum_i \lambda^{\text{gap}(i)} \varphi_0(x[i])$$

**Figure:** Example of $K_{\text{RKN}}$ 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)

<table>
<thead>
<tr>
<th>Method</th>
<th>#Params</th>
<th>Accuracy</th>
<th>Level-stratified accuracy (top1/top5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>top 1</td>
<td>top 5</td>
</tr>
<tr>
<td>PSI-BLAST</td>
<td>-</td>
<td>84.53</td>
<td>86.48</td>
</tr>
<tr>
<td>DeepSF</td>
<td>920k</td>
<td>73.00</td>
<td>90.25</td>
</tr>
<tr>
<td>CKN (128 filters)</td>
<td>211k</td>
<td>76.30</td>
<td>92.17</td>
</tr>
<tr>
<td>CKN (512 filters)</td>
<td>843k</td>
<td>84.11</td>
<td>94.29</td>
</tr>
<tr>
<td>RKN (128 filters)</td>
<td>211k</td>
<td>77.82</td>
<td>92.89</td>
</tr>
<tr>
<td>RKN (512 filters)</td>
<td>843k</td>
<td><strong>85.29</strong></td>
<td><strong>94.95</strong></td>
</tr>
</tbody>
</table>

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)**

<table>
<thead>
<tr>
<th>Method</th>
<th>pooling</th>
<th>one-hot</th>
<th>BLOSUM62</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>auROC</td>
<td>auROC50</td>
</tr>
<tr>
<td>SVM-pairwise</td>
<td></td>
<td>0.724</td>
<td>0.359</td>
</tr>
<tr>
<td>Mismatch</td>
<td></td>
<td>0.814</td>
<td>0.467</td>
</tr>
<tr>
<td>LA-kernel</td>
<td></td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LSTM</td>
<td></td>
<td>0.830</td>
<td>0.566</td>
</tr>
<tr>
<td>CKN</td>
<td></td>
<td>0.837</td>
<td>0.572</td>
</tr>
<tr>
<td><strong>RKN</strong></td>
<td>mean</td>
<td>0.829</td>
<td>0.541</td>
</tr>
<tr>
<td><strong>RKN</strong></td>
<td>max</td>
<td>0.844</td>
<td>0.587</td>
</tr>
<tr>
<td><strong>RKN (unsup)</strong></td>
<td>mean</td>
<td>0.805</td>
<td>0.504</td>
</tr>
</tbody>
</table>

[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.

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

Construction of the RKHS for continuous signals

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

\[ x_0 : \Omega \rightarrow \mathcal{H}_0 : \text{continuous input signal, with } \Omega = \mathbb{R}^d: \text{location} \]

- \( x_0(u) \in \mathcal{H}_0: \) input value at location \( u \)  

\[ (d = 2 \text{ for images}). \]

\[ (\mathcal{H}_0 = \mathbb{R}^3 \text{ for RGB images}). \]
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

- $x_0(u) \in \mathcal{H}_0$: input value at location $u$

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 $L^2(\Omega, \mathcal{H}_0)$

$x_0 : \Omega \to \mathcal{H}_0$: continuous input signal, with $\Omega = \mathbb{R}^d$: location $(d = 2$ for images). $\quad (\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).

- $x_0(u) \in \mathcal{H}_0$: input value at location $u$

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

$x_k : \Omega \to \mathcal{H}_k$: 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

- $x_0(u) \in \mathcal{H}_0$: input value at location $u$

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 $\sigma_k$. 

$(d = 2$ for images). 

$(\mathcal{H}_0 = \mathbb{R}^3$ for RGB images).
Construction of the RKHS for continuous signals

\[ x_k : \Omega \rightarrow \mathcal{H}_k \]
\[ x_{k-0.5} : \Omega \rightarrow \mathcal{H}_k \]
\[ x_{k-1} : \Omega \rightarrow \mathcal{H}_{k-1} \]
\[ x_{k-0.5}(v) = \varphi_k(P_k x_{k-1}(v)) \in \mathcal{H}_k \]
\[ P_k x_{k-1}(v) \in \mathcal{P}_k \text{ (patch extraction)} \]
\[ x_k(w) \in \mathcal{H}_k \text{ (linear pooling)} \]
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

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

- \( \sigma_k \) 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

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

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

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 $\kappa(Z^T Z)^{-1/2}$ is fun). The kernel interpretation provides regularization mechanisms. Kernel representations can possibly be used in other contexts (statistical testing? kernel PCA? CCA? K-means?).
Convolutional Kernel Networks in Practice

What is the difference with a CNN?

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

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 $\kappa(Z^TZ)^{-1/2}$ 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**
Take-home messages

- unsupervised representations are shallow and high-dimensional;
- supervised representations may be deep and compact;
- **Our code is freely available at**

Open

- how to close the gap between the approximate embedding and the exact kernel?
Theory for Deep Learning Models

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

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

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

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

**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]: \( \theta \) stays close to \( \theta_0 \) 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

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

**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]: \( \theta \) stays close to \( \theta_0 \) when training with large \( m \)
  \[ f_\theta(x) \approx f_{\theta_0}(x) + \langle \theta - \theta_0, \nabla_\theta f_{\theta_0}(x) |_{\theta=\theta_0} \rangle. \]
- Gradient descent for \( m \to \infty \approx \) kernel ridge regression with **neural tangent kernel**
  \[ K_{NTK}(x, x') = \lim_{m \to \infty} \langle \nabla_\theta f_{\theta_0}(x), \nabla_\theta f_{\theta_0}(x') \rangle \]
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**.

Paradigm 3: Deep Kernel Machines
A quick zoom on convolutional neural networks still involves the ERM problem:

\[
\min_{f \in \mathcal{F}} \sum_{i=1}^{n} L(y_i, f(x_i)) + \|\mathcal{F}(f)\|
\]

[LeCun et al., 1989, 1998, Ciresan et al., 2012, Krizhevsky et al., 2012]...

Julien Mairal Soutenance HdR 8/33

From kernel methods to deep learning 50/107
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].

$$f(x) = \sigma_k(W_k\sigma_{k-1}(W_{k-1} \ldots \sigma_2(W_2\sigma_1(W_1x)) \ldots)) = \langle f, \Phi(x) \rangle_{\mathcal{H}}.$$  

- 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].

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

- 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 $\Phi$.
- **Stability to deformations and non-expansiveness** for $\Phi$.
- 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 $\Phi$.
- **Stability to deformations and non-expansiveness** for $\Phi$.
- 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

\[ z \mapsto \text{ReLU}(w^\top z) \quad \longrightarrow \quad z \mapsto \|z\| \sigma\left(\frac{w^\top z}{\|z\|}\right). \]
Recap: Construction of the RKHS for continuous signals

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

\[ x_0 : \Omega \rightarrow \mathcal{H}_0 : \text{continuous input signal, with } \Omega = \mathbb{R}^d : \text{location} \quad (d = 2 \text{ for images}). \]

\[ x_0(u) \in \mathcal{H}_0 : \text{input value at location } u \]

(\( \mathcal{H}_0 = \mathbb{R}^3 \) for RGB images).

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

\[ x_k : \Omega \rightarrow \mathcal{H}_k : \text{feature map at layer } k \]

- \( P_k : \text{patch extraction} \) 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 \( u \)).

- \( M_k : \text{non-linear mapping} \) operator, maps each patch to a new Hilbert space \( \mathcal{H}_k \) with a pointwise non-linear function \( \varphi_k(\cdot) \).

- \( A_k : \text{(linear) pooling} \) operator at scale \( \sigma_k \).
Recap: Construction of the RKHS for continuous signals

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

$x_0 : \Omega \to \mathcal{H}_0$: 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 $L^2(\Omega, \mathcal{H}_{k-1})$

$x_k : \Omega \to \mathcal{H}_k$: 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 $\sigma_k$. 

Julien Mairal

From kernel methods to deep learning
Recap: Construction of the RKHS for continuous signals

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

$x_0 : \Omega \to \mathcal{H}_0$: continuous input signal, with $\Omega = \mathbb{R}^d$: location

- $x_0(u) \in \mathcal{H}_0$: input value at location $u$

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

$x_k : \Omega \to \mathcal{H}_k$: 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 $\sigma_k$. 
Recap: Construction of the RKHS for continuous signals

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

$x_0 : \Omega \to \mathcal{H}_0$: **continuous** input signal, with $\Omega = \mathbb{R}^d$: location

- $x_0(u) \in \mathcal{H}_0$: input value at location $u$

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

$x_k : \Omega \to \mathcal{H}_k$: **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 $\sigma_k$. 

($d = 2$ for images). ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images). 

Julien Mairal
From kernel methods to deep learning 54/107
Construction of the RKHS for continuous signals

\[ x_k : \Omega \rightarrow \mathcal{H}_k \]

\[ x_{k-0.5} : \Omega \rightarrow \mathcal{H}_k \]

\[ x_{k-0.5}(v) = \varphi_k(P_k x_{k-1}(v)) \in \mathcal{H}_k \]

kernel mapping

\[ x_{k-1}(u) \in \mathcal{H}_{k-1} \]

linear pooling

\[ P_k x_{k-1}(v) \in \mathcal{P}_k \text{ (patch extraction)} \]
Construction of the RKHS for continuous signals

Multilayer representation

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

- \( \sigma_k \) grows exponentially in practice (i.e., fixed with subsampling).
Construction of the RKHS for continuous signals

Multilayer representation

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

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

Prediction layer

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

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

$x_{k-1}(u) \in \mathcal{H}_{k-1}$

$x_{k-1} : \Omega \rightarrow \mathcal{H}_{k-1}$

$P_k x_{k-1}(u) \in \mathcal{P}_k$ (patch extraction)
Patch extraction operator $P_k$

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

- $S_k$: patch shape, e.g. box
Non-linear mapping operator $M_k$

$$M_kP_kx_{k-1}(u) := \varphi_k(P_kx_{k-1}(u)) \in \mathcal{H}_k$$
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 $A_k$

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

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

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

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

$P_k \times k^{−1} : \Omega \rightarrow H_k^{xk^{−1}}(u) \in H_k^{xk^{−1}}$

$P_kx_k^{−1}(v) \in P_k$ (patch extraction)

$P_kx_k^{−1}(v) = \varphi_k(P_kx_k^{−1}(v)) \in H_k$

kernel mapping

$P_kx_k^{−1}(v) \in P_k$ (patch extraction)

$x_{k−0.5}(v) = \varphi_k(P_kx_k^{−1}(v)) \in H_k$

linear pooling

$x_{k−0.5}(w) \in H_k$

linear pooling

$x_{k−0.5} : \Omega \rightarrow H_k$

$x_k : \Omega \rightarrow H_k$

$x_k \in H_k$

kernel mapping

$x_k(w) \in H_k$

$x_k(w) \in H_k$

$x_k : \Omega \rightarrow H_k$

$x_k(w) \in H_k$
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

Stability to deformations

Deformations

- $\tau : \Omega \to \Omega$: $C^1$-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
- $C_2 \to 0$: 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:** \( f(x) = \langle f, \Phi(x) \rangle \)

\[
|f(x) - f(x')| \leq \|f\|_H \cdot \|\Phi(x) - \Phi(x')\|_H
\]

- \(\|f\|_H\) 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 \([\text{CKNs}, \text{Mairal, 2016b}]\) with efficient approximations
- Extends to neural tangent kernels \([\text{NTKs}, \text{Jacot et al., 2018}]\) of infinitely wide CNNs \([\text{Bietti and Mairal, 2019b}]\)
Recap: multilayer construction

Multilayer representation

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

- $S_k$, $\sigma_k$ grow exponentially in practice (i.e., fixed with subsampling).
Recap: multilayer construction

Multilayer representation

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

- \( S_k, \sigma_k \) 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 \( \sigma_0 \) (**anti-aliasing**).
Recap: multilayer construction

Multilayer representation

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

- \( S_k, \sigma_k \) 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 \( \sigma_0 \) (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: \( L_c x(u) = x(u - c) \).
- \textit{Equivariance} - all operators commute with \( L_c \): \( \square L_c = L_c \square \).

\[
\|\Phi_n(L_c x) - \Phi_n(x)\| = \|L_c \Phi_n(x) - \Phi_n(x)\|
\leq \|L_c A_n - A_n\| \cdot \|M_n P_n \Phi_{n-1}(x)\|
\leq \|L_c A_n - A_n\|||x||.
\]
Warmup: translation invariance

Representation

\[ \Phi_n(x) \overset{\Delta}{=} A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x. \]

How to achieve translation invariance?

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

\[
\| \Phi_n(L_c x) - \Phi_n(x) \| = \| L_c \Phi_n(x) - \Phi_n(x) \| \\
\leq \| L_c A_n - A_n \| \cdot \| M_n P_n \Phi_{n-1}(x) \| \\
\leq \| L_c A_n - A_n \| \| x \| .
\]

- 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 \( L_c \): \( \Box L_c = L_c \Box \).

\[
\| \Phi_n(L_c x) - \Phi_n(x) \| = \| L_c \Phi_n(x) - \Phi_n(x) \|
\leq \| L_c A_n - A_n \| \cdot \| M_n P_n \Phi_{n-1}(x) \|
\leq \| L_c A_n - A_n \| \| x \|. 
\]

- Mallat [2012]: \( \| L_c A_n - A_n \| \leq \frac{C_2}{\sigma_n} c \) (operator norm).
- Scale \( \sigma_n \) 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 $L_\tau$!
Stability to deformations

Representation

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

How to achieve stability to deformations?

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

\[ \|A_0 \| \leq \frac{C_1}{\kappa} \]

\[ \|A_1 \| \leq \frac{C_1}{\kappa^2} \]

\[ \|A_2 \| \leq \frac{C_1}{\kappa^3} \]

\[ \cdots \]

\[ \|A_n \| \leq \frac{C_1}{\kappa^n} \]
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 \( L_\tau \)!
- \( \|[A_k, L_\tau]\| \leq C_1 \|\nabla \tau\|_\infty \) [from Mallat, 2012].
Stability to deformations

Representation

\[ \Phi_n(x) \overset{\Delta}{=} A_n M_n P_n A_{n-1} M_{n-1} P_{n-1} \cdots A_1 M_1 P_1 A_0 x. \]

How to achieve stability to deformations?

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

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

Representation

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

How to achieve stability to deformations?

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

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

- $C_{1,\kappa}$ 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 \( \sigma_n \)
- 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 $\sigma_n$
- Stability: small patch sizes ($\beta \approx$ patch size, $C_\beta = O(\beta^3)$ for images)
- Signal preservation: subsampling factor $\approx$ patch size
  $$
  \implies \text{need several layers with small patches } n = O(\log(\sigma_n/\sigma_0) / \log \beta)
  $$

- 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 $\sigma_{k-1}$: $\text{diam } S_k \leq \beta \sigma_{k-1}$
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:**
  \[ P x(u) = (x(uv))_{v \in S} \]

- **Non-linear mapping:** equivariant because pointwise!

- **Pooling** (\( \mu \): left-invariant Haar measure):
  \[ A x(u) = \int_G x(uv)h(v)d\mu(v) = \int_G x(v)h(u^{-1}v)d\mu(v). \]

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

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

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 $\|\nabla \tau\|_\infty \leq 1/2$

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

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 $\sigma_k \approx s_k$:

$$\bar{x}_k[n] = A_k M_k P_k \bar{x}_{k-1}[ns_k].$$

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 $\bar{H}_k$):

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

Warning: no claim that recovery is practical and/or stable.
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 $s_k \leq \text{patch size}$.

Warning: no claim that recovery is practical and/or stable.
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 $\sigma_k \approx s_k$:

$$\bar{x}_k[n] = A_k M_k P_k \bar{x}_{k-1}[ns_k].$$

- 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 $\bar{\mathcal{H}}_k$)

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

and

$$\bar{P}_k \bar{x}_{k-1}(u) = \sum_{w \in B} \langle f_w, M_k \bar{P}_k \bar{x}_{k-1}(u) \rangle w.$$
Discretization and signal preservation: example in 1D

- Discrete signal $\bar{x}_k$ in $\ell^2(\mathbb{Z}, \bar{H}_k)$ 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 $s_k \leq \text{patch size}$.
- How? Recover patches with linear functions (contained in $\bar{H}_k$)
  \[
  \langle f_w, \bar{M}_k \bar{P}_k \bar{x}_{k-1}(u) \rangle = f_w(\bar{P}_k \bar{x}_{k-1}(u)) = \langle w, \bar{P}_k \bar{x}_{k-1}(u) \rangle,
  \]
  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

\[ \bar{x}_{k-1} \]

deconvolution

\[ \bar{A}_k \bar{x}_{k-1} \]

recovery with linear measurements

\[ \bar{x}_k \]

downsampling

\[ \bar{A}_k \bar{M}_k \bar{P}_k \bar{x}_{k-1} \]

dot-product kernel

\[ \bar{M}_k \bar{P}_k \bar{x}_{k-1} \]

linear pooling

\[ \bar{x}_{k-1} \bar{P}_k \bar{x}_{k-1}(u) \in \mathcal{P}_k \]
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**: $\|f\|_H^2 \leq C_\sigma^2(\|g\|^2) = \sum_{j=0}^{\infty} \frac{a_j^2}{b_j} \|g\|^2 < \infty$.

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

![Diagram showing comparison of ReLU and sReLU functions]
Constructing a CNN in the RKHS $\mathcal{H}_K$

Some CNNs live in the RKHS: “linearization” principle

$$f(x) = \sigma_k(W_k\sigma_{k-1}(W_{k-1} \ldots \sigma_2(W_2\sigma_1(W_1x)) \ldots)) = \langle f, \Phi(x) \rangle_{\mathcal{H}}.$$
Constructing a CNN in the RKHS $\mathcal{H}_K$

Some CNNs live in the RKHS: “linearization” principle

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

- 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 $\sigma$.

- The CNN can be constructed hierarchically in $\mathcal{H}_K$.

- Norm (linear layers):
  $$\|f\sigma\|^2 \leq \|W_{n+1}\|_2^2 \cdot \|W_n\|_2^2 \cdot \|W_{n-1}\|_2^2 \ldots \|W_1\|_2^2.$$

- Linear layers: product of spectral norms.
Link with generalization

Direct application of classical generalization bounds

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

\[ \mathcal{F}_B = \{ f \in \mathcal{H}_K, \| f \| \leq B \} \implies \text{Rad}_N(\mathcal{F}_B) \leq O\left( \frac{BR}{\sqrt{N}} \right). \]
Link with generalization

Direct application of classical generalization bounds

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

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

- 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].
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 ≤ 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

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 $(A=(1,0,0,0), C=(0,1,0,0), \ldots)$.
- 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

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

(a real ostrich)
Adversarial Examples

88% tabby cat -> adversarial perturbation -> 99% guacamole

https://github.com/anishathalye/obfuscated-gradients
Convolutional Neural Networks

\[
\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda \Omega(f) .
\]

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 \(\Omega\)?
A kernel perspective: regularization

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

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

$\| . \|_{\mathcal{H}}$ 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:

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

$\| . \|_{\mathcal{H}}$ 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 $\theta$, which live in the RKHS:

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

This is different than CKNs since $f_\theta$ admits a classical parametrization.
A kernel perspective: regularization

Consider a classical CNN parametrized by $\theta$, which live in the RKHS:

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

This is different than CKNs since $f_\theta$ 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 $\| . \|_{\mathcal{H}}$. 
A kernel perspective: regularization

Consider a classical CNN parametrized by $\theta$, which live in the RKHS:

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

This is different than CKNs since $f_{\theta}$ admits a classical parametrization.

**Problem**

$\|f_{\theta}\|_H$ is *intractable*...

**One solution** [Bietti et al., 2019]

use approximations (lower- and upper-bounds), based on mathematical properties of $\|.\|_H$.

**This is the subject of this part.**
A kernel perspective: regularization

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

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

Upper-bounds

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

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 $\theta$, which live in the RKHS:

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

Lower-bounds

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

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

From kernel methods to deep learning 87/107
A kernel perspective: regularization

Adversarial penalty

We know that $\Phi$ 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

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

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 $\Phi$ 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 $\Phi$ 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.

A kernel perspective: regularization

Adversarial deformation penalties

We know that $\Phi$ 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

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

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>1k VGG-11</th>
<th>1k ResNet-18</th>
</tr>
</thead>
<tbody>
<tr>
<td>No weight decay</td>
<td>50.70 / 43.75</td>
<td>45.23 / 37.12</td>
</tr>
<tr>
<td>Weight decay</td>
<td>51.32 / 43.95</td>
<td>44.85 / 37.09</td>
</tr>
<tr>
<td>SN projection</td>
<td>54.14 / 46.70</td>
<td>47.12 / 37.28</td>
</tr>
<tr>
<td>PGD-(\ell_2)</td>
<td>51.25 / 44.40</td>
<td>45.80 / 41.87</td>
</tr>
<tr>
<td>grad-(\ell_2)</td>
<td><strong>55.19</strong> / 43.88</td>
<td><strong>49.30</strong> / 44.65</td>
</tr>
<tr>
<td>(|f|_2^2) penalty</td>
<td>51.41 / 45.07</td>
<td>48.73 / 43.72</td>
</tr>
<tr>
<td>(|\nabla f|_2^2) penalty</td>
<td>54.80 / 46.37</td>
<td><strong>48.99</strong> / 44.97</td>
</tr>
<tr>
<td>PGD-(\ell_2) + SN proj</td>
<td>54.19 / 46.66</td>
<td>47.47 / 41.25</td>
</tr>
<tr>
<td>grad-(\ell_2) + SN proj</td>
<td><strong>55.32</strong> / 46.88</td>
<td>48.73 / 42.78</td>
</tr>
<tr>
<td>(|f|_2^2) + SN proj</td>
<td>54.02 / 46.72</td>
<td>48.12 / 43.56</td>
</tr>
<tr>
<td>(|\nabla f|_2^2) + SN proj</td>
<td><strong>55.24</strong> / 46.80</td>
<td><strong>49.06</strong> / 44.92</td>
</tr>
</tbody>
</table>
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,

<table>
<thead>
<tr>
<th>Method</th>
<th>300 VGG</th>
<th>1k VGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight decay</td>
<td>89.32</td>
<td>94.08</td>
</tr>
<tr>
<td>SN projection</td>
<td>90.69</td>
<td>95.01</td>
</tr>
<tr>
<td>grad-$\ell_2$</td>
<td>93.63</td>
<td>96.67</td>
</tr>
<tr>
<td>$|f|_2^2$ penalty</td>
<td>94.17</td>
<td>96.99</td>
</tr>
<tr>
<td>$|\nabla f|_2^2$ penalty</td>
<td>94.08</td>
<td>96.82</td>
</tr>
<tr>
<td>Weight decay (*)</td>
<td>92.41</td>
<td>95.64</td>
</tr>
<tr>
<td>grad-$\ell_2$ (*)</td>
<td>95.05</td>
<td>97.48</td>
</tr>
<tr>
<td>$|D_\tau f|_2^2$ penalty</td>
<td>94.18</td>
<td>96.98</td>
</tr>
<tr>
<td>$|f|_2^2$ penalty</td>
<td>94.42</td>
<td>97.13</td>
</tr>
<tr>
<td>$|f|_2^2 + |\nabla f|_2^2$</td>
<td>94.75</td>
<td>97.40</td>
</tr>
<tr>
<td>$|f|_2^2 + |f|_2^2$</td>
<td>95.23</td>
<td>97.66</td>
</tr>
<tr>
<td>$|f|_2^2 + |f|_2^2$ (*)</td>
<td>95.53</td>
<td>97.56</td>
</tr>
<tr>
<td>$|f|_2^2 + |f|_2^2$ + SN proj</td>
<td>95.20</td>
<td>97.60</td>
</tr>
<tr>
<td>$|f|_2^2 + |f|_2^2$ + SN proj (*)</td>
<td>95.40</td>
<td>97.77</td>
</tr>
</tbody>
</table>
### Experiments with Few labeled Samples

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

<table>
<thead>
<tr>
<th>Method</th>
<th>No DA</th>
<th>DA</th>
</tr>
</thead>
<tbody>
<tr>
<td>No weight decay</td>
<td>0.446</td>
<td>0.500</td>
</tr>
<tr>
<td>Weight decay</td>
<td>0.501</td>
<td>0.546</td>
</tr>
<tr>
<td>SN proj</td>
<td>0.591</td>
<td>0.632</td>
</tr>
<tr>
<td>PGD-$\ell_2$</td>
<td>0.575</td>
<td>0.595</td>
</tr>
<tr>
<td>grad-$\ell_2$</td>
<td>0.540</td>
<td>0.552</td>
</tr>
<tr>
<td>$|f|_\delta^2$</td>
<td>0.600</td>
<td>0.608</td>
</tr>
<tr>
<td>$|\nabla f|_2^2$</td>
<td>0.585</td>
<td>0.611</td>
</tr>
<tr>
<td>PGD-$\ell_2$ + SN proj</td>
<td>0.596</td>
<td>0.627</td>
</tr>
<tr>
<td>grad-$\ell_2$ + SN proj</td>
<td>0.592</td>
<td>0.624</td>
</tr>
<tr>
<td>$|f|_\delta^2$ + SN proj</td>
<td>0.630</td>
<td>0.644</td>
</tr>
<tr>
<td>$|\nabla f|_2^2$ + SN proj</td>
<td>0.603</td>
<td>0.625</td>
</tr>
</tbody>
</table>

*Note*: statistical tests have been conducted for all of these experiments (see paper).
Experiments with Few labeled Samples

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

<table>
<thead>
<tr>
<th>Method</th>
<th>No DA</th>
<th>DA</th>
</tr>
</thead>
<tbody>
<tr>
<td>No weight decay</td>
<td>0.446</td>
<td>0.500</td>
</tr>
<tr>
<td>Weight decay</td>
<td>0.501</td>
<td>0.546</td>
</tr>
<tr>
<td>SN proj</td>
<td>0.591</td>
<td>0.632</td>
</tr>
<tr>
<td>PGD-(\ell_2)</td>
<td>0.575</td>
<td>0.595</td>
</tr>
<tr>
<td>grad-(\ell_2)</td>
<td>0.540</td>
<td>0.552</td>
</tr>
<tr>
<td>(|f|_\delta^2)</td>
<td>0.600</td>
<td>0.608</td>
</tr>
<tr>
<td>(|\nabla f|_2^2)</td>
<td>0.585</td>
<td>0.611</td>
</tr>
<tr>
<td>PGD-(\ell_2) + SN proj</td>
<td>0.596</td>
<td>0.627</td>
</tr>
<tr>
<td>grad-(\ell_2) + SN proj</td>
<td>0.592</td>
<td>0.624</td>
</tr>
<tr>
<td>(|f|_\delta^2 + SN proj)</td>
<td>0.630</td>
<td>0.644</td>
</tr>
<tr>
<td>(|\nabla f|_2^2 + SN proj)</td>
<td>0.603</td>
<td>0.625</td>
</tr>
</tbody>
</table>

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 training with different regularization strengths.
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 II


References IV


References V


References VI


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 VIII


References IX


References


References XI


References XII


