Approximation and Generalization with Convolutional Kernels

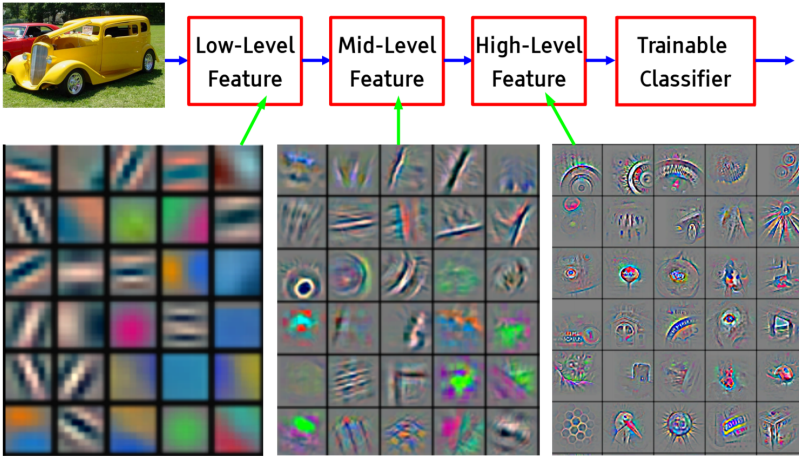
Alberto Bietti

NYU

EPFL. April 26, 2021.

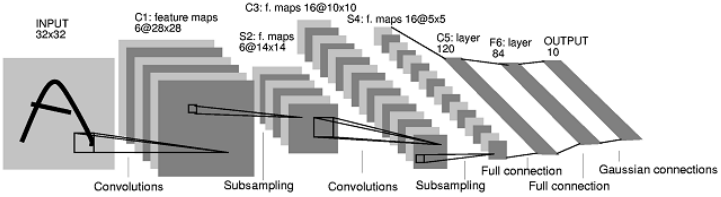
Convolutional networks

Exploiting structure of natural images (LeCun et al., 1989)



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutional networks



(LeCun et al., 1998)

Convolutional networks

- Model local neighborhoods at different scales
- Provide some invariance through pooling
- Useful **inductive bias** for learning efficiently on natural images

Understanding deep learning

The challenge of deep learning theory

- **Over-parameterized** (millions of parameters)
- **Expressive** (can approximate any function)
- Complex **architectures** for exploiting problem structure
- Yet, **easy to optimize** to zero training error with (stochastic) gradient descent!

Which function does optimization find?

Understanding deep learning

The challenge of deep learning theory

- **Over-parameterized** (millions of parameters)
- **Expressive** (can approximate any function)
- Complex **architectures** for exploiting problem structure
- Yet, **easy to optimize** to zero training error with (stochastic) gradient descent!

Which function does optimization find?

A functional space viewpoint

- View deep networks as functions in some functional space
- Non-parametric models, natural measures of complexity (*e.g.*, norms)
- Optimization performs **implicit regularization** towards

$$\min_f \Omega(f) \quad \text{s.t.} \quad y_i = f(x_i), \quad i = 1, \dots, n$$

Understanding deep learning

The challenge of deep learning theory

- **Over-parameterized** (millions of parameters)
- **Expressive** (can approximate any function)
- Complex **architectures** for exploiting problem structure
- Yet, **easy to optimize** to zero training error with (stochastic) gradient descent!

Which function does optimization find?

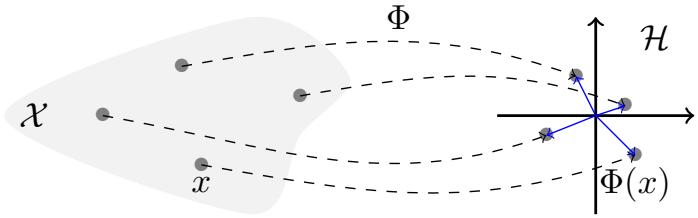
A functional space viewpoint

- View deep networks as functions in some functional space
- Non-parametric models, natural measures of complexity (*e.g.*, norms)
- Optimization performs **implicit regularization** towards

$$\min_f \Omega(f) \quad \text{s.t.} \quad y_i = f(x_i), \quad i = 1, \dots, n$$

What is an appropriate functional space / norm?

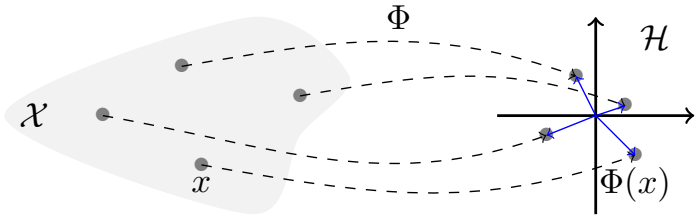
Kernels to the rescue



Kernels?

- Map data x to high-dimensional space, $\Phi(x) \in \mathcal{H}$ ($\mathcal{H}$: "RKHS")
- Functions $f \in \mathcal{H}$ are linear in features: $f(x) = \langle f, \Phi(x) \rangle$ (f can be non-linear in x!)
- Learning with a positive definite kernel $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$
  - $\mathcal{H}$ can be infinite-dimensional! (*kernel trick*)
  - Need to compute kernel matrix $K = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{N \times N}$

Kernels to the rescue



Clean and well-developed theory

- Tractable methods (convex optimization)
- Statistical and approximation properties well understood for many kernels
- Costly (kernel matrix of size N^2) but approximations are possible

Kernels for deep models: deep kernel machines

Hierarchical kernels (Cho and Saul, 2009)

- Kernels can be constructed **hierarchically**

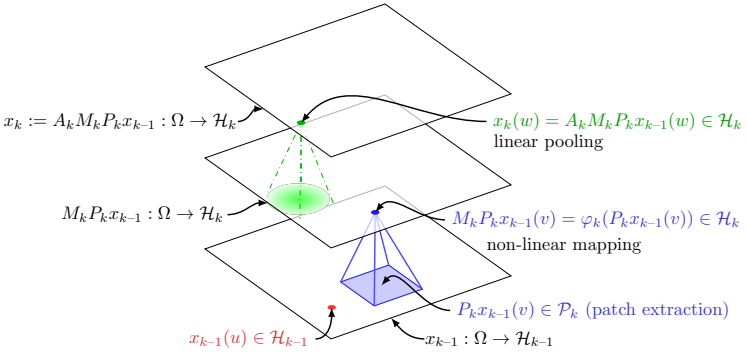
$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle \text{ with } \Phi(x) = \varphi_2(\varphi_1(x))$$

- *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, 2016)



- Good empirical performance with tractable approximations (Nyström)

Kernels for deep models: infinite-width networks

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

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

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

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): 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

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

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): 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 $m \to \infty \approx$ kernel ridge regression with **neural tangent kernel**

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

Kernels for deep models: infinite-width networks

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

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): 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 $m \to \infty \approx$ kernel ridge regression with **neural tangent kernel**

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

RF and NTK extend to deep architectures

Approximation and regularization with kernels

What functions can we represent with a bounded RKHS norm?

Translation-invariant kernel: $K(x, y) = \kappa(x - y)$

$$\|f\|_{\mathcal{H}}^2 = \frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} \frac{|\hat{f}(\omega)|^2}{\hat{\kappa}(\omega)} d\omega.$$

Approximation and regularization with kernels

What functions can we represent with a bounded RKHS norm?

Translation-invariant kernel: $K(x, y) = \kappa(x - y)$

$$\|f\|_{\mathcal{H}}^2 = \frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} \frac{|\hat{f}(\omega)|^2}{\hat{\kappa}(\omega)} d\omega.$$

Dot-product kernel: $K(x, y) = \kappa(\langle x, y \rangle)$ (*e.g.*, RF/NTK for fully-connected networks)

$$f(x) = \sum_{k \geq 0} \sum_{j=1}^{N(d,k)} f_{k,j} Y_{k,j}(x), \quad x \in \mathbb{S}^{d-1}$$

$$\|f\|_{\mathcal{H}}^2 = \sum_{k \geq 0} \sum_{j=1}^{N(d,k)} \frac{f_{k,j}^2}{\mu_k(\kappa)}.$$

Example: polynomials decays $\mu_k \sim k^{-2\beta}$ (*e.g.*, ReLU RF/NTK) $\Leftrightarrow \|f\|_{\mathcal{H}} \approx \|\nabla^\beta f\|_{L^2(\mathbb{S}^{d-1})}$

Issues with fully-connected kernels

- Architecture/depth plays minor role (Bietti and Bach, 2021)
- Curse of dimensionality (unless target is very smooth)

Issues with fully-connected kernels

- Architecture/depth plays minor role (Bietti and Bach, 2021)
- Curse of dimensionality (unless target is very smooth)

Learning on high-dimensional signals/images

- Architecture is important (*e.g.*, for stability, Bietti and Mairal, 2019a)
- **Q: How do convolutional kernels break the curse of dimensionality?**

Simple structured kernels on patches

- Signal x: "pixels" $x[u] \in \mathbb{R}^p$ for $u \in \Omega = \{0, \dots, |\Omega| - 1\}$ (Ω circular)
- Patches $x_u = (x[u + v])_{v \in S} \in \mathbb{R}^{p|S|}$, *e.g.*, $S = \{-1, 0, 1\}$
- Dot-product kernel k on patches with RKHS $\mathcal{H}$

Simple structured kernels on patches

- Signal x: "pixels" $x[u] \in \mathbb{R}^p$ for $u \in \Omega = \{0, \dots, |\Omega| - 1\}$ (Ω circular)
- Patches $x_u = (x[u + v])_{v \in S} \in \mathbb{R}^{p|S|}$, *e.g.*, $S = \{-1, 0, 1\}$
- Dot-product kernel k on patches with RKHS $\mathcal{H}$

One-layer convolutional kernel, no pooling

$$K(x, x') = \sum_{u \in \Omega} k(x_u, x'_u)$$

Functions in RKHS: $f(x) = \sum_u g_u(x_u)$. **Penalty**: $\|g\|^2_{L^2(\Omega, \mathcal{H})} = \sum_u \|g_u\|^2_{\mathcal{H}}$

Simple structured kernels on patches

- Signal x: "pixels" $x[u] \in \mathbb{R}^p$ for $u \in \Omega = \{0, \dots, |\Omega| - 1\}$ (Ω circular)
- Patches $x_u = (x[u+v])_{v \in S} \in \mathbb{R}^{p|S|}$, *e.g.*, $S = \{-1, 0, 1\}$
- Dot-product kernel k on patches with RKHS $\mathcal{H}$

One-layer convolutional kernel, no pooling

$$K(x, x') = \sum_{u \in \Omega} k(x_u, x'_u)$$

Functions in RKHS: $f(x) = \sum_u g_u(x_u)$. **Penalty**: $\|g\|^2_{L^2(\Omega, \mathcal{H})} = \sum_u \|g_u\|^2_{\mathcal{H}}$

- Much smaller RKHS than fully-connected architecture!
- For target $f^*(x) = g^*(x_{\bar{u}})$, breaks curse of dimensionality ($p|\Omega| \to p|S|$)
- Still missing: invariance, expressivity on larger visual neighborhoods

One-layer convolutional kernel with pooling

- $h[u]$: pooling filter (*e.g.*, Gaussian)
- A_h: (circular) convolution operator $A_h x[u] = \sum_{v \in \Omega} h[u-v]x[v]$
- $\Phi(x)[u] = \varphi(x_u) \in \mathcal{H}$ (φ: kernel mapping of k)

1-layer convolutional kernel

$$K(x, x') = \sum_{u \in \Omega} \sum_{v, v'} h[u-v]h[u-v']k(x_v, x'_{v'}) = \langle A_h \Phi(x), A_h \Phi(x') \rangle_{L^2(\Omega, \mathcal{H})}$$

Functions in RKHS: Same functions $f(x) = \sum_u g[u](x_u)$, different **penalty**: $\|A_h^\dagger g\|^2_{L^2(\Omega,\mathcal{H})}$.

One-layer convolutional kernel with pooling

- $h[u]$: pooling filter (*e.g.*, Gaussian)
- A_h: (circular) convolution operator $A_h x[u] = \sum_{v \in \Omega} h[u-v]x[v]$
- $\Phi(x)[u] = \varphi(x_u) \in \mathcal{H}$ (φ: kernel mapping of k)

1-layer convolutional kernel

$$K(x, x') = \sum_{u \in \Omega} \sum_{v, v'} h[u-v]h[u-v']k(x_v, x'_{v'}) = \langle A_h \Phi(x), A_h \Phi(x') \rangle_{L^2(\Omega, \mathcal{H})}$$

Functions in RKHS: Same functions $f(x) = \sum_u g[u](x_u)$, different **penalty**: $\|A_h^\dagger g\|^2_{L^2(\Omega,\mathcal{H})}$.

$\implies$ Encourages spatial smoothness: for $g_z[u] := g[u](z)$, we have

$$\widehat{A_h^\dagger g_z}[w] = \frac{\hat{g}_z[w]}{\hat{h}[w]}$$

Large pooling $\leftrightarrow$ fast decay of $\hat{h}[w]$ $\leftrightarrow$ stronger penalty on high frequencies of g_z.

Generalization benefits of pooling

Translation-invariant target $f^*(x) = \sum_u g(x_u)$, with $g \in \mathcal{H}$.

Learn using kernel K_h with pooling with filter $h \geq 0$, $\|h\|_1 = 1$, *e.g.*:

- **no pooling**: $h[u] = \delta_{u,0}$
- **global pooling**: $h[u] = 1/|\Omega|$
- $A_h(g, \ldots, g) = (g, \ldots, g) \implies$ **same RKHS norm** for any h!

Generalization benefits of pooling

Translation-invariant target $f^*(x) = \sum_u g(x_u)$, with $g \in \mathcal{H}$.

Learn using kernel K_h with pooling with filter $h \geq 0$, $\|h\|_1 = 1$, *e.g.*:

- **no pooling**: $h[u] = \delta_{u,0}$
- **global pooling**: $h[u] = 1/|\Omega|$
- $A_h(g, \dots, g) = (g, \dots, g) \implies$ **same RKHS norm** for any h!

Basic generalization bound with 1-Lipschitz loss on $\mathcal{F} = \{\|f\|_{K_h} \leq B\}$

$$\mathbb{E}\, L(f_n) - \min_{f \in \mathcal{F}} L(f) \lesssim \frac{B\sqrt{\mathbb{E}_x[K_h(x,x)]}}{\sqrt{n}}$$

Under simple data models, $\mathbb{E}_x[k(x_u, x_u)] = 1$, $\mathbb{E}_x[k(x_u, x_v)] \leq \epsilon \ll 1$ for $u \neq v$

- **no pooling**: $\mathbb{E}[K_h(x,x)] = |\Omega|$
- **global pooling**: $\mathbb{E}[K_h(x,x)] \leq 1 + \epsilon|\Omega|$

Generalization benefits of pooling

Translation-invariant target $f^*(x) = \sum_u g(x_u)$, with $g \in \mathcal{H}$.

Learn using kernel K_h with pooling with filter $h \geq 0$, $\|h\|_1 = 1$, *e.g.*:

- **no pooling**: $h[u] = \delta_{u,0}$
- **global pooling**: $h[u] = 1/|\Omega|$
- $A_h(g, \dots, g) = (g, \dots, g) \implies$ **same RKHS norm** for any h!

Basic generalization bound with 1-Lipschitz loss on $\mathcal{F} = \{\|f\|_{K_h} \leq B\}$

$$\mathbb{E}\, L(f_n) - \min_{f \in \mathcal{F}} L(f) \lesssim \frac{B\sqrt{\mathbb{E}_x[K_h(x,x)]}}{\sqrt{n}}$$

Under simple data models, $\mathbb{E}_x[k(x_u, x_u)] = 1$, $\mathbb{E}_x[k(x_u, x_v)] \leq \epsilon \ll 1$ for $u \neq v$

- **no pooling**: $\mathbb{E}[K_h(x,x)] = |\Omega|$
- **global pooling**: $\mathbb{E}[K_h(x,x)] \leq 1 + \epsilon|\Omega| \implies$ **need $\sim |\Omega|$ fewer samples!**

Generalization benefits of pooling

Translation-invariant target $f^*(x) = \sum_u g(x_u)$, with $g \in \mathcal{H}$.

Learn using kernel K_h with pooling with filter $h \geq 0$, $\|h\|_1 = 1$, *e.g.*:

- **no pooling**: $h[u] = \delta_{u,0}$
- **global pooling**: $h[u] = 1/|\Omega|$
- $A_h(g, \dots, g) = (g, \dots, g) \implies$ **same RKHS norm** for any h!

Basic generalization bound with 1-Lipschitz loss on $\mathcal{F} = \{\|f\|_{K_h} \leq B\}$

$$\mathbb{E}\, L(f_n) - \min_{f \in \mathcal{F}} L(f) \lesssim \frac{B\sqrt{\mathbb{E}_x[K_h(x,x)]}}{\sqrt{n}}$$

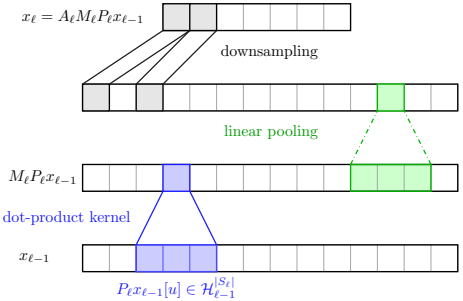
Under simple data models, $\mathbb{E}_x[k(x_u, x_u)] = 1$, $\mathbb{E}_x[k(x_u, x_v)] \leq \epsilon \ll 1$ for $u \neq v$

- **no pooling**: $\mathbb{E}[K_h(x,x)] = |\Omega|$
- **global pooling**: $\mathbb{E}[K_h(x,x)] \leq 1 + \epsilon|\Omega| \implies$ **need $\sim |\Omega|$ fewer samples!**
- General h: $\mathbb{E}[K_h(x,x)] \leq |\Omega|\|h\|_2^2 + \epsilon|\Omega|(1 - \|h\|_2^2)$

More layers

- Same construction, replace $x \in L^2(\Omega, \mathbb{R}^p)$ by $A\Phi(x) \in L^2(\Omega, \mathcal{H})$
- Dot-product kernel can still be defined with inputs in a Hilbert space

$$K(x, x') = \langle \Psi(x), \Psi(x') \rangle, \qquad \text{with } \Psi(x) = A_L M_L P_L \cdots A_1 M_1 P_1 x$$



P_ℓ with patch shape S_ℓ, A_ℓ with Gaussian pooling h_ℓ at scale σ_ℓ

Some experiments on Cifar10

2-layers, 3x3 patches, pooling/downsampling sizes (2,5). Patch kernels κ_1, κ_2.

κ_1	κ_2	Test acc. (10k examples)	Test acc. (50k examples)
Exp	Exp	80.5%	87.9% (84.1%)
Exp	Poly3	80.5%	87.7% (84.1%)
Exp	Poly2	79.4%	86.9% (83.4%)
Poly2	Exp	77.4%	- (81.5%)
Poly2	Poly2	75.1%	- (81.2%)
Exp	- (Lin)	74.2%	- (76.3%)

In parentheses: Nyström approximation of the kernel (Mairal, 2016) with [256,4096] filters, instead of the full kernel.

Two-layer convolutional kernel

- Quadratic patch kernel $k_2(z, z') = (z^\top z')^2 = \langle z \otimes z, z' \otimes z' \rangle_{(\mathcal{H} \otimes \mathcal{H})^{|S_2| \times |S_2|}}$
- $\mathcal{H} \otimes \mathcal{H}$: contains functions $g(x_u, x_v)$ of 2 patches (Wahba, 1990)

Two-layer convolutional kernel

- Quadratic patch kernel $k_2(z, z') = (z^\top z')^2 = \langle z \otimes z, z' \otimes z' \rangle_{(\mathcal{H}\otimes\mathcal{H})^{|S_2|\times|S_2|}}$
- $\mathcal{H} \otimes \mathcal{H}$: contains functions $g(x_u, x_v)$ of 2 patches (Wahba, 1990)

RKHS of 2-layer convolutional kernel (patch size $|S_2| = 1$): Contains functions

$$f(x) = \sum_{u,v\in\Omega} G[u, v](x_u, x_v),$$

with $G[u, v] = 0$ if $|u - v| > \text{diam}(\text{supp}(h_1))$. **Penalty**:

$$\|A_2^\dagger \,\text{diag}((A_1 \otimes A_1)^\dagger G)\|^2_{L^2(\Omega_2, \mathcal{H}\otimes\mathcal{H})}$$

Two-layer convolutional kernel

- Quadratic patch kernel $k_2(z, z') = (z^\top z')^2 = \langle z \otimes z, z' \otimes z' \rangle_{(\mathcal{H}\otimes\mathcal{H})^{|S_2|\times|S_2|}}$
- $\mathcal{H} \otimes \mathcal{H}$: contains functions $g(x_u, x_v)$ of 2 patches (Wahba, 1990)

RKHS of 2-layer convolutional kernel (patch size $|S_2| = 1$): Contains functions

$$f(x) = \sum_{u,v\in\Omega} G[u, v](x_u, x_v),$$

with $G[u, v] = 0$ if $|u - v| > \text{diam}(\text{supp}(h_1))$. **Penalty**:

$$\|A_2^\dagger \,\text{diag}((A_1 \otimes A_1)^\dagger G)\|^2_{L^2(\Omega_2, \mathcal{H}\otimes\mathcal{H})}$$

- $(A_1 \otimes A_1)^\dagger$: encourages 2D smoothness of "image" $G[u, v]$, bandwidth σ_1
- $A_2^\dagger$: encourage 1D smoothness along diagonal of G, bandwidth σ_2
- $\sigma_1 > \sigma_2 \implies G[u, v]$ can depend more strongly on $u - v$ than u or v

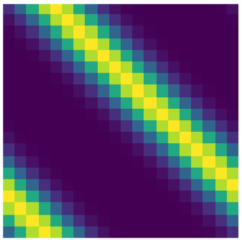
Two-layer convolutional kernel

RKHS of 2-layer convolutional kernel (any patch size $|S_2|$): Contains functions

$$f(x) = \sum_{p,q \in S_2} \sum_{u,v \in \Omega} G_{pq}[u,v](x_u, x_v),$$

with $G_{pq}[u,v] = 0$ if $|u - v - (p - q)| > \mathrm{diam}(\mathrm{supp}(h_1))$. **Penalty**:

$$\sum_{p,q \in S_2} \|A_2^\dagger \, \mathrm{diag}((L_p A_1 \otimes L_q A_1)^\dagger G_{pq})\|^2_{L^2(\Omega_2, \mathcal{H} \otimes \mathcal{H})}$$



Two-layer convolutional kernel

RKHS of 2-layer convolutional kernel (any patch size $|S_2|$): Contains functions

$$f(x) = \sum_{p,q \in S_2} \sum_{u,v \in \Omega} G_{pq}[u,v](x_u, x_v),$$

with $G_{pq}[u,v] = 0$ if $|u - v - (p - q)| > \mathrm{diam}(\mathrm{supp}(h_1))$. **Penalty**:

$$\sum_{p,q \in S_2} \|A_2^\dagger \, \mathrm{diag}((L_p A_1 \otimes L_q A_1)^\dagger G_{pq})\|^2_{L^2(\Omega_2, \mathcal{H} \otimes \mathcal{H})}$$

"Variance" term: $\sqrt{\mathbb{E}_x[K(x,x)]} \leq |\Omega||S_2|^2 \sum_v \langle h_2, L_v h_w \rangle \langle h_1, L_v h_1 \rangle^2 + O(\epsilon)$

Two-layer convolutional kernel

RKHS of 2-layer convolutional kernel (any patch size $|S_2|$): Contains functions

$$f(x) = \sum_{p,q \in S_2} \sum_{u,v \in \Omega} G_{pq}[u,v](x_u, x_v),$$

with $G_{pq}[u,v] = 0$ if $|u - v - (p - q)| > \mathrm{diam}(\mathrm{supp}(h_1))$. **Penalty**:

$$\sum_{p,q \in S_2} \|A_2^\dagger \, \mathrm{diag}((L_p A_1 \otimes L_q A_1)^\dagger G_{pq})\|^2_{L^2(\Omega_2, \mathcal{H} \otimes \mathcal{H})}$$

"Variance" term: $\sqrt{\mathbb{E}_x[K(x,x)]} \leq |\Omega||S_2|^2 \sum_v \langle h_2, L_v h_w \rangle \langle h_1, L_v h_1 \rangle^2 + O(\epsilon)$

Extensions:

- κ_2 higher-degree polynomial $\implies$ higher-order interactions
- more layers: also higher-order interactions, but more structured penalty

More experiments: 3 layers

3-layers, 3x3 patches, pooling/downsampling sizes (2,2,2), patch kernels κ_1, κ_2 and κ_3.

κ_1	κ_2	κ_3	Test ac. (10k)	Test ac. (50k)
Exp	Exp	Exp	80.7%	**88.2%**
Exp	Poly2	Poly2	80.5%	87.9%
Exp	Poly4	Lin	80.2%	-
Exp	Lin	Poly4	79.2%	-
Exp	Lin	Lin	74.1%	-

Shankar et al. (2020) also obtains 88.2% but using 10 layers

More experiments: larger patches

2-layers, 3x3 patches at first layer, pooling/downsampling sizes (2,5)
patch kernels κ_1/κ_2 and different patch sizes at the second layer.

κ_1	κ_2	$\lvert S_2\rvert$	Test ac. (10k)	Test ac. (50k)
Exp	Exp	5x5	81.1%	**88.3%**
Exp	Poly4	5x5	81.3%	**88.3%**
Exp	Poly3	5x5	81.1%	-
Exp	Poly2	7x7	80.1%	-
Exp	Poly2	5x5	80.1%	-
Exp	Poly2	3x3	79.4%	-
Exp	Poly2	1x1	76.3%	-
ReLU	ReLU	3x3	78.51%	86.6%
NTK	(NTK)	3x3	79.24%	87.2%

Conclusion and perspectives

Learning with convolutional kernels

- Additive interaction models on patches
- Pooling is important for encouraging spatial regularities, and better sample complexity
- 2/3 layers seems enough on Cifar10

Conclusion and perspectives

Learning with convolutional kernels

- Additive interaction models on patches
- Pooling is important for encouraging spatial regularities, and better sample complexity
- 2/3 layers seems enough on Cifar10

What is missing compared to full DL?

- **Feature selection**: avoid dependence on dimensionality (of the patch) by finding good weights at the first layer
  - Bach (2017)
  - Finding good filters (e.g. Gabors) could also avoid the need for explicit pooling
- **Hierarchy**: efficiently find hierarchical **compositions** of functions
  - Allen-Zhu and Li (2020): SGD can do it under strong assumptions

References I

Z. Allen-Zhu and Y. Li. Backward feature correction: How deep learning performs deep learning. *arXiv preprint arXiv:2001.04413*, 2020.

F. Bach. Breaking the curse of dimensionality with convex neural networks. *Journal of Machine Learning Research (JMLR)*, 18(19):1–53, 2017.

A. Bietti and F. Bach. Deep equals shallow for relu networks in kernel regimes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

A. Bietti and J. Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations. *Journal of Machine Learning Research (JMLR)*, 20(25):1–49, 2019a.

A. Bietti and J. Mairal. On the inductive bias of neural tangent kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b.

J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(8):1872–1886, 2013.

L. Chizat, E. Oyallon, and F. Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Y. Cho and L. K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

References II

A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Mairal. End-to-End Kernel Learning with Supervised Convolutional Kernel Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

S. Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10): 1331–1398, 2012.

R. M. Neal. *Bayesian learning for neural networks*. Springer, 1996.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

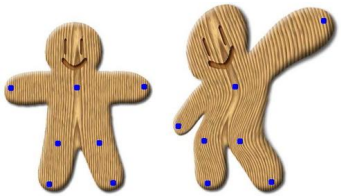
V. Shankar, A. Fang, W. Guo, S. Fridovich-Keil, L. Schmidt, J. Ragan-Kelley, and B. Recht. Neural kernels without tangents. *arXiv preprint arXiv:2003.02237*, 2020.

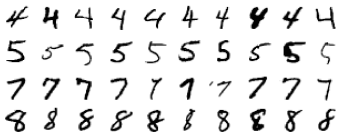
G. Wahba. *Spline models for observational data*, volume 59. Siam, 1990.

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





- Studied for wavelet-based scattering transform (Mallat, 2012; Bruna and Mallat, 2013)

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: $f(x) = \langle f, \Phi(x) \rangle$

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

- $\|f\|_{\mathcal{H}}$ 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, 2016) with efficient approximations
- Extends to neural tangent kernels (**NTKs**, Jacot et al., 2018) of infinitely wide CNNs (Bietti and Mairal, 2019b)

Construction of convolutional kernels

Construct a sequence of feature maps $x_1, \dots, x_n$

- $x_0 : \Omega \to \mathcal{H}_0$: initial (**continuous**) signal
  - $u \in \Omega = \mathbb{R}^d$: location ($d = 2$ for images)
  - $x_0(u) \in \mathcal{H}_0$: value ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images)

Construction of convolutional kernels

Construct a sequence of feature maps $x_1, \dots, x_n$

- $x_0 : \Omega \to \mathcal{H}_0$: initial (**continuous**) signal
  - $u \in \Omega = \mathbb{R}^d$: location ($d = 2$ for images)
  - $x_0(u) \in \mathcal{H}_0$: value ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images)
- $x_k : \Omega \to \mathcal{H}_k$: **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

Construction of convolutional kernels

Construct a sequence of feature maps $x_1, \dots, x_n$

- $x_0 : \Omega \to \mathcal{H}_0$: initial (**continuous**) signal
  - $u \in \Omega = \mathbb{R}^d$: location ($d = 2$ for images)
  - $x_0(u) \in \mathcal{H}_0$: value ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images)
- $x_k : \Omega \to \mathcal{H}_k$: **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
  - M_k: **non-linear mapping** operator, maps each patch to a new point with a **pointwise** non-linear function $\varphi_k(\cdot)$ (kernel mapping)

Construction of convolutional kernels

Construct a sequence of feature maps $x_1, \dots, x_n$

- $x_0 : \Omega \to \mathcal{H}_0$: initial (**continuous**) signal
  - $u \in \Omega = \mathbb{R}^d$: location ($d = 2$ for images)
  - $x_0(u) \in \mathcal{H}_0$: value ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images)
- $x_k : \Omega \to \mathcal{H}_k$: **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
  - M_k: **non-linear mapping** operator, maps each patch to a new point with a **pointwise** non-linear function $\varphi_k(\cdot)$ (kernel mapping)
  - A_k: (linear, Gaussian) **pooling** operator at scale σ_k

Construction of convolutional kernels

Construct a sequence of feature maps $x_1, \dots, x_n$

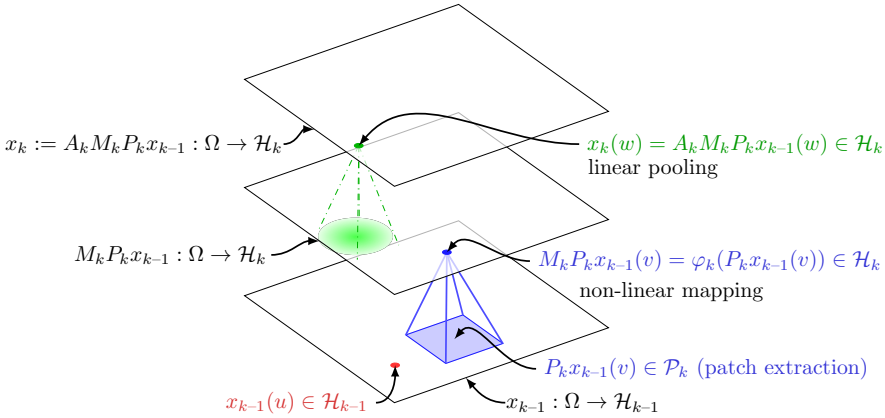
- $x_0 : \Omega \to \mathcal{H}_0$: initial (**continuous**) signal
  - $u \in \Omega = \mathbb{R}^d$: location ($d = 2$ for images)
  - $x_0(u) \in \mathcal{H}_0$: value ($\mathcal{H}_0 = \mathbb{R}^3$ for RGB images)
- $x_k : \Omega \to \mathcal{H}_k$: **feature map** at layer k

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

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

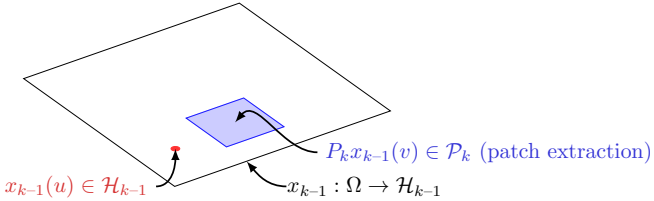
Goal: control stability of these operators through their norms

CKN construction



Patch extraction operator P_k

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



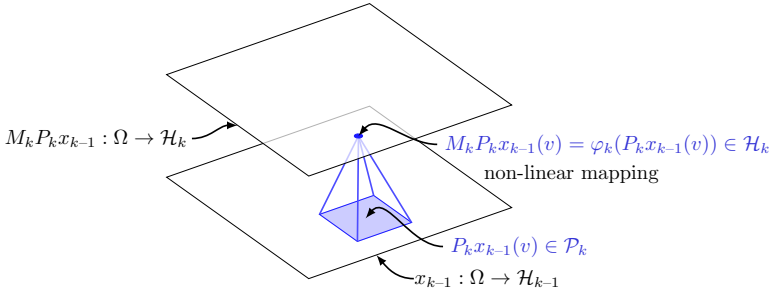
Patch extraction operator P_k

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

- S_k: patch shape, e.g. box
- P_k is **linear**, and **preserves the L^2 norm**: $\|P_k x_{k-1}\| = \|x_{k-1}\|$

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

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

- $\varphi_k : \mathcal{P}_k \to \mathcal{H}_k$ pointwise non-linearity on patches (kernel map)
- We assume **non-expansivity**: for $z, z' \in \mathcal{P}_k$

$$\|\varphi_k(z)\| \leq \|z\| \quad \text{and} \quad \|\varphi_k(z) - \varphi_k(z')\| \leq \|z - z'\|$$

- M_k then satisfies, for $x, x' \in L^2(\Omega, \mathcal{P}_k)$

$$\|M_k x\| \leq \|x\| \quad \text{and} \quad \|M_k x - M_k x'\| \leq \|x - x'\|$$

φ_k from kernels

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

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

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

- Commonly used for hierarchical kernels
- $\|\varphi_k(z)\| = K_k(z, z)^{1/2} = \|z\|$
- $\|\varphi_k(z) - \varphi_k(z')\| \leq \|z - z'\|$ if $\kappa_k'(1) \leq 1$
- $\implies$ **non-expansive**

φ_k from kernels

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

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

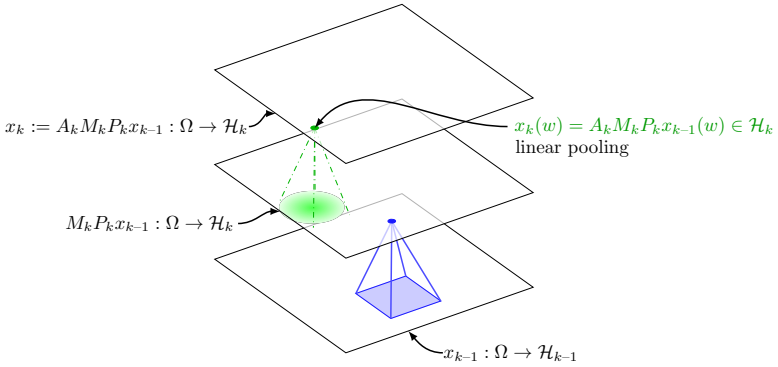
$\kappa_k(u) = \sum_{j=0}^{\infty} b_j u^j$ with $b_j \geq 0$, $\kappa_k(1) = 1$

Examples

- $\kappa_{\exp}(\langle z, z'\rangle) = e^{\langle z, z'\rangle - 1}$ (Gaussian kernel on the sphere)
- $\kappa_{\text{inv-poly}}(\langle z, z'\rangle) = \frac{1}{2 - \langle z, z'\rangle}$
- $\kappa_\sigma(\langle z, z'\rangle) = \mathbb{E}_w[\sigma(w^\top z)\sigma(w^\top z')]$ (Random features)
  - arc-cosine kernels for the ReLU $\sigma(u) = \max(0, u)$

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

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

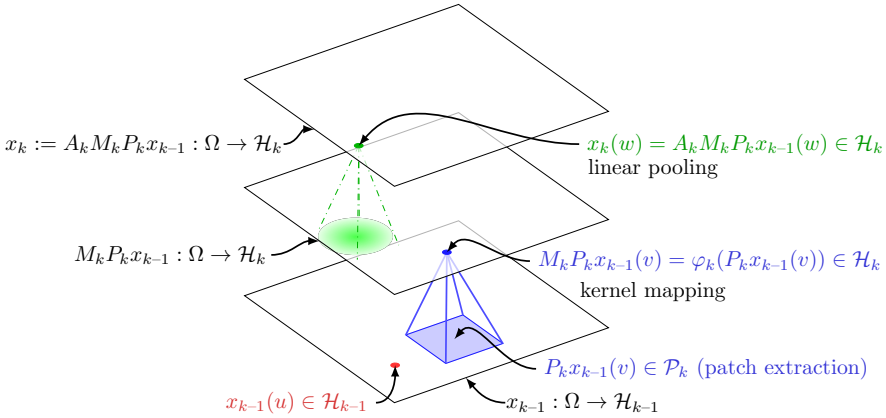
- h_{σ_k}: pooling filter at scale σ_k
- $h_{\sigma_k}(u) := \sigma_k^{-d} h(u/\sigma_k)$ with $h(u)$ **Gaussian**
- **linear, non-expansive operator**: $\|A_k\| \leq 1$

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



Multilayer construction

Assumption on x_0

- x_0 is typically a **discrete** signal aquired 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**).

Multilayer construction

Assumption on x_0

- x_0 is typically a **discrete** signal aquired 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**).

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).

Multilayer construction

Assumption on x_0

- x_0 is typically a **discrete** signal aquired 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**).

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).

Final kernel

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

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

  $\Longrightarrow$ **need several layers with small patches** $\quad n = O(\log(\sigma_n/\sigma_0)/\log \beta)$

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

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

Comparison with random feature CKN on deformed MNIST digits:

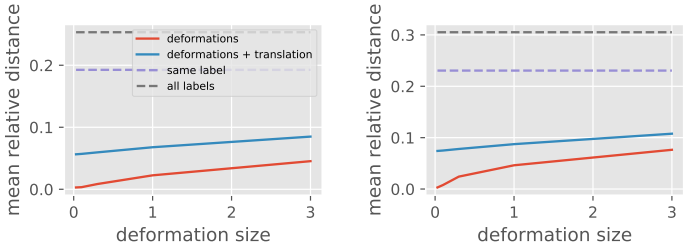
Stability to deformations for convolutional NTK

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

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

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

Comparison with random feature CKN on deformed MNIST digits:



(a) CKN (b) NTK