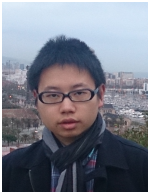
From Graph Kernels to Graph Transformers

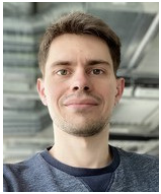
and a bonus story about learning rates

Julien Mairal

Inria, Univ. Grenoble-Alpes

Collaborators


Dexiong Chen


Gregoire Mialon


Emmanuel Jehanno

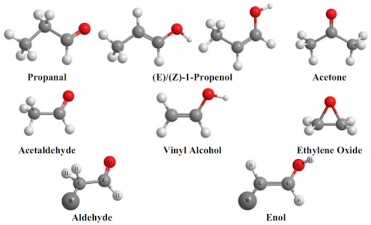

Margot Selosse


Romain Menegaux

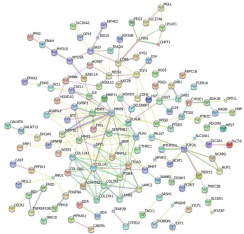

Laurent Jacob

- D. Chen, L. Jacob and J. Mairal. Convolutional Kernel Networks for Graph-Structured Data. *ICML*. 2020.
- G. Mialon, D. Chen, M. Selosse, and J. Mairal. GraphiT: Encoding Graph Structure in Transformers. *arXiv:2106.05667*. 2021.
- R. Menegaux, E. Jehanno, M. Selosse and J. Mairal. Self-Attention in Colors: Another Take on Encoding Graph Structure in Transformers. *TMLR*. 2023.

Graph-structured data is everywhere

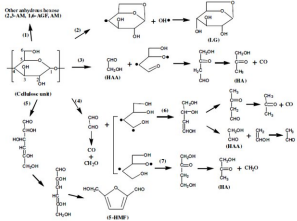


(a) molecules



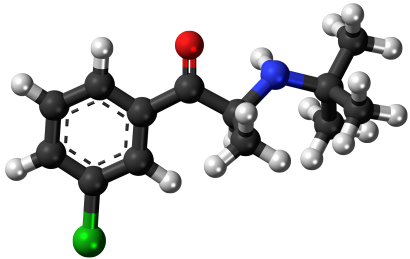
(b) protein regulation

(c) social networks



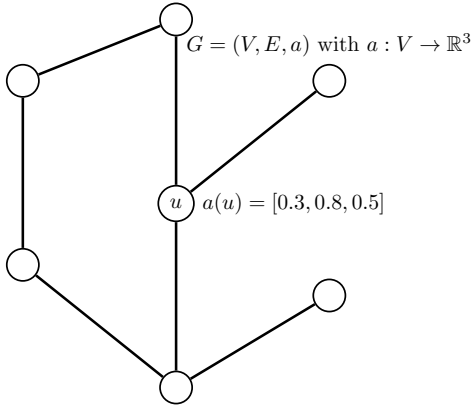
(d) chemical pathways

Learning graph representations: challenges



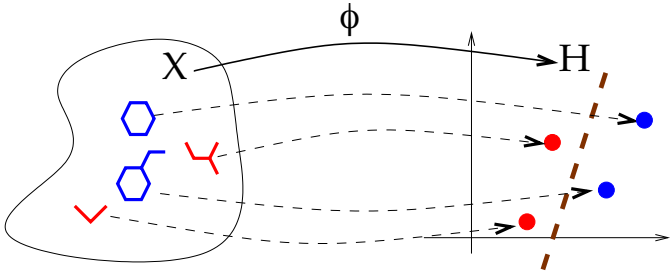
- **Expressiveness**: Find a representation (vector) that is able to discriminate graphs with different structures (distinguish non-isomorphic graphs, to some extent).
- **Tractability**: The representation should be efficiently computable on modern hardware.
- **Learnable**: One should be able to adapt the representation to the task and to the data.
- **Taking into account physics**: long-range potentials, 3D geometry, symmetries...

Graphs with node attributes



- We consider graphs $G = (V, E, a)$ where V and E are the sets of vertices and edges, and $a : V \to \mathbb{R}^p$ is a function assigning attributes to each node.

Graph kernel mappings

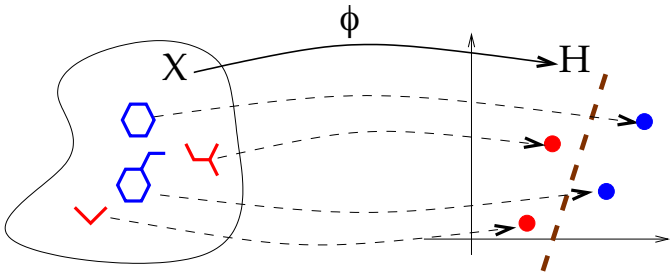


- Map each graph G to a vector $\Phi(G)$ in $\mathcal{H}$, which lends itself to learning tasks.
- A large class of graph kernel mappings can be written in the form

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

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

Graph kernel mappings



- Map each graph G 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

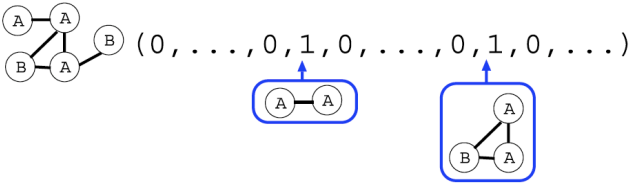
$$K(G, G') = \Big\langle \underbrace{\sum_{u \in V} \varphi_{\text{base}}(\ell_G(u))}_{\Phi(G)}, \underbrace{\sum_{u' \in V'} \varphi_{\text{base}}(\ell_{G'}(u'))}_{\Phi(G')} \Big\rangle .$$

Kernel representations with substructure enumeration

Find a high-dimensional representation $\Phi(G)$ in $\mathcal{H}$ for which we can efficiently compute

$$K(G, G') = \langle \Phi(G), \Phi(G') \rangle_{\mathcal{H}}.$$

There is a very rich literature about graph kernels performing (implicitly or explicitly) **substructure enumeration**.



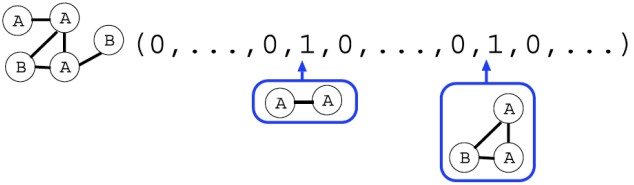
- **subgraphs and path** kernels (NP-hard, [Gärtner et al., 2003]).
- **walk** kernels [Kashima et al., 2003, Mahé et al., 2004].
- **shortest-path** kernels [Borgwardt and Kriegel, 2005].
- **graphlets** kernels [Shervashidze et al., 2009].
- **Weisfeiler-Lehman** kernels [Shervashidze et al., 2011].

Kernel representations with substructure enumeration

Find a high-dimensional representation $\Phi(G)$ in $\mathcal{H}$ for which we can efficiently compute

$$K(G, G') = \langle \Phi(G), \Phi(G') \rangle_{\mathcal{H}}.$$

There is a very rich literature about graph kernels performing (implicitly or explicitly) **substructure enumeration**.



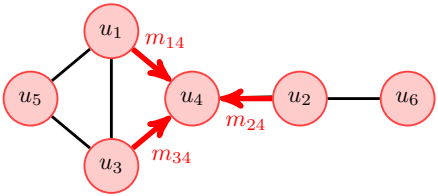
For a review, see the course material

- `https://mva-kernel-methods.github.io/course-2023-2024/`

Learning graph representations with deep learning

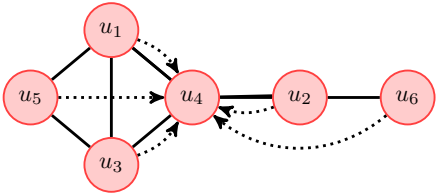
Graph neural networks with message passing

- multi-layer construction.
- sequence of local operations.
- limited expressivity [Xu et al., 2018].



Graph transformers

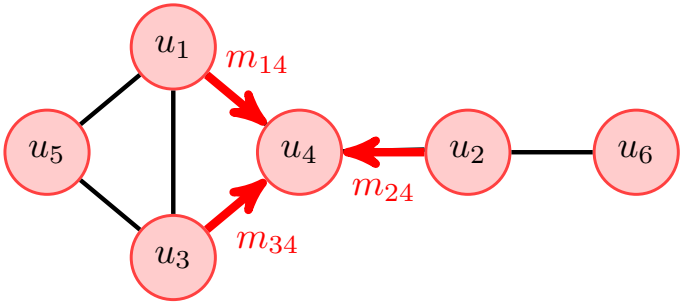
- non-local operations with attention.
- challenge: **encoding the graph structure**.



For a detailed review, see

- survey on graph transformers: [Müller et al., 2023].
- course material from Xavier Bresson `https://lnkd.in/dZZWay3Z`.

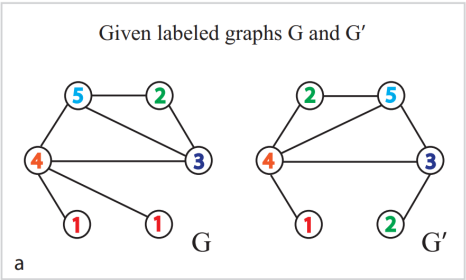
From Weisfeiler-Lehman to graph neural networks and an abstract multilayer graph kernel

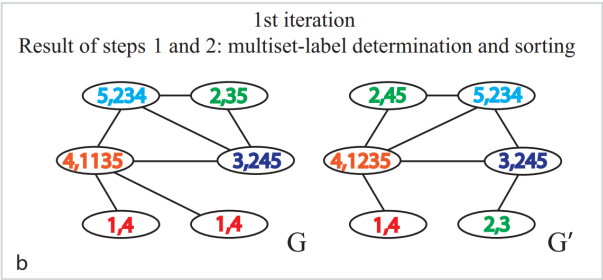


Principles of the Weisfeiler-Lehman kernel

Consider a graph $G = (V, E, a)$ with discrete labels $l_0(u) = a(u)$ at each vertex u.

- This is a **multi-layer** construction producing new labels $l_k(u)$ for each vertex at layer k.
- A label $l_k(u)$ represents $(l_{k-1}(u), \{l_{k-1}(v) : v \in \mathcal{N}(u)\})$.
- Based on the **graph isomorphism test** of Weisfeiler and Lehman, 1968.



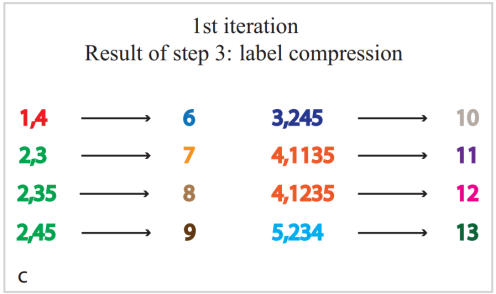


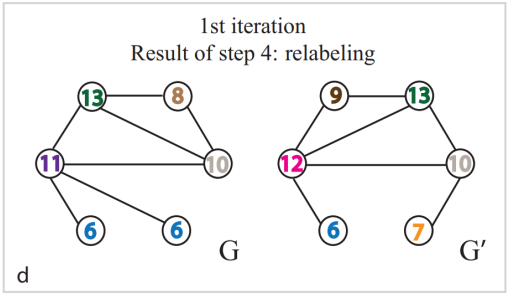
Pictures courtesy of Shervashidze et al. [2011].

Principles of the Weisfeiler-Lehman kernel

Consider a graph $G = (V, E, a)$ with discrete labels $l_0(u) = a(u)$ at each vertex u.

- This is a **multi-layer** construction producing new labels $l_k(u)$ for each vertex at layer k.
- A label $l_k(u)$ represents $(l_{k-1}(u), \{l_{k-1}(v) : v \in \mathcal{N}(u)\})$.
- Based on the **graph isomorphism test** of Weisfeiler and Lehman, 1968.

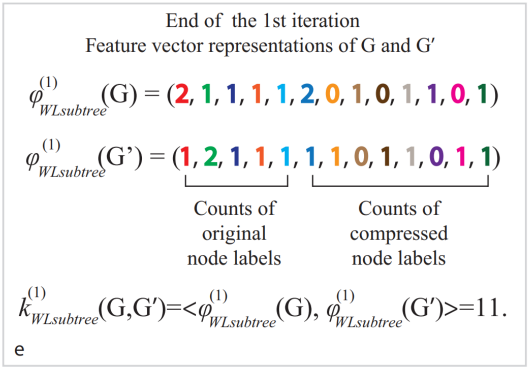




Pictures courtesy of Shervashidze et al. [2011].

Principles of the Weisfeiler-Lehman kernel

- The final representation is a histogram of **label occurences.**
- Extensions with **substructure enumeration**.



Pictures courtesy of Shervashidze et al. [2011].

Principles of the Weisfeiler-Lehman kernel

Given a graph $G = (V, E, a)$ with discrete labels $l_0(u) = a(u)$ in $\mathcal{A}_0$ for all u in V.

The Weisfeiler-Lehmann kernel representation

- **Representation at layer** k: Label $l_k(u) \in \mathcal{A}_k$ for all u in V.
- **Construction of layer** k (**message passing**):

$$l_k(u) = \text{Relabel}(l_{k-1}(u), \{l_{k-1}(v) : v \in \mathcal{N}(u)\}).$$

- **Last layer representation** with **global aggregation**:

$$\Phi_{\text{WL}}(G) = \sum_{v \in V} \text{one-hot-encoding}(l_K(u)) \in \mathbb{R}^{|\mathcal{A}|}.$$

Principles of graph neural networks with message passing

Given a graph $G = (V, E, a)$ with continous attributes $f_0(u) = a(u)$ in $\mathbb{R}^{p_0}$ for all u in V.

Canonical form of message passing architecture

- **Representation at layer** k: $f_k(u) \in \mathbb{R}^{p_k}$ for all u in V.
- **Construction of layer** k (**message passing**):

$$\begin{aligned} f_k(u) &= \mathsf{Process}(f_{k-1}(u), \{f_{k-1}(v) : v \in \mathcal{N}(u)\}) \\ &= \sum_{v \in \mathcal{N}(u) \cup u} \mathsf{ReLU}(Z_k^\top f_{k-1}(v)) \quad \text{(for example)} \end{aligned}$$

- **Last layer representation** with **global pooling**:

$$f_{\mathsf{GNN}}(G) = \sum_{v \in V} f_K(u) \in \mathbb{R}^{p_K}.$$

An abstract multi-layer graph kernel [Chen et al., 2020]

Consider graph $G = (V, E, a)$ with attributes $\varphi_0(u) = a(u)$ living in some RKHS $\mathcal{H}_0$.

Multilayer kernel construction

- **Representation at layer** k: $\varphi_k(u) \in \mathcal{H}_k$ for all u in V.
- **Construction of layer** k (message passing):
  - Define a kernel K_k on features from layer $k-1$. Call $\mathcal{H}_k$ its RKHS and $\phi_k : \mathcal{H}_{k-1} \to \mathcal{H}_k$ the corresponding kernel mapping.
  - Aggregate with **message passing**

$$\varphi_k(u) = \sum_{v \in \mathcal{N}(u) \cup u} \phi_k(\varphi_{k-1}(v)).$$

- **Last layer representation** with **global pooling**:

$$\Phi_{\text{MLGK}}(G) = \sum_{v \in V} \varphi_K(u) \in \mathcal{H}_K.$$

An abstract multi-layer graph kernel [Chen et al., 2020]

Dot-product or RBF kernels are natural candidates for K_k:

$$K(a,b) = \quad \kappa(\langle a,b\rangle) \quad \text{or} \quad \|a\|\|b\|\kappa\left(\left\langle \frac{a}{\|a\|}, \frac{b}{\|b\|}\right\rangle\right) \quad \text{or} \quad e^{-\alpha\|a-b\|^2}.$$

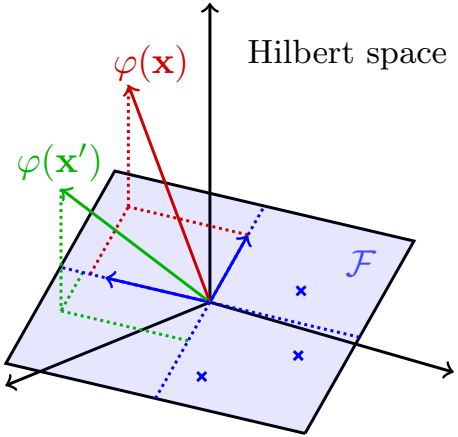
An abstract multi-layer graph kernel [Chen et al., 2020]

Dot-product or RBF kernels are natural candidates for K_k:

$$K(a,b) = \quad \kappa(\langle a,b\rangle) \quad \text{or} \quad \|a\|\|b\|\kappa\left(\left\langle \frac{a}{\|a\|}, \frac{b}{\|b\|}\right\rangle\right) \quad \text{or} \quad e^{-\alpha\|a-b\|^2}.$$

- Is the resulting kernel $K(G,G') = \langle \Phi(G), \Phi(G')\rangle$ easily **computable**?
- Can we improve its **expressiveness**?
- Can we make it **"trainable"**?

The Nyström approximation for dot-product kernels and its connection to neural networks



Nyström aproximation for dot-product kernels

Consider a dot product kernel $K(a, b) = \kappa(a, b)$ where a, b are in $\mathbb{R}^p$.

an approximate finite-dimensional embedding: The Nyström method [Williams and Seeger, 2001] provides a function $\psi : \mathbb{R}^p \to \mathbb{R}^d$ such that

$$\kappa(a, b) = \langle \phi(a), \phi(b) \rangle_{\mathcal{H}} \approx \langle \psi(a), \psi(b) \rangle_{\mathbb{R}^d}.$$

Geometric interpretation: The method performs orthogonal projections onto a finite-dimensional subspace spanned by some anchor points $\phi(z_1), \ldots, \phi(z_d)$ in $\mathcal{H}$.

Analytical formula: Given the anchor points $z_1, \ldots, z_d$ in $\mathbb{R}^p$,

$$\psi(a) = \kappa(Z^\top Z)^{-1/2} \kappa(Z^\top a),$$

where $Z = [z_1, \ldots, z_d]$ is in $\mathbb{R}^{p \times d}$.

Nyström aproximation for dot-product kernels

$$\psi(a) = \kappa(Z^\top Z)^{-1/2}\kappa(Z^\top a),$$

How to find good anchor points Z?

- **random data samples**: the original method.
- **unsupervised learning**: set the anchor points as the centroids of a K-means algorithm on observed patterns [Zhang et al., 2008].
- **back-propagation**: This embedding may be interpreted as a neural network layer which is compatible with end-to-end learning [Mairal, 2016]. Given a p.d. symmetric matrix A,

$$d(A^{-1/2}) = -U(F \circ (U^\top (dA) U))U^\top \quad \text{where} \quad A = U\Delta U^\top$$

$$\text{and} \quad F_{kl} = \frac{1}{\sqrt{\delta_k}\sqrt{\delta_l}(\sqrt{\delta_k} + \sqrt{\delta_l})}$$

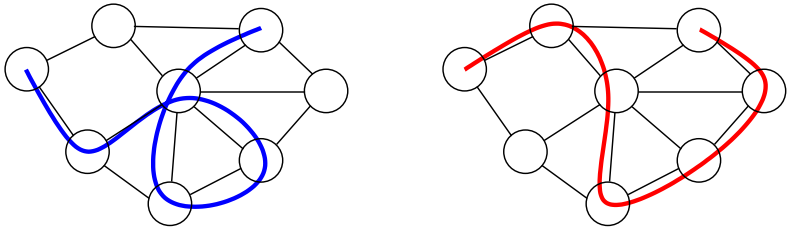
Graph convolutional kernel networks [Chen et al., 2020]

- the abstract **multilayer graph kernel** for continuous attributes.
- $+$ the **Nyström approximation** to make everything computable and learnable.
- $+$ an extension with **paths enumeration** to gain expressiveness.

$=$

- **kernel viewpoint:** an **unsupervised** (relatively high-dimensional) representation performing well with two layers on benchmarks from 2020.
- **deep learning viewpoint**: a **learnable** lower-dimensional representation performing equally well (and probably better on large datasets).

Basic kernels: walk and path kernels



- Path kernels are more **expressive** than walk kernels, but less preferred for **computational** reasons.
- $\mathcal{P}_L(G, u) :=$ paths of length L from node u in G.
- $\Phi(G)$ can be interpreted as a **histogram** of paths occurrences (label sequences);

An abstract multi-layer graph kernel [Chen et al., 2020]

Consider graph $G = (V, E, a)$ with attributes $\varphi_0(u) = a(u)$ living in some RKHS $\mathcal{H}_0$.

Multilayer kernel construction

- **Representation at layer** k: $\varphi_k(u) \in \mathcal{H}_k$ for all u in V.
- **Construction of layer** k (message passing):
  - Define a kernel K_k on features from layer $k-1$. Call $\mathcal{H}_k$ its RKHS and $\phi_k : \mathcal{H}_{k-1} \to \mathcal{H}_k$ the corresponding kernel mapping.
  - Aggregate with **message passing**

$$\varphi_k(u) = \sum_{v \in \mathcal{N}(u) \cup u} \phi_k(\varphi_{k-1}(v)).$$

- **Last layer representation** with **global pooling**:

$$\Phi_{\text{MLGK}}(G) = \sum_{v \in V} \varphi_K(u) \in \mathcal{H}_K.$$

An abstract multi-layer path kernel [Chen et al., 2020]

Consider graph $G = (V, E, a)$ with attributes $\varphi_0(u) = a(u)$ living in some RKHS $\mathcal{H}_0$.

Multilayer kernel construction

- **Representation at layer** k: $\varphi_k(u) \in \mathcal{H}_k$ for all u in V.
- **Construction of layer** k (message passing):
  - Define a kernel K_k on paths of length L from layer $k-1$. Call $\mathcal{H}_k$ its RKHS and $\phi_k : \mathcal{H}_{k-1}^L \to \mathcal{H}_k$ the corresponding kernel mapping.
  - Aggregate with **message passing**

$$\varphi_k(u) = \sum_{p \in \mathcal{P}_L(G,u)} \phi_k^{\text{path}}(p).$$

- **Last layer representation** with **global pooling**:

$$\Phi_{\text{MLPK}}(G) = \sum_{v \in V} \varphi_K(u) \in \mathcal{H}_K.$$

The graph convolutional kernel network model [Chen et al., 2020]

Consider graph $G = (V, E, a)$ with attributes $\psi_0(u) = a(u)$ living in $\mathbb{R}^{p_0}$.

Multilayer kernel construction

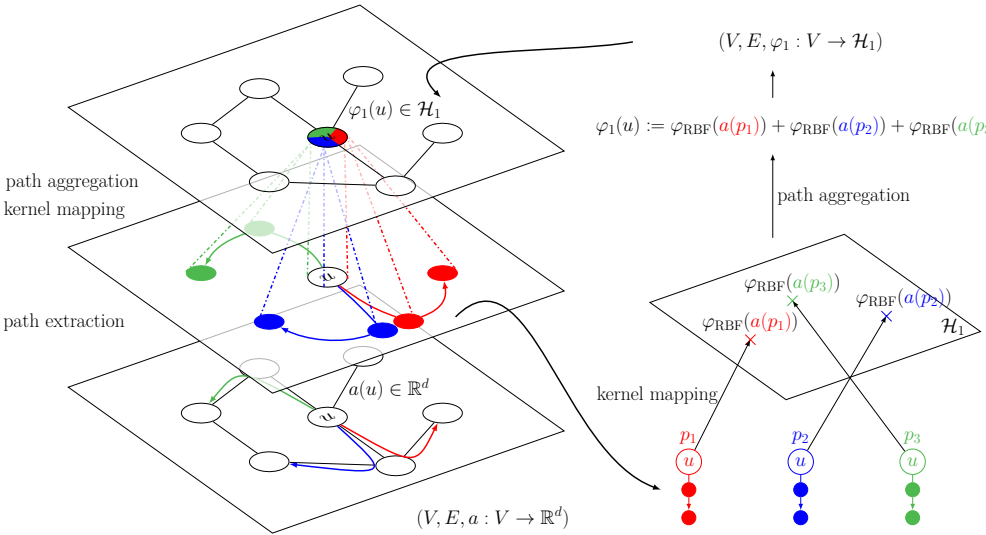
- **Representation at layer** k: $\psi_k(u) \in \mathbb{R}^{p_k}$ for all u in V.
- **Construction of layer** k (message passing):
  - Define a kernel K_k on paths of length L from layer $k-1$. Call $\mathcal{H}_k$ its RKHS and $\psi_k^{\text{path}} : \mathbb{R}^{p_{k-1}{}^L} \to \mathbb{R}^{p_k}$ the Nyström approximation with parameters Z_k.
  - Aggregate with **message passing**

$$\psi_k(u) = \sum_{p \in \mathcal{P}_L(G,u)} \psi_k^{\text{path}}(p).$$

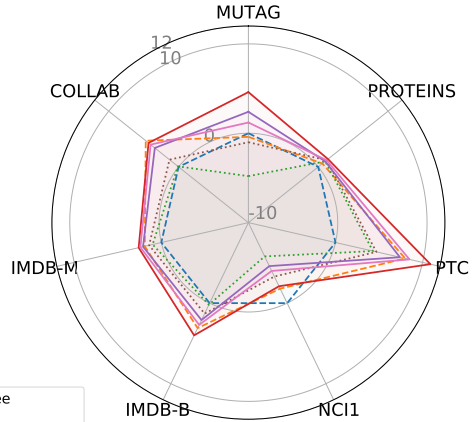
- **Last layer representation** with **global pooling**:

$$\Psi_{\text{GCKN}}(G) = \sum_{v \in V} \psi_K(u) \in \mathbb{R}^{p_K}.$$

Construction of one-layer GCKN



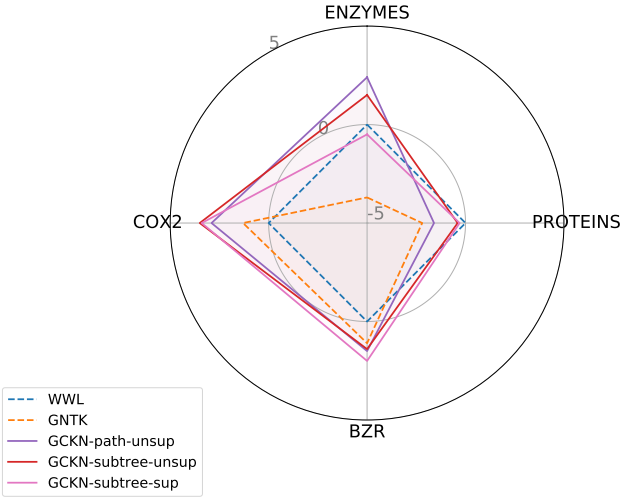
Benchmark on graphs with discrete attributes (2020)



- 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]

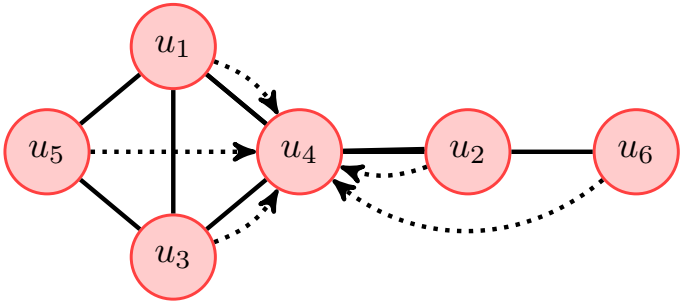
Benchmarks on graphs with continuous attributes (2020)



- Accuracy improvement with respect to the WWL kernel.
- Results similar to discrete case.

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

Graph transformers, GraphiT and CSA



- G. Mialon, D. Chen, M. Selosse, and J. Mairal. GraphiT: Encoding Graph Structure in Transformers. *arXiv:2106.05667*. 2021.
- R. Menegaux, E. Jehanno, M. Selosse and J. Mairal. Self-Attention in Colors: Another Take on Encoding Graph Structure in Transformers. *TMLR*. 2023.

From GNNs to Graph transformers

An example of GNN layer (GCN, Kipf and Welling, 2017)

$$f_k(u) = \mathsf{ReLU}\left(Z^\top\left(\frac{1}{|\mathcal{N}(u)|+1}\sum_{v\in\mathcal{N}(u)\cup u} f_{k-1}(v)\right)\right).$$

From GNNs to Graph transformers

An example of GNN layer (GCN, Kipf and Welling, 2017)

$$f_k(u) = \mathsf{ReLU}\left(Z^\top\left(\frac{1}{|\mathcal{N}(u)|+1}\sum_{v\in\mathcal{N}(u)\cup u} f_{k-1}(v)\right)\right).$$

The basic transformer layer with self attention

$$f_k(u) = \mathsf{ReLU}\left(Z^\top\left(f_{k-1}(u) + \sum_{v\in V} A_{uv} f_{k-1}(u)\right)\right)$$

$$\text{with} \quad A = \mathsf{Softmax}\left(\frac{f_{k-1}W_Q^\top W_K f_{k-1}^\top}{\sqrt{d}}\right).$$

(Note that a classical residual connection has been removed for simplicity).

From GNNs to Graph transformers

The basic transformer layer with self attention

$$f_k(u) = \text{ReLU}\left(Z^\top\left(f_{k-1}(u) + \sum_{v \in V} A_{uv} f_{k-1}(u)\right)\right)$$

$$\text{with} \quad A = \text{Softmax}\left(\frac{f_{k-1} W_Q^\top W_K f_{k-1}^\top}{\sqrt{d}}\right).$$

Challenges

- How to encode the graph structure?
- How to take into account edge features?

Graph transformers: recipes

How to take into account edge features?

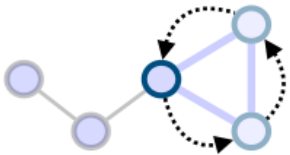
- treat edge features as node features with additional variables $E_k(u, v)$ undergoing "similar" updates.

Local structure encoding

- Enrich input features. A successful feature is based on the diagonals of random walk kernels

$$p(u) = [RW_{uu}, \dots, RW^p_{uu}]$$

where RW^p_{uu} probability for a p-step random walk to loop back to node u:



[Dwivedi and Bresson, 2020, Rampášek et al., 2022, Lim et al., 2022]

Graph transformers: recipes

Modulate the attention matrix with relative positional encoding

- Graphormer computes an average of the dot-products of edge feature and a learnable embedding along shortest paths

$$A = \text{Softmax}\left(\frac{f_{k-1} W_Q^\top W_K f_{k-1}^\top}{\sqrt{d}} + B_k^{\text{shortest-paths}}\right).$$

- GraphiT weights the attention with a diffusion kernel. This captures both short-range and long-range graph topology

$$A = \text{Normalize}\left(\text{Exp}\left(\frac{f_{k-1} W_Q^\top W_K f_{k-1}^\top}{\sqrt{d}}\right) \circ K_\sigma\right).$$

[Ying et al., 2021, Mialon et al., 2021]

Graph transformers: recipes

Modulate the attention matrix with relative positional encoding

- GraphiT uses a hard-coded kernel and does not include edge features in the attention.
- CSA first enriches original edge features with random walks kernels:

$$E_{uv}^{\mathsf{rw}} = [RW_{uv}, \ldots, RW_{uv}^{p}]$$

and then learns how to exploit these features to modulate the attention matrix

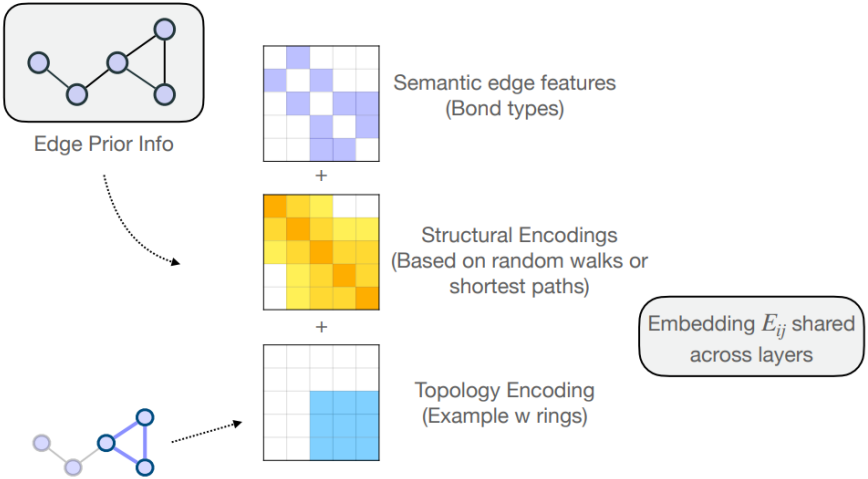
$$A = \mathsf{Softmax}\left(\frac{f_{k-1}W_Q^\top W_K f_{k-1}^\top}{\sqrt{d}} + W_E^\top E_{k-1}\right).$$

Additional tricks

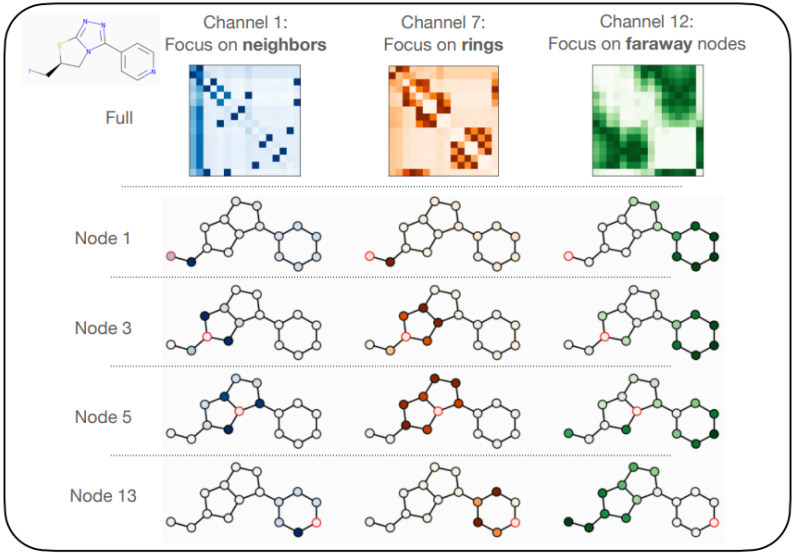
- introduce features for structures that are known to be useful (carbon rings).

[Menegaux et al., 2023]

All of this summarized in a pretty picture



Visualizing self attention



Benchmarks

	Model	ZINC MAE ↓
MPNN	GCN (Kipf & Welling, 2017)	0.367 ± 0.011
	GatedGCN (Dwivedi et al., 2022a)	0.090 ± 0.001
	GPS (Rampášek et al., 2022)	0.070 ± 0.004
h-MPNN	CIN (Bodnar et al., 2021a)	0.079 ± 0.006
	CRaWL (Toenshoff et al., 2021)	0.085 ± 0.004
	GIN-AK+ (Zhao et al., 2022)	0.080 ± 0.001
Transformers	SAN (Kreuzer et al., 2021)	0.139 ± 0.006
	Graphormer (Ying et al., 2021)	0.122 ± 0.006
	SAT (Chen et al., 2022)	0.094 ± 0.008
	EGT (Hussain et al., 2022)	0.108 ± 0.009
	GRPE (Park et al., 2022)	0.094 ± 0.002
	CSA (ours)	$\mathbf{0.070 \pm 0.003}$
	CSA-rings (ours)	$\mathbf{0.056 \pm 0.002}$

(12k graphs)

Benchmarks

	Model	PCQM4Mv2 (4M graphs)	
		Validation MAE ↓	# Param.
MPNN	GCN	0.1379	2.0M
	GCN-virtual	0.1153	4.9M
	GIN	0.1195	3.8M
	GIN-virtual	0.1083	6.7M
Transformers	Graphormer	0.0864	48.3M
	EGT	0.0869	89.3M
	GRPE	0.0890	46.2M
	GPS-small	0.0938	6.2M
	GPS-medium	0.0858	19.4M
	CSA-small (ours)	0.0898	2.8M
	CSA-deep (ours)	**0.0853**	8.3M

Just in case: diffusion kernels

- Given a function f on the graph with Laplacian L, $f^\top L f = \sum_{u \sim v} (f_u - f_v)^2$ can be interpreted as a way to quantify the smoothness of f.
- The Laplacian is often used via its eigenvalue decomposition $L = \sum_i \lambda_i u_i u_i^\top$.
- A whole family of kernels on graphs are defined as $K_r = \sum_i r(\lambda_i) u_i u_i^\top$.
- With $r(\lambda) = e^{-\beta\lambda}$, we obtain the **diffusion kernel**:

$$K_r = e^{-\beta L} = \lim_{p \to +\infty} \left(I - \frac{\beta}{p} L \right)^p.$$

- With $r(\lambda) = (I - \gamma L)^p$, we obtain the p-step random walk kernel

$$K_r = \left(I - \frac{\beta}{p} L \right)^p.$$

[Smola and Kondor, 2003, Kondor and Lafferty, 2002, Belkin and Niyogi, 2003]

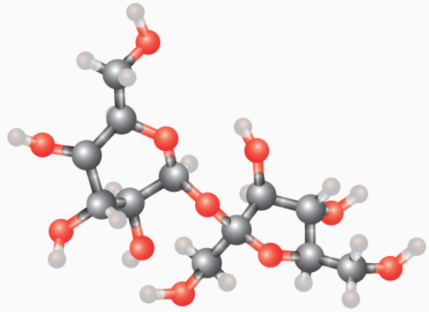
Conclusion

- Many ideas from the kernel world naturally appear in the graph neural networks literature, in particular for transformers.
- Encoding prior information within graph transformers makes a difference, in particular for medium-sized datasets.

Ongoing Work and Challenges: Physics and Geometry

Ex: Molecular graphs (*e.g.*, ZINC or OGB datasets)

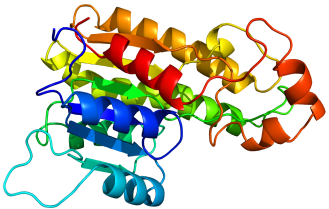
- Nodes are *atoms*, edges are *bonds*.
- Node features can be *atom-type, spatial position*, ...
- Edge features are bond types (*single,double, triple*).

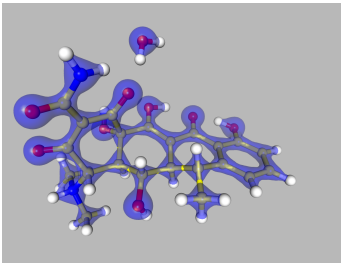


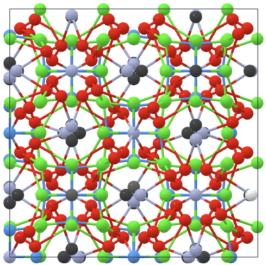
Ongoing Work and Challenges: Physics and Geometry

Challenges

- Is there another structure within the graph? (*e.g.*, chain of amino acids for proteins).
- Is the graph part of a larger structure (crystallography)?
- Does the representation model the right symmetries and inv/equivariances?
- Is the graph construction satisfactory? What about long-range potentials?







Ongoing Work and Challenges: Physics and Geometry

Challenges

- Is there another structure within the graph? (*e.g.*, chain of amino acids for proteins).
- Is the graph part of a larger structure (crystallography)?
- Does the representation model the right symmetries and inv/equivariances?
- Is the graph construction satisfactory? What about long-range potentials?

Useful material

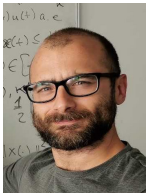
- see the survey on graph neural networks for 3D atomic systems [Duval et al., 2023].

Bonus Story:

On the Benefits of Large Learning Rates for Kernel Methods

Slides courtesy of Gaspard Beugnot

Gaspard Beugnot



Alessandro Rudi

- G. Beugnot, J. Mairal, and A. Rudi. On the Benefits of Large Learning Rates for Kernel Methods. International Conference on Learning Theory (COLT). 2022.

In brief

- **Motivation**: common choice of learning rate for SGD in deep learning results in **poor optimization** but provides **better generalization**

In brief

- **Motivation**: common choice of learning rate for SGD in deep learning results in **poor optimization** but provides **better generalization**
- **Approach**: a simple convex model where the training loss/generalization error are **quadratic functions in a RKHS**. This extends an intuition from Nakkiran [2020] on a 2D toy problem.

In brief

- **Motivation**: common choice of learning rate for SGD in deep learning results in **poor optimization** but provides **better generalization**
- **Approach**: a simple convex model where the training loss/generalization error are **quadratic functions in a RKHS**. This extends an intuition from Nakkiran [2020] on a 2D toy problem.
- **Contribution**: predicts when there are benefits for generalization by taking large step sizes (close to $2/L$). Notable example : kernel ridge regression for classification.

Losses in Machine Learning

1. Usual settings: minimize **empirical loss**

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_{\text{train}}(\theta(x_i), y_i) + \Omega(\theta) \quad \text{(Optimization)}$$

Rationale: F is a proxy for the real downstream task = minimize the **generalization error** R:

$$R(\theta) = \mathbb{E}_{x,y}\left[\ell_{\text{test}}(\theta(x), y)\right] \quad \text{(Statistics)}$$

$\Longrightarrow$ what if there are big **discrepancies** between F and R?

[1] Agnostic to supervised learning

Losses in Machine Learning

1. Usual settings: minimize **empirical loss**

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell_{\text{train}}(\theta(x_i), y_i) + \Omega(\theta) \quad \text{(Optimization)}$$

Rationale: F is a proxy for the real downstream task = minimize the **generalization error** R:

$$R(\theta) = \mathbb{E}_{x,y}\left[\ell_{\text{test}}(\theta(x), y)\right] \quad \text{(Statistics)}$$

$\Longrightarrow$ what if there are big **discrepancies** between F and R?

2. Modelization: two **quadratic functions**[1] in $\mathcal{H}$:

$$F(\theta) = \frac{1}{2}\left\|\theta - \theta^\star\right\|_{\mathsf{T}}^2 + \text{cst}, \quad \text{and} \quad R(\theta) = \frac{1}{2}\left\|\theta - \nu^\star\right\|_{\mathsf{U}}^2, \tag{1}$$

[1] Agnostic to supervised learning

Quadratic functions? Example with Ridge Regression

$x_1, \ldots, x_n$: data points in $\mathbb{R}^d$; $y_1, \ldots, y_n$ prediction variables; $X \in \mathbb{R}^{n \times d}$: data matrix.
Ridge regression estimator:

$$\forall \theta \in \mathbb{R}^d, \quad F(\theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (\theta^\top x_i - y_i)^2 + \frac{\lambda}{2} \|\theta\|^2$$

Quadratic functions? Example with Ridge Regression

$x_1, \ldots, x_n$: data points in $\mathbb{R}^d$; $y_1, \ldots, y_n$ prediction variables; $X \in \mathbb{R}^{n \times d}$: data matrix.
Ridge regression estimator:

$$\begin{aligned}\forall \theta \in \mathbb{R}^d, \quad F(\theta) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\theta^\top x_i - y_i)^2 + \frac{\lambda}{2}\|\theta\|^2 \\ &= \frac{1}{2}\|\theta - \theta^\star\|_{\mathsf{T}}^2 + \text{cst}, \quad \text{with} \quad \mathsf{T} = \frac{1}{n}(X^\top X + \lambda \mathbf{I}_d).\end{aligned}$$

Quadratic functions? Example with Ridge Regression

$x_1, \ldots, x_n$: data points in $\mathbb{R}^d$; $y_1, \ldots, y_n$ prediction variables; $X \in \mathbb{R}^{n \times d}$: data matrix.
Ridge regression estimator:

$$\begin{aligned}\forall \theta \in \mathbb{R}^d, \quad F(\theta) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\theta^\top x_i - y_i)^2 + \frac{\lambda}{2}\|\theta\|^2 \\ &= \frac{1}{2}\|\theta - \theta^\star\|_{\mathsf{T}}^2 + \text{cst}, \quad \text{with} \quad \mathsf{T} = \frac{1}{n}(X^\top X + \lambda \mathbf{I}_d).\end{aligned}$$

Ok, but what about the statistical risk?

Quadratic functions? Example with Ridge Regression

$x_1, \ldots, x_n$: data points in $\mathbb{R}^d$; $y_1, \ldots, y_n$ prediction variables; $X \in \mathbb{R}^{n \times d}$: data matrix.
Ridge regression estimator:

$$\begin{aligned}\forall \theta \in \mathbb{R}^d, \quad F(\theta) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\theta^\top x_i - y_i)^2 + \frac{\lambda}{2}\|\theta\|^2 \\ &= \frac{1}{2}\|\theta - \theta^\star\|_{\mathsf{T}}^2 + \text{cst}, \quad \text{with} \quad \mathsf{T} = \frac{1}{n}(X^\top X + \lambda \mathbf{I}_d).\end{aligned}$$

Ok, but what about the statistical risk?

- Model: $y_i = x_i^\top \nu^\star + \epsilon_i$ with $\nu^\star \in \mathbb{R}^d$

Quadratic functions? Example with Ridge Regression

$x_1, \ldots, x_n$: data points in $\mathbb{R}^d$; $y_1, \ldots, y_n$ prediction variables; $X \in \mathbb{R}^{n \times d}$: data matrix.
Ridge regression estimator:

$$\begin{aligned}\forall \theta \in \mathbb{R}^d, \quad F(\theta) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\theta^\top x_i - y_i)^2 + \frac{\lambda}{2}\|\theta\|^2 \\ &= \frac{1}{2}\|\theta - \theta^\star\|_{\mathsf{T}}^2 + \text{cst}, \quad \text{with} \quad \mathsf{T} = \frac{1}{n}(X^\top X + \lambda \mathbf{I}_d).\end{aligned}$$

Ok, but what about the statistical risk?

- Model: $y_i = x_i^\top \nu^\star + \epsilon_i$ with $\nu^\star \in \mathbb{R}^d$
- **Population loss**: $\mathcal{P}(\theta) = \mathbb{E}\, 1/2 \left(\theta^\top x - y\right)^2$

Quadratic functions? Example with Ridge Regression

$x_1, \ldots, x_n$: data points in $\mathbb{R}^d$; $y_1, \ldots, y_n$ prediction variables; $X \in \mathbb{R}^{n \times d}$: data matrix.
Ridge regression estimator:

$$\forall \theta \in \mathbb{R}^d, \quad F(\theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (\theta^\top x_i - y_i)^2 + \frac{\lambda}{2} \|\theta\|^2$$
$$= \frac{1}{2} \|\theta - \theta^\star\|_{\mathsf{T}}^2 + \text{cst}, \quad \text{with} \quad \mathsf{T} = \frac{1}{n}(X^\top X + \lambda \mathbf{I}_d).$$

Ok, but what about the statistical risk?

- Model: $y_i = x_i^\top \nu^\star + \epsilon_i$ with $\nu^\star \in \mathbb{R}^d$
- **Population loss**: $\mathcal{P}(\theta) = \mathbb{E}\, 1/2 \left(\theta^\top x - y\right)^2$
- **Excess risk**: $R(\theta) = \mathcal{P}(\theta) - \inf_\nu \mathcal{P}(\nu)$ satisfies

$$R(\theta) = \frac{1}{2} \|\theta - \nu^\star\|_{\mathsf{U}}^2, \quad \text{with} \quad \mathsf{U} = \mathbb{E}\left[x x^\top\right], \quad \nu^\star \text{ regression function.}$$

Mismatch between U and T for classification with kernel ridge regression

Case of interest: Binary classification on a **low-noise** dataset [Pillaud-Vivien et al., 2018]:

- Classes are **well separated by a non-zero margin**.
- the conditional probability $\mathbb{E}[y|x]$ is regular enough.
- $\nu^\star$ (minimizer of the binary classification error $B(\theta)$) is in the RKHS $\mathcal{H}$ with norm $\|.\|$.

Then, $B(\theta) - B(\nu^\star)$ **decreases exponentially** in $\|\theta - \nu^\star\|^2$.

Mismatch between U and T for classification with kernel ridge regression

Case of interest: Binary classification on a **low-noise** dataset [Pillaud-Vivien et al., 2018]:

- Classes are **well separated by a non-zero margin**.
- the conditional probability $\mathbb{E}[y|x]$ is regular enough.
- $\nu^\star$ (minimizer of the binary classification error $B(\theta)$) is in the RKHS $\mathcal{H}$ with norm $\|.\|$.

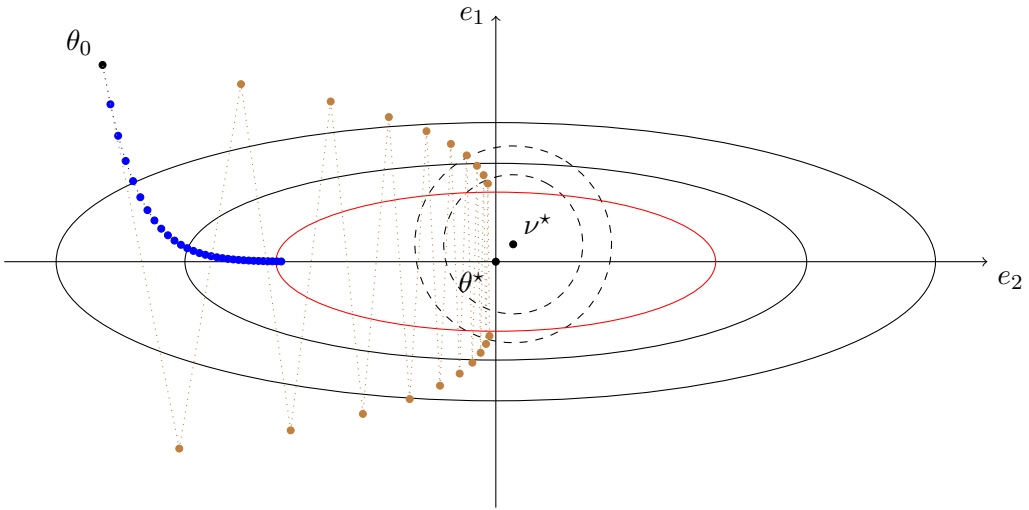
Then, $B(\theta) - B(\nu^\star)$ **decreases exponentially** in $\|\theta - \nu^\star\|^2$.

The dramatic consequence

Thus: big discrepencies between **train loss** F we optimize and **Hilbert norm** R which is a good proxy for the **classification error**!

$$F(\theta) = \frac{1}{2}\|\theta - \theta^\star\|_{\mathsf{T}}^2 \qquad R(\theta) = \frac{1}{2}\|\theta - \nu^\star\|^2$$

Mismatch between U and T for classification with kernel ridge regression



Main result

Main result, informal.

Under assumptions on (1) the operators T and U, (2) the learning rate, (3) the initialization and (4) the target training loss α. Perform GD, with either small LR η_s or big LR η_b, and stop as soon as $F(\theta_t) \leq \alpha$. Then

$$R(\theta_b) - R(\nu^\star) \;\leq\; 34 \frac{\kappa_{\mathsf{U}}}{\kappa_{\mathsf{T}}} (R(\theta_s) - R(\nu^\star)).$$

Main result

Main result, informal.

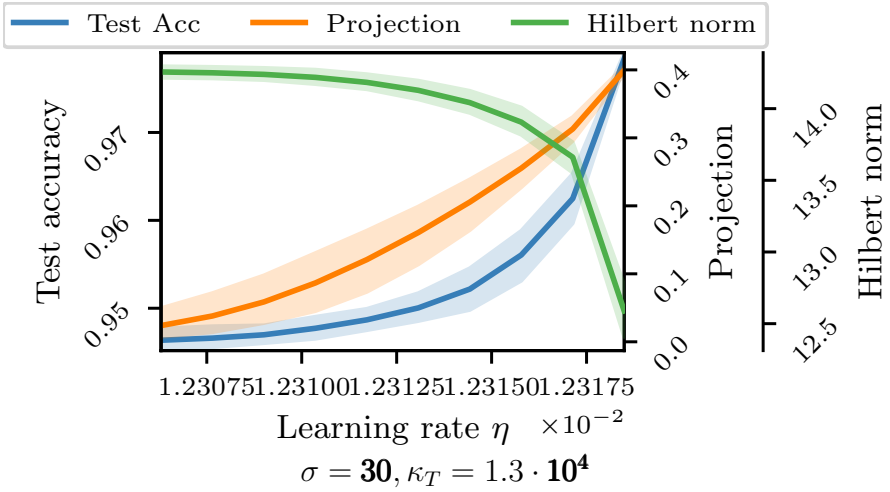
Under assumptions on (1) the operators T and U, (2) the learning rate, (3) the initialization and (4) the target training loss α. Perform GD, with either small LR η_s or big LR η_b, and stop as soon as $F(\theta_t) \leq \alpha$. Then

$$R(\theta_b) - R(\nu^\star) \;\leq\; 34 \frac{\kappa_{\mathsf{U}}}{\kappa_{\mathsf{T}}} (R(\theta_s) - R(\nu^\star)).$$

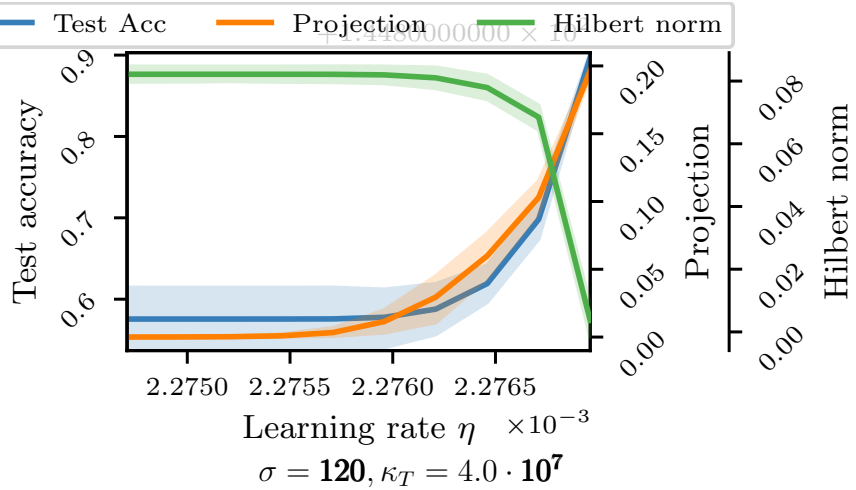
- It reads: "Doing big step size gives you better statistical error as soon as the empirical risk is badly conditioned."
- The worse the conditioning, the bigger the improvement.
- Assumptions (1,3) are loose, assumption (4) is more restrictive.
- There is a tiny range for η_b in Assumption (2):

$$0 < \eta_s < \frac{2}{\sigma_1 + \sigma_n} < \eta_b < \frac{2}{\sigma_1}.$$

Experiments *(Classification on CKN-MNIST with the Gaussian kernel – target accuracy α fixed)*



Experiments *(Classification on CKN-MNIST with the Gaussian kernel – target accuracy α fixed)*



Conclusion

- A simple model which illustrates a well known phenomenon in deep learning and provide a **clear intuition**;
- Highlights the role of the **condition number** of the training loss: the more ill conditioned, the bigger the improvement with big step sizes;
- We do not advocate for using big learning rates in term of generalization/time complexity.
- Natural extensions: different loss functions, local extensions to non-convex loss landscapes, optimization algorithms which amplifies this phenomenon.

References I

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

Dexiong Chen, Laurent Jacob, and Julien Mairal. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning*, pages 1576–1586. PMLR, 2020.

Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Alexandre Duval, Simon V Mathis, Chaitanya K Joshi, Victor Schmidt, Santiago Miret, Fragkiskos D Malliaros, Taco Cohen, Pietro Liò, Yoshua Bengio, and Michael Bronstein. A hitchhiker's guide to geometric gnns for 3d atomic systems. *arXiv preprint arXiv:2312.07511*, 2023.

VP Dwivedi and X Bresson. A generalization of transformer networks to graphs. arxiv. *arXiv preprint arXiv:2012.09699*, 2020.

References II

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *COLT*, pages 129–143. Springer, 2003.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 321–328, 2003.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, volume 2002, pages 315–322, 2002.

Nils M Kriege, Marion Neumann, Christopher Morris, Kristian Kersting, and Petra Mutzel. A unifying view of explicit and implicit feature maps of graph kernels. *Data Mining and Knowledge Discovery*, 33(6):1505–1547, 2019.

References III

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.

Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, and Jean-Philippe Vert. Extensions of marginalized graph kernels. In *Proceedings of the twenty-first international conference on Machine learning*, page 70, 2004.

J. Mairal. End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Romain Menegaux, Emmanuel Jehanno, Margot Selosse, and Julien Mairal. Self-attention in colors: Another take on encoding graph structure in transformers. *Transactions on Machine Learning Research (TMLR)*, 2023.

References IV

Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. *arXiv preprint arXiv:2302.04181*, 2023.

Preetum Nakkiran. Learning rate annealing can provably help generalization, even for convex problems. *arXiv preprint arXiv:2005.07360*, 2020.

Loucas Pillaud-Vivien, Alessandro Rudi, and Francis Bach. Statistical optimality of stochastic gradient descent on hard learning problems through multiple passes. *Advances in Neural Information Processing Systems*, 31, 2018.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pages 488–495. PMLR, 2009.

References V

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12: 2539–2561, 2011.

Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*, pages 144–158. Springer, 2003.

Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein Weisfeiler-Lehman graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Christopher KI Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

References VI

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Kai Zhang, Ivor W Tsang, and James T Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.