Structured Data Modeling with Deep Kernel Machines and Applications in Computational Biology

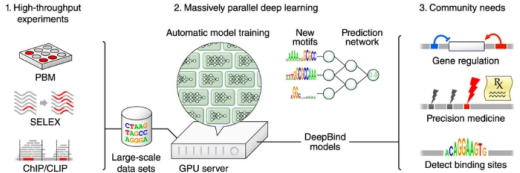
Dexiong Chen

Inria Grenoble

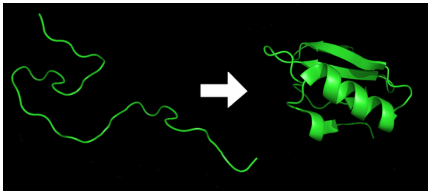
December 15, 2020

Advisors: Julien Mairal (Inria), Laurent Jacob (CNRS/U. Lyon 1)
Reviewers: Chloé-Agathe Azencott (Mines ParisTech), Karsten Borgwardt (ETH Zürich)
Examiners: Florence d'Alché-Buc (Télécom Paris), Arthur Gretton (UCL)

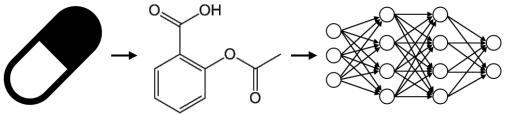
Some success of deep learning in bioinformatics



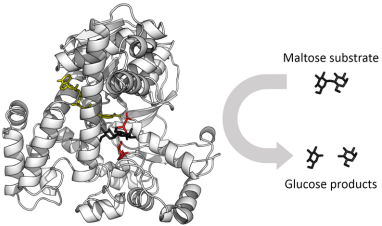
(a) transcription factor binding prediction



(b) protein folding prediction

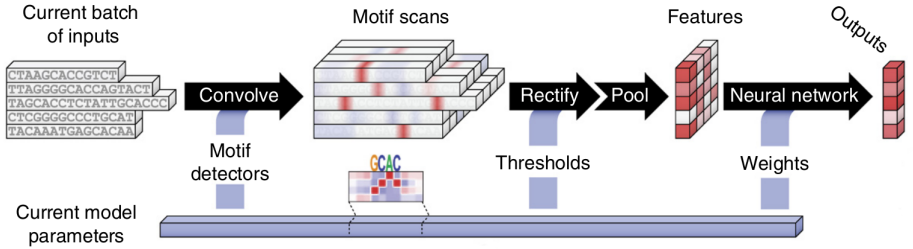


(c) drug discovery



(d) distinguishing enzyme structures

Deep learning for structured data

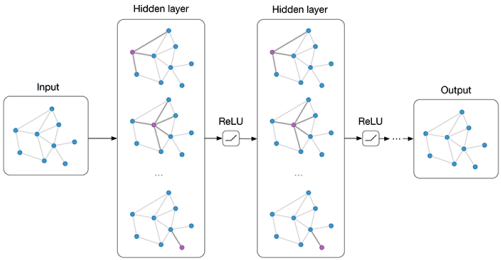


[Alipanahi et al., 2015]

Convolutional neural networks for biological sequences

- borrow ideas from **natural image** modeling, do not work well when labels are scarce;
- outperform classical approaches (e.g. kernel methods) in several tasks;

Deep learning for structured data

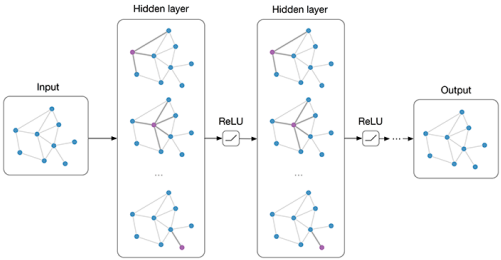


[Kipf and Welling, 2017]

Convolutional neural networks for graphs

- borrow ideas from **natural image** modeling, do not work well when labels are scarce;
- borrow ideas from graph kernels (e.g. Weisfeiler-Lehman graph kernels);

Deep learning for structured data



[Kipf and Welling, 2017]

Convolutional neural networks for graphs

- borrow ideas from **natural image** modeling, do not work well when labels are scarce;
- borrow ideas from graph kernels (e.g. Weisfeiler-Lehman graph kernels);
- how do we describe the **functions** defined by these networks? **Interpretation**?

Supervised learning for structured data modeling

- Goal: learning a **predictive** function $f : \mathcal{X} \to \mathbb{R}$ based on the training examples $(x_i, y_i)_{i=1,\dots,n}$ in $\mathcal{X} \times \mathbb{R}$

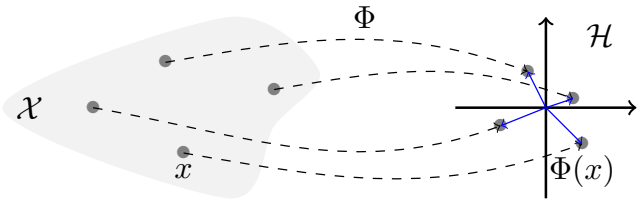
$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\mu \Omega(f)}_{\text{regularization}} .$$

Kernel-based supervised learning for structured data modeling

- Goal: learning a **predictive** function $f : \mathcal{X} \to \mathbb{R}$ based on the training examples $(x_i, y_i)_{i=1,\ldots,n}$ in $\mathcal{X} \times \mathbb{R}$

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

- Map data x in $\mathcal{X}$ to $\Phi(x)$ in $\mathcal{H}$ and work with **linear forms**: $f(x) = \langle f, \Phi(x) \rangle_{\mathcal{H}}$



Kernel-based supervised learning for structured data modeling

- Goal: learning a **predictive** function $f : \mathcal{X} \to \mathbb{R}$ based on the training examples $(x_i, y_i)_{i=1,\ldots,n}$ in $\mathcal{X} \times \mathbb{R}$

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

- $f(x) = \langle f, \Phi(x) \rangle_{\mathcal{H}}$ but $\Phi(x)$ may be **high or infinite-dimensional**.
- Learning only requires manipulating **inner-products** $K(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}$.
- The predictive function can be **regularized** by controlling $\|.\|_{\mathcal{H}}$.

Bridging the gap with deep kernel machines

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?

Bridging the gap with deep kernel machines

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?

Deep kernel machines for sequences and graphs

- Success of deep kernels for image classification [Mairal, 2016];
- A large number of well-studied kernels for sequences and graphs;
- Understand building blocks of deep networks through classical kernels?
- Build deep kernels for sequences and graphs that perform as well as deep networks?

Contributions of the thesis

Biological sequence modeling

- D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks.
  Bioinformatics, 2019a and also in *Research in Computational Molecular Biology (RECOMB)*, 2019c
- D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks.
  In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b

Contributions of the thesis

Biological sequence modeling

Graph modeling

- D. Chen, L. Jacob, and J. Mairal. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning (ICML)*, 2020

Contributions of the thesis

Biological sequence modeling

Graph modeling

Feature aggregation for structured data

- G. Mialon*, D. Chen*, A. d'Aspremont, and J. Mairal. A trainable optimal transport embedding for feature aggregation.
  arXiv preprint arXiv:2006.12065, 2020

Contributions of the thesis

Biological sequence modeling

Graph modeling

Feature aggregation for structured data

Other work on regularization for deep neural networks

- A. Bietti, G. Mialon, D. Chen, and J. Mairal. A kernel perspective for regularizing deep neural networks.
  In *International Conference on Machine Learning (ICML)*, 2019

Contributions of the thesis

Biological sequence modeling

Graph modeling

Feature aggregation for structured data

Other work on regularization for deep neural networks

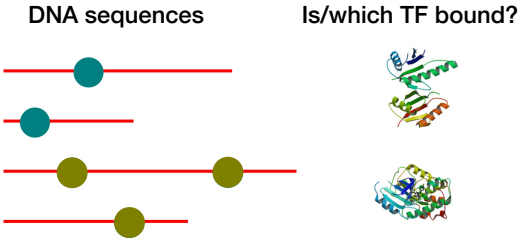
Software: `https://dexiong.me/software/`

Biological Sequence Modeling

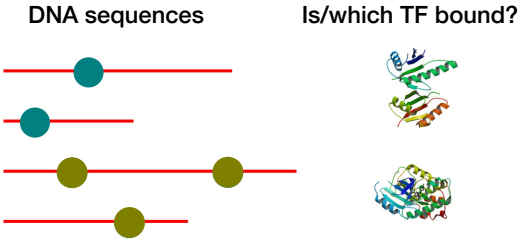
D. Chen, L. Jacob, and J. Mairal. **Biological sequence modeling with convolutional kernel networks.**
***Bioinformatics*, 35(18):3294–3302, 2019a**

D. Chen, L. Jacob, and J. Mairal. **Recurrent kernel networks.**
In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b

Sequence modeling as a supervised learning problem



Sequence modeling as a supervised learning problem



- Biological sequences $\mathbf{x}_1, \dots \mathbf{x}_n \in \mathcal{X}$ and their associated labels $y_1, \dots, y_n$.
- Goal: learning a **predictive** and **interpretable** function $f : \mathcal{X} \to \mathbb{R}$

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

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

String kernels

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

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

- the number of occurrences of u in $\mathbf{x} \Rightarrow$ **spectrum kernel** [Leslie et al., 2002]
- the number of occurrences of u in $\mathbf{x}$ up to m mismatches $\Rightarrow$ **mismatch kernel** [Leslie et al., 2004]
- the number of occurrences of u in $\mathbf{x}$ allowing gaps, with a weight decaying exponentially with the number of gaps $\Rightarrow$ **substring kernel** [Lodhi et al., 2002]

The feature map $\Phi(\mathbf{x})$ can be interpreted as a **histogram of subsequence occurrences**.

Convolutional kernel networks for sequences

We consider a continuous relaxation of the mismatch kernel

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

- We use one-hot encoding to represent k-mers:

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

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

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

Convolutional kernel networks for sequences

We consider a continuous relaxation of the mismatch kernel

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

- We use one-hot encoding to represent k-mers:

$$\mathbf{x}_{[i:i+5]} := \mathrm{TTGAG} \mapsto \begin{matrix} \mathrm{A} \\ \mathrm{T} \\ \mathrm{C} \\ \mathrm{G} \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

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

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

Convolutional kernel networks for sequences

We consider a continuous relaxation of the mismatch kernel

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

- We use one-hot encoding to represent k-mers:

$$\mathbf{x}_{[i:i+5]} := \mathrm{TTGAG} \mapsto \begin{matrix} \mathrm{A} \\ \mathrm{T} \\ \mathrm{C} \\ \mathrm{G} \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

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

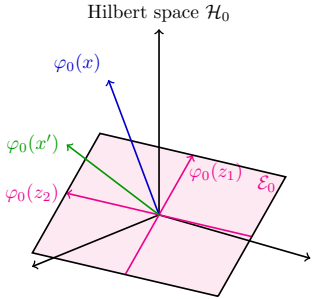
[Chen et al., 2019a, Morrow et al., 2017]

Nyström approximation of kernel mapping

$$K_0(x, x') = \langle \varphi_0(x), \varphi_0(x') \rangle_{\mathcal{H}_0}$$

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

$$\mathcal{E}_0 := \text{span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}$$

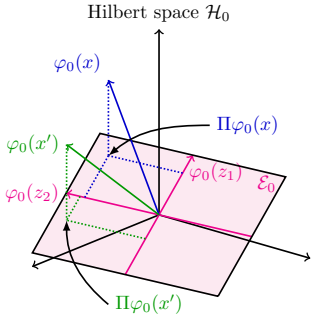


Nyström approximation of kernel mapping

$$K_0(x, x') = \langle \varphi_0(x), \varphi_0(x') \rangle_{\mathcal{H}_0} \approx \langle \Pi\varphi_0(x), \Pi\varphi_0(x') \rangle_{\mathcal{H}_0}$$

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

$$\mathcal{E}_0 := \text{span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}$$

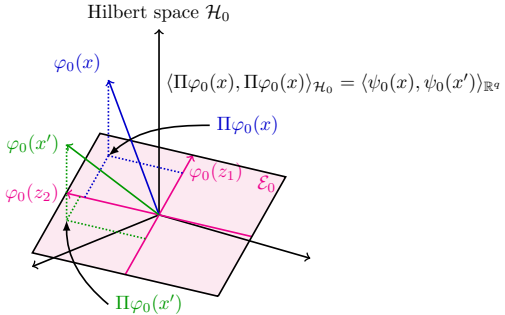


Nyström approximation of kernel mapping

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

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

$$\mathcal{E}_0 := \mathrm{span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}$$



Nyström approximation of kernel mapping

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

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

$$\mathcal{E}_0 := \text{span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}$$

- **General case**:
$\psi_0(x) = [K_0(z_i, z_j)]_{ij}^{-1/2} [K_0(z_1, x), \ldots, K_0(z_q, x)]^\top = K_0(Z, Z)^{-1/2} K_0(Z, x)$

[Williams and Seeger, 2001, Zhang et al., 2008]

Nyström approximation of kernel mapping

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

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

$$\mathcal{E}_0 := \text{span}(\varphi_0(z_1), \ldots, \varphi_0(z_q)) \text{ parametrized by } Z = \{z_1, \ldots, z_q\}$$

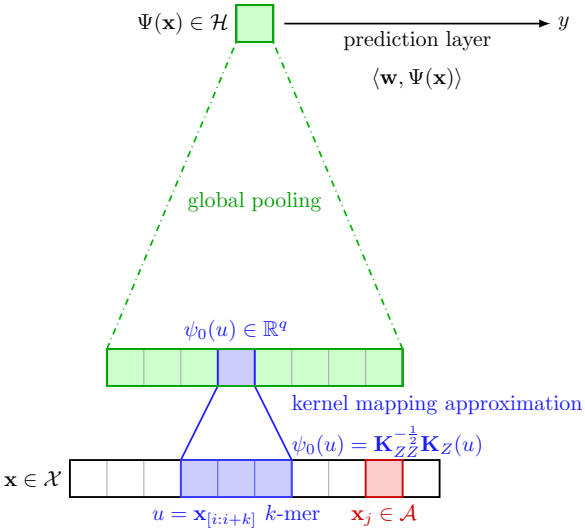
- Case of **dot-product kernels** $K_0(x, x') = \kappa(\langle x, x' \rangle)$:

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

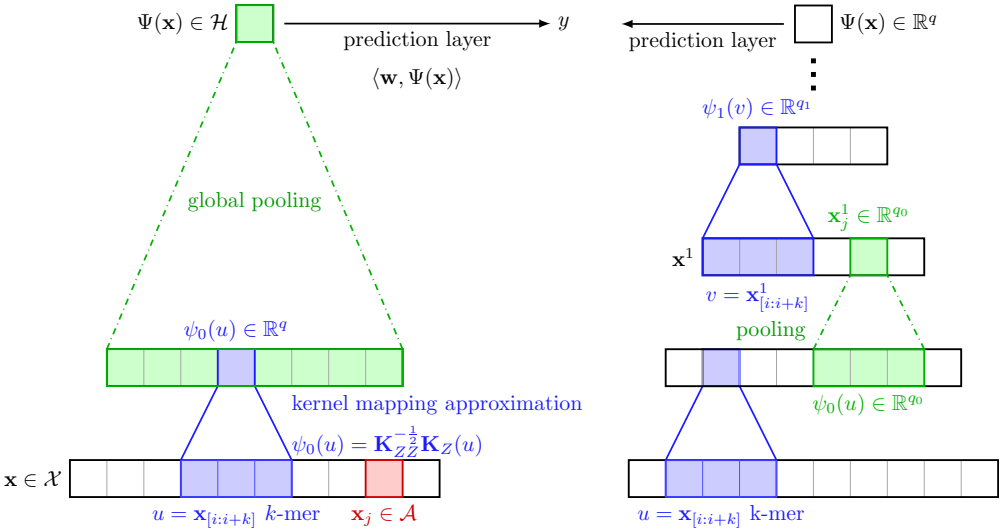
linear operation - pointwise non-linearity - linear operation.
$\kappa(s) = e^{\alpha(s-1)}$ for the Gaussian kernel on unit sphere.

[Williams and Seeger, 2001, Zhang et al., 2008]

Single and multi-layer CKN for sequences



Single and multi-layer CKN for sequences



How do we learn the anchor points Z?

Without supervision:

- 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 as the centroids;
- **compute** the projection matrix $\kappa(Z^\top Z)^{-1/2}$.

How do we learn the anchor points Z?

Without supervision:

- 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 as the centroids;
- **compute** the projection matrix $\kappa(Z^\top Z)^{-1/2}$.

With supervision:

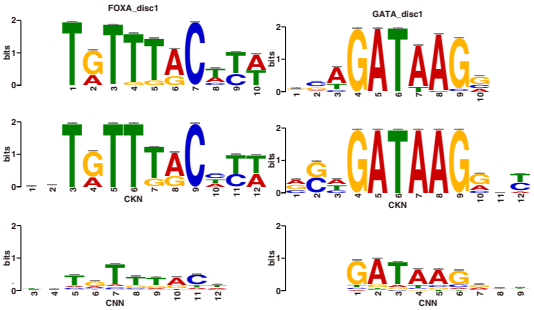
- using **back-propagation** on a supervised loss function with respect to Z;
- differentiating $\kappa(Z^\top Z)^{-1/2}$ requires an eigendecomposition;
- use the above unsupervised procedure as initialization.

Visualization of anchor points for TF binding prediction

We use the representations $\Psi(\mathbf{x})$ obtained with a single-layer CKN

$$\min_{\mathbf{w}\in\mathbb{R}^q,\, Z\in\mathbb{R}^{qkd}} \sum_{i=1}^{n} L(\mathbf{w}^\top \Psi(\mathbf{x}_i), y_i) + \mu\|\mathbf{w}\|^2,$$

where y is a binary label which equals to 1 if $\mathbf{x}$ binds to the TF of interest.

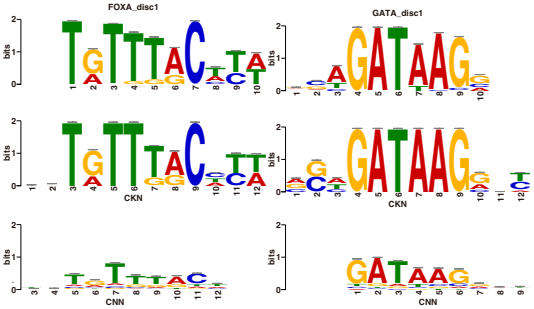


Visualization of anchor points for TF binding prediction

We use the representations $\Psi(\mathbf{x})$ obtained with a single-layer CKN

$$\min_{\mathbf{w}\in\mathbb{R}^q,\, Z\in\mathbb{R}^{qkd}} \sum_{i=1}^{n} L(\mathbf{w}^\top \Psi(\mathbf{x}_i), y_i) + \mu\|\mathbf{w}\|^2,$$

where y is a binary label which equals to 1 if $\mathbf{x}$ binds to the TF of interest.



Limitation: unable to capture **gapped motifs** (e.g. useful to model genetic insertions.)

From k-mers to k-substring

k-mers with gaps

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

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

- The length of the gaps in the substring is

$$\text{gaps}(\mathbf{i}) = \text{number of gaps in the indices.}$$

- Example: $\mathbf{x} = \text{BAARACADACRB}$

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

Recurrent kernel networks

Comparing all the k-mers between a pair of sequences

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

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

[Lodhi et al., 2002, Lei et al., 2017]

Recurrent kernel networks

Comparing all the k-substrings between a pair of sequences

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

- The kernel mapping is $\Phi(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{I}(k, |\mathbf{x}|)} \lambda^{\mathrm{gaps}(\mathbf{i})} \varphi_0(\mathbf{x}_{[\mathbf{i}]})$.
- This is a differentiable relaxation of the substring kernel.
- $\lambda \in [0; 1]$ is a hyperparameter that penalizes the gaps in k-substrings.

[Lodhi et al., 2002, Lei et al., 2017]

Approximation and recursive computation of RKN

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

$$\Psi(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{I}(k, |\mathbf{x}|)} \lambda^{\text{gaps}(\mathbf{i})} \psi_0(\mathbf{x}_{[\mathbf{i}]}),$$

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

- Exhaustive enumeration of all substrings can be **exponentially** costly.

Approximation and recursive computation of RKN

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

$$\Psi(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{I}(k, |\mathbf{x}|)} \lambda^{\text{gaps}(\mathbf{i})} \psi_0(\mathbf{x}_{[\mathbf{i}]}),$$

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

- Exhaustive enumeration of all substrings can be **exponentially** costly.
- The sum can be computed using **dynamic programming** [Lodhi et al., 2002],
- which leads to a **particular** recurrent neural network [Lei et al., 2017].

The feature map of RKN

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

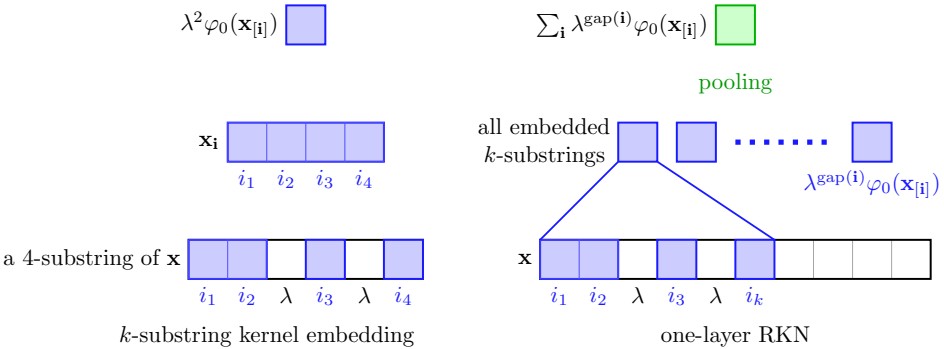
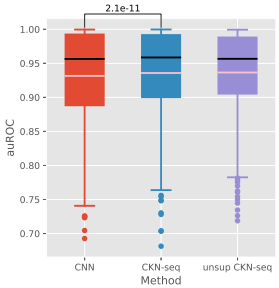
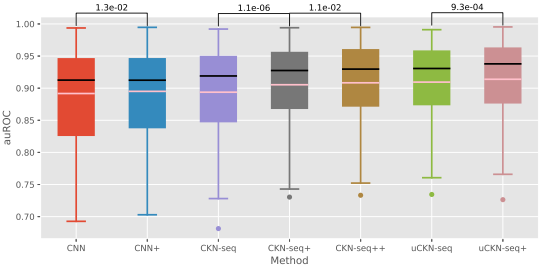


Figure: Example of K_{RKN} for $k = 4$

Transcription factor binding prediction



(a) ENCODE



(b) only 500 subsamples from ENCODE

- Increasing ♯layers does not improve performance for short sequences ($\sim$101bp).
- CKNs outperform CNNs especially when few training examples are available.
- In this case, **non-supervision** and **data augmentation** can improve performance.

Protein fold classification

Protein fold classification on SCOP 2.06 [Hou et al., 2018] (sequence features include one-hot encoding, PSSM, secondary structure and solvent accessibility)
A dataset with few labels: 19,245 sequences from 1,195 different classes of fold.

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

[Hou et al., 2018, Chen et al., 2019a,b]

Protein fold classification

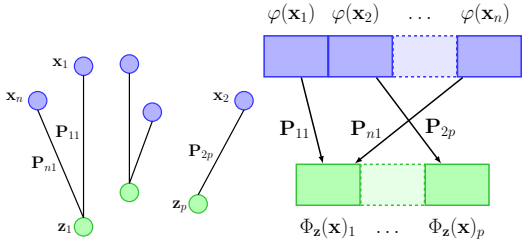
Protein fold classification on SCOP 2.06 [Hou et al., 2018] (sequence features include one-hot encoding, PSSM, secondary structure and solvent accessibility)
A dataset with few labels: 19,245 sequences from 1,195 different classes of fold.

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

Can we do even better?

Replacing the mean pooling with our **optimal transport based adaptive pooling** (OTKE [Mialon*, Chen*, d'Aspremont and Mairal 2020]): 85.29 $\rightarrow$ 91.24

Basic idea: a trainable optimal transport embedding



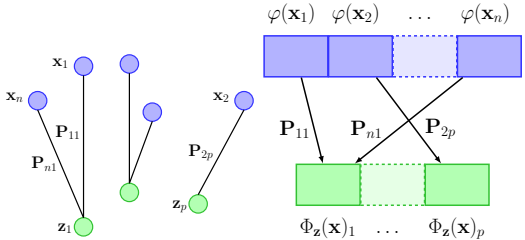
- View sequence as a set of k-mer features $\varphi(\mathbf{x}_i)$ extracted by CKN before pooling.
- Compare a pair of sequences based on an optimal transport between two sets:

$$K(\mathbf{x}, \mathbf{x}') = \min_{\mathbf{P} \in U(\mathbf{x}, \mathbf{x}')} \sum_{ij} -\mathbf{P}_{ij} \kappa(\mathbf{x}_i, \mathbf{x}'_j) - \varepsilon H(\mathbf{P}), \tag{1}$$

$$U(\mathbf{x}, \mathbf{x}') = \{\mathbf{P} \in \mathbb{R}_+^{n \times n'} : \mathbf{P}\mathbf{1}_n = 1/n \text{ and } \mathbf{P}^\top \mathbf{1}_{n'} = 1/n'\}.$$

G. Mialon*, D. Chen* et al. A trainable optimal transport embedding for feature aggregation. arXiv 2020

Basic idea: a trainable optimal transport embedding



- Let $\mathbf{P}(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{n \times p}$ be the solution of the OT problem 1 between $\mathbf{z}$ and $\mathbf{x}$, and

$$\Phi_{\mathbf{z}}(\mathbf{x}) := \sqrt{p} \times \left(\sum_{i=1}^{n} \mathbf{P}(\mathbf{x}, \mathbf{z})_{i1} \varphi(\mathbf{x}_i), \ldots, \sum_{i=1}^{n} \mathbf{P}(\mathbf{x}, \mathbf{z})_{ip} \varphi(\mathbf{x}_i) \right) = \sqrt{p} \times \mathbf{P}(\mathbf{x}, \mathbf{z})^\top \varphi(\mathbf{x})$$

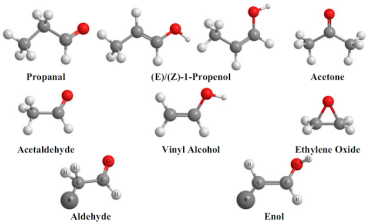
- A valid kernel can be defined as $K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{p} \langle \Phi_{\mathbf{z}}(\mathbf{x})_i, \Phi_{\mathbf{z}}(\mathbf{x}')_i \rangle$.
- Parameter $\mathbf{z}$ can be learned in both unsupervised and supervised ways.

G. Mialon*, D. Chen* et al. A trainable optimal transport embedding for feature aggregation. arXiv 2020

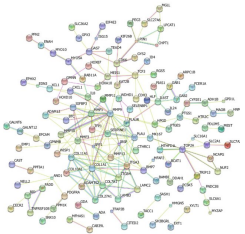
Graph Modeling

D. Chen, L. Jacob, and J. Mairal. **Convolutional kernel networks for graph-structured data.** In *International Conference on Machine Learning (ICML)*, 2020

Graph-structured data are ubiquitous

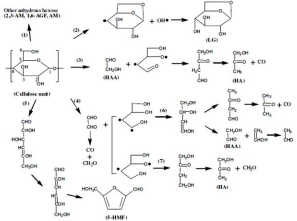


(c) molecules



(d) protein regulation

(e) social networks



(f) 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

Learning graph representations

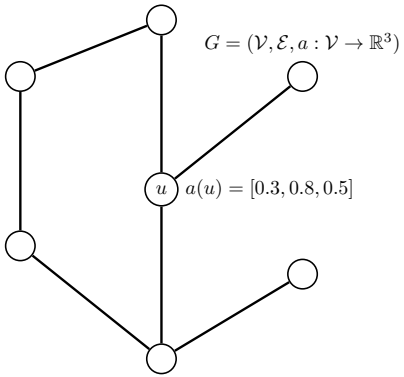
State-of-the-art models for representing graphs

- **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

Our model:

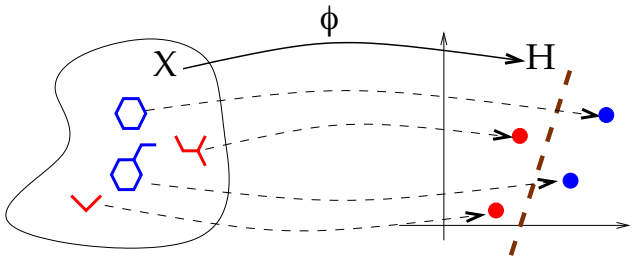
- A new type of **multilayer** graph kernel: more **expressive** than WL kernels
- Learning easy-to-regularize and scalable **unsupervised** graph representations
- Learning **supervised** graph representations like GNNs

Graphs with node attributes



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

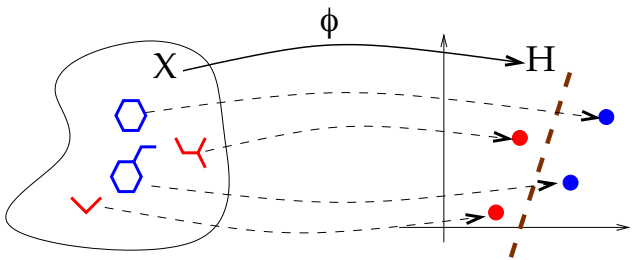
Graph kernel mappings



- Map each graph G in $\mathcal{X}$ to a vector $\Phi(G)$ in $\mathcal{H}$, which lends itself to learning tasks.

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

Graph kernel mappings

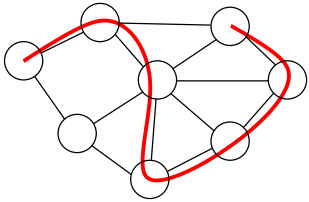


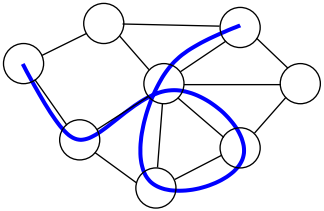
- Map each graph G in $\mathcal{X}$ to a vector $\Phi(G)$ in $\mathcal{H}$, which lends itself to learning tasks.
- A large class of graph kernel mappings can be written in the form

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

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

Basic kernels: walk and path kernel mappings



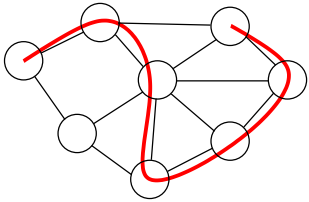


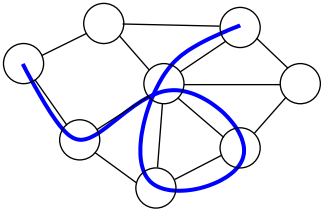
- $\mathcal{P}_k(G, u) :=$ paths of length k from node u in G. The **k-path** mapping is

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

- $a(p)$: concatenated attributes in p; δ: the Dirac function.
- $\Phi_{\text{path}}(G)$ can be interpreted as a **histogram** of paths occurrences.

Basic kernels: walk and path kernel mappings



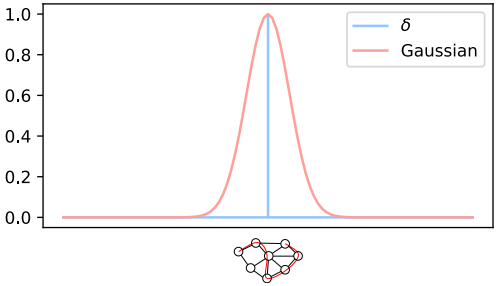


- $\mathcal{P}_k(G, u) :=$ paths of length k from node u in G. The **k-path** mapping is

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

- $a(p)$: concatenated attributes in p; δ: the Dirac function.
- $\Phi_{\text{path}}(G)$ can be interpreted as a **histogram** of paths occurrences.
- Path kernels are more **expressive** than walk kernels, but less preferred for **computational** reasons.

A relaxed path kernel

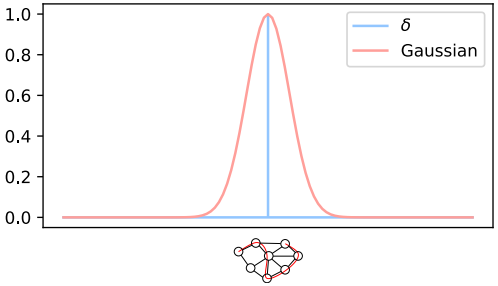


$$\varphi_{\text{path}}(u) = \sum_{p \in \mathcal{P}_k(G,u)} \delta_{a(p)}(\cdot)$$

Issues of the path kernel mapping:

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

A relaxed path kernel



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

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

Issues of the path kernel mapping:

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

Relax it with a "soft" and differentiable mapping

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

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

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

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

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

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

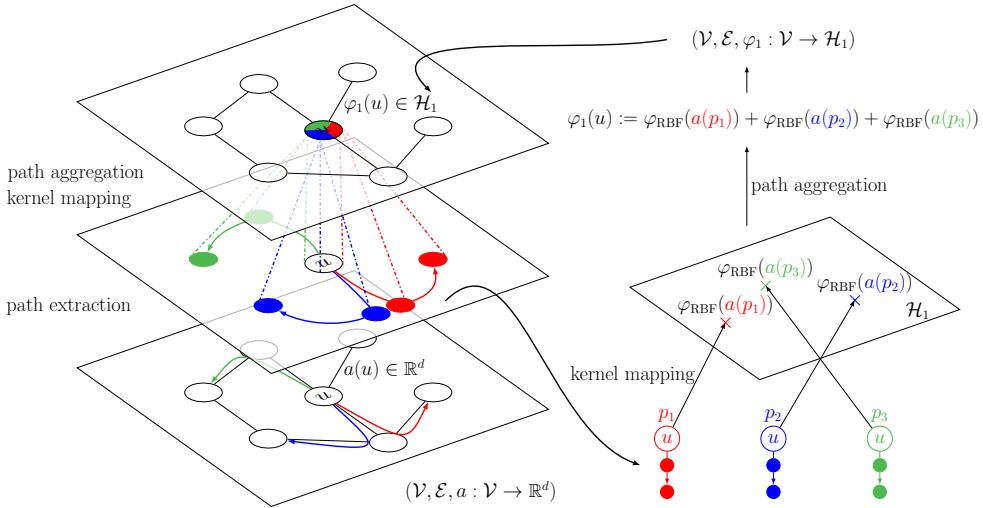
- We define the one-layer GCKN as the relaxed path kernel mapping

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

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

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

Construction of one-layer GCKN



From one-layer to multilayer GCKN

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

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

- $\varphi_j(u)$ represents the information about a neighborhood of u.
- Final graph representation at layer j, $\Phi_j(G) = \sum_{u \in \mathcal{V}} \varphi_j(u)$.

From one-layer to multilayer GCKN

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

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

- $\varphi_j(u)$ represents the information about a neighborhood of u.
- Final graph representation at layer j, $\Phi_j(G) = \sum_{u \in \mathcal{V}} \varphi_j(u)$.
- Why is the multilayer model interesting ?
  - applying φ_{path} once can capture **paths**: GCKN-path;
  - applying twice can capture **subtrees**: GCKN-subtree;
  - so applying even 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

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

- $\varphi_{\text{RBF}}(x) = e^{-\frac{\alpha}{2}\|x-\cdot\|^2} \in \mathcal{H}$ is infinite-dimensional (expensive to compute).

[Chen et al., 2019a,b]

Scalable approximation of Gaussian kernel mapping

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

- $\varphi_{\text{RBF}}(x) = e^{-\frac{\alpha}{2}\|x-\cdot\|^2} \in \mathcal{H}$ is infinite-dimensional (expensive to compute).
- **Nyström** provides a **finite-dimensional** approximation $\psi(x) \in \mathbb{R}^q$ by orthogonally projecting $\varphi_{\text{RBF}}(x)$ 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]

Scalable approximation of Gaussian kernel mapping

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

- $\varphi_{\text{RBF}}(x) = e^{-\frac{\alpha}{2}\|x-\cdot\|^2} \in \mathcal{H}$ is infinite-dimensional (expensive to compute).
- **Nyström** provides a **finite-dimensional** approximation $\psi(x) \in \mathbb{R}^q$ by orthogonally projecting $\varphi_{\text{RBF}}(x)$ onto some finite-dimensional subspace:

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

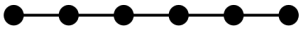
  where $z_j \in \mathbb{R}^{dk}$ can be interpreted as path features.
- The parameters Z can be learned by
  - (unsupervised) K-means on the set of path features;
  - (supervised) end-to-end learning with back-propagation.

[Chen et al., 2019a,b]

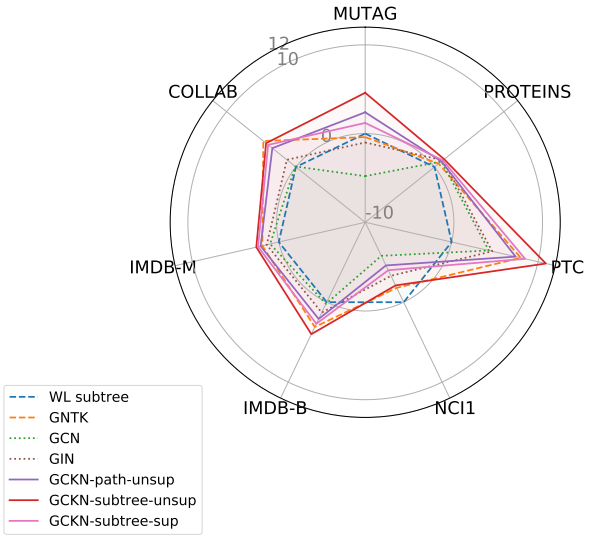
Comparison of GCKN and GNN

GCKN	vs.	GNN
$\Psi_{\text{GCKN}}(G) = \sum_{u \in G} \psi_j(u)$		$\Psi_{\text{GNN}}(G) = \sum_{u \in G} f_j(u)$
$\psi_j(u) = \sum_{p \in \mathcal{P}_k(G,u)} \kappa(Z^\top Z)^{-\frac{1}{2}} \kappa(Z^\top \psi_{j-1}(p))$		$f_j(u) = \sum_{v \in \mathcal{N}(u)} \text{ReLU}(Z^\top f_{j-1}(v))$
local path aggregation		neighborhood aggregation
projection in a known RKHS		unknown functional space
both supervised and unsupervised		only supervised

If G is a (directed) path graph, GCKN becomes a CKN while GNN will not recover a CNN for $k > 1$.



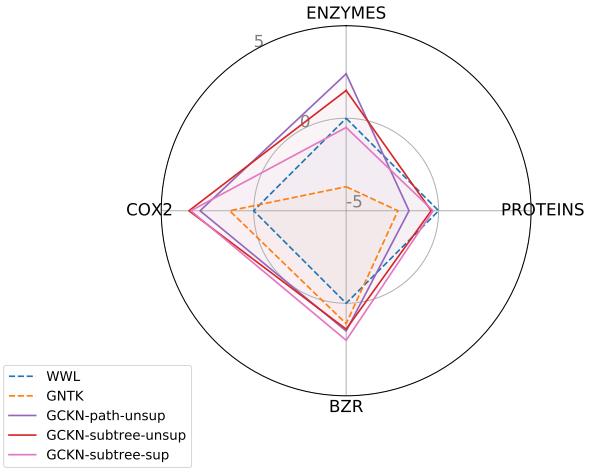
Experiments on graphs with discrete attributes



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

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

Experiments on graphs with continuous attributes

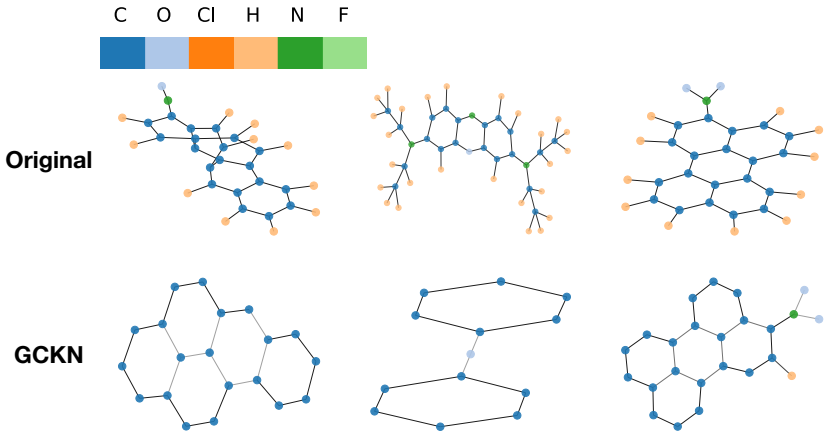


- Accuracy improvement with respect to the WWL kernel.
- Results similar to discrete case.
- Path features seem presumably predictive enough.

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

Model interpretation for mutagenicity prediction

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



[Ying et al., 2019]

Conclusion and Future Research

Conclusion

Convolutional and recurrent kernel networks for biological sequences

- Multilayer kernels for biological sequences.
- Achieve SOTA in TF binding prediction and protein fold classification.
- RKN is able to model gaps with a RNN structure, useful for remote homology detection.
- Best results were obtained with one-layer models for short sequences.
- Non-supervision and data augmentation can improve performance when labels are scarce.

Conclusion

Convolutional kernel networks for graphs

- A multilayer kernel for graphs based on paths.
- Allows to control the trade-off between computation and expressiveness.
- A straightforward model interpretation is provided.
- Long path features could be useful for toxicology prediction.
- Ongoing collaboration on protein model quality assessment.

Conclusion

Supervised vs. unsupervised representations

- Without supervision, models provide effective but high-dimensional embeddings.
- With supervision, models trained with backpropagation are much more compact.

Feature aggregation

- Max pooling generally outperforms mean pooling in practice but less stable.
- Max pooling can be simulated in RKHSs.
- An optimal transport based adaptive pooling performs even better.

Future research and perspectives

Efficient learning pipelines to deal with genome-scale data

- Training CKNs or CNNs directly on genome-scale data can be costly and inefficient.
- Localize large relevant regions with selection methods?
- Then perform refined learning on selected regions.

Future research and perspectives

More compact and accurate unsupervised representations

- Nyström approximation is **not efficient** for higher layers.
- Better approximation methods for deep kernels [Shankar et al., 2020]?
- Self-supervised learning to learn more compact representations [Caron et al., 2018, Rives et al., 2019]?

Future research and perspectives

Performance gap between kernel methods and ResNets

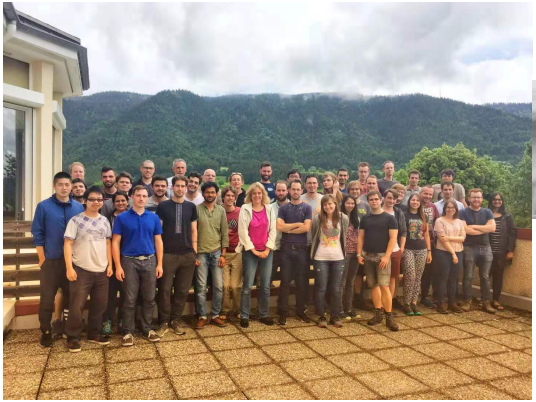
- ResNets perform "hierarchical learning" while kernels cannot [Allen-Zhu and Li, 2019].
- Deep kernels can perform as well as convolutional networks but worse than ResNets on CIFAR-10 [Shankar et al., 2020].
- Multiple kernel learning can select kernels defined on different layers.

Future research and perspectives

Better feature aggregation for structured data

- Optimal transport for better feature aggregation, theoretical guarantee?
- Other inductive bias from kernel literature (e.g. Fisher kernels)?

Thank you!



References I

B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.

Z. Allen-Zhu and Y. Li. What can resnet learn efficiently, going beyond kernels? In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9017–9028, 2019.

A. Bietti, G. Mialon, D. Chen, and J. Mairal. A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, 35(18):3294–3302, 2019a.

D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b.

D. Chen, L. Jacob, and J. Mairal. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning (ICML)*, 2020.

S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (Neurips)*, 2019.

References II

S. Hochreiter, M. Heusel, and K. Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736, 2007.

J. Hou, B. Adhikari, and J. Cheng. Deepsf: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics*, 34(8):1295–1303, 2018.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

N. M. Kriege, M. Neumann, C. Morris, K. Kersting, and P. Mutzel. A unifying view of explicit and implicit feature maps of graph kernels. *Data Mining and Knowledge Discovery*, 33(6):1505–1547, 2019.

T. Lei, W. Jin, R. Barzilay, and T. Jaakkola. Deriving neural architectures from sequence and graph kernels. In *International Conference on Machine Learning (ICML)*, 2017.

C. Leslie, E. Eskin, J. Weston, and W. Noble. Mismatch String Kernels for SVM Protein Classification. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2003.

C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, volume 7, pages 566–575. Hawaii, USA, 2002.

C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships. *Journal of computational biology*, 10(6):857–868, 2003.

References III

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research (JMLR)*, 2(Feb):419–444, 2002.

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

G. Mialon*, D. Chen*, A. d'Aspremont, and J. Mairal. A trainable optimal transport embedding for feature aggregation. *arXiv preprint arXiv:2006.12065*, 2020.

A. Morrow, V. Shankar, D. Petersohn, A. Joseph, B. Recht, and N. Yosef. Convolutional kitchen sinks for transcription factor binding site prediction. *arXiv preprint arXiv:1706.00125*, 2017.

A. Rives, S. Goyal, J. Meier, D. Guo, M. Ott, C. L. Zitnick, J. Ma, and R. Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, 2019.

V. Shankar, A. Fang, W. Guo, S. Fridovich-Keil, L. Schmidt, J. Ragan-Kelley, and B. Recht. Neural kernels without tangents. In *International Conference on Machine Learning (ICML)*, 2020.

N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12(9), 2011.

M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In *Advances in Neural Information Processing Systems (Neurips)*, 2019.

J.-P. Vert, H. Saigo, and T. Akutsu. Convolution and local alignment kernels. *Kernel methods in computational biology*, pages 131–154, 2004.

References IV

C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems (NIPS)*, 2001.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In *Advances in neural information processing systems (Neurips)*, 2019.

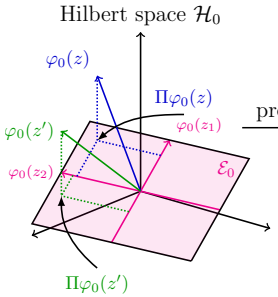
K. Zhang, I. W. Tsang, and J. T. Kwok. Improved nyström low-rank approximation and error analysis. In *International Conference on Machine Learning (ICML)*, 2008.

Visualization of anchor points in CKN

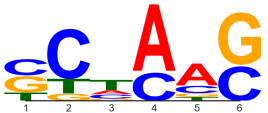
- For a one-layer, find the preimage of filter i by optimizing

$$\min_{u \in \mathcal{M}} \|\varphi_0(u) - \varphi_0(z_i)\|_{\mathcal{H}_0}^2,$$

  where $\mathcal{M} \subseteq \mathbb{R}^{k \times 4}$ is an appropriate simplex of motifs.
- Projection onto the simplex induces sparsity thus more informative motif.



preimage



A	0	0	0.2	0.7	0.6	0
C	0.4	0.7	0.2	0.3	0.2	0.4
G	0.4	0.1	0.1	0	0.1	0.6
T	0.2	0.2	0.5	0	0.1	0

motif associated with $\varphi_0(z_1)$

Computation of recurrent kernel networks

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

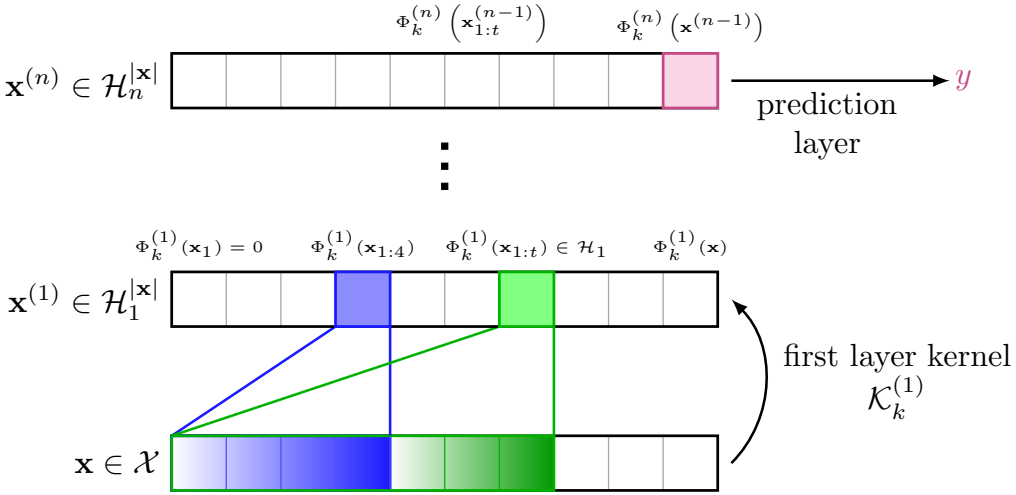
$$\psi_j(\mathbf{x}_{1:t}) = \sum_{\mathbf{i}\in\mathcal{I}(j,t)} \lambda^{\text{gaps}(\mathbf{i})}\psi_0(\mathbf{x}_{1:t}[\mathbf{i}]) = K_{Z_jZ_j}^{-1/2} \sum_{\mathbf{i}\in\mathcal{I}(j,t)} \lambda^{\text{gaps}(\mathbf{i})} K_{Z_j}(\mathbf{x}[\mathbf{i}]) := K_{Z_jZ_j}^{-1/2}\mathbf{h}_j[t],$$

for any $j \in \{1,\dots,k\}$ and $t \in \{1,\dots,|\mathbf{x}|\}$. Z_j is a matrix in $\mathbb{R}^{d\times q}$ whose i-th column is the j-th vector of z_i.
We can prove that $\mathbf{h}_j[t]$ in $\mathbb{R}^q$ obeying some recursion similar to the one used in substring kernel

$$\begin{aligned}
\mathbf{c}_j[1] &= \mathbf{h}_j[1] = 0 && 1 \le j \le k,\\
\mathbf{c}_0[t] &= 1 && 1 \le t \le |\mathbf{x}|,\\
\mathbf{c}_j[t] &= \lambda\mathbf{c}_j[t-1] + \mathbf{c}_{j-1}[t-1] \odot \kappa(Z_j^\top \mathbf{x}_t) && 1 \le j \le k,\\
\mathbf{h}_j[t] &= \mathbf{h}_j[t-1] + \mathbf{c}_{j-1}[t-1] \odot \kappa(Z_j^\top \mathbf{x}_t) && 1 \le j \le k,
\end{aligned}$$

where κ is a non-linear function $\kappa(x) = e^{\alpha(x-1)}$.

Multilayer construction of RKNs



Results on SCOP 1.67

Protein fold recognition on SCOP 1.67 (widely used benchmark)

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

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

Walks vs paths

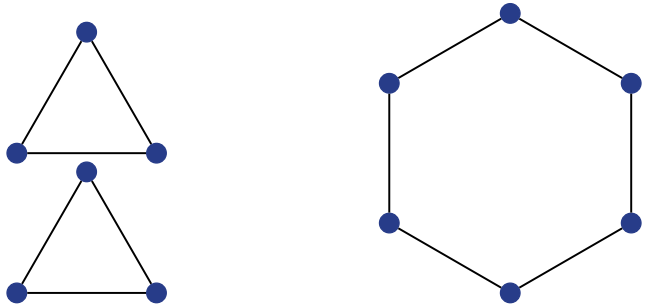
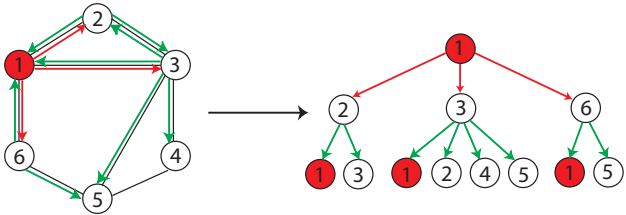


Figure: An example about connectivity where $\varphi_{\text{walk}}(G) = \varphi_{\text{walk}}(G')$ but $\varphi_{\text{path}}(G) \neq \varphi_{\text{path}}(G')$

- **Tottering** walks seem **irrelevant** for many applications.
- Path kernels are generally **more expressive** than walk kernels.
- Most existing methods rely on walks for **computational reason**.

Weisfeiler-Lehman subtree kernel



- Enumerating **subtree patterns** can be exponentially costly. Is there a fast way ?
- WL algorithm: **iterative enumeration** for graphs with discrete node labels.
  - We define **a sequence of node labels** initialized with $a_0 = a$.
  - At iteration $i \geq 1$, $a_i(u) = \text{hash}([a_{i-1}(u), \text{sort}(\{a_{i-1}(v) \mid v \in \mathcal{N}(u)\})])$.
- WL subtree kernel at depth k is defined as

$$\kappa_{\text{subtree}}(u, u') = \delta(a_i(u), a'_i(u'))$$

[Shervashidze et al., 2011]

Motivation: link between walk and WL subtree kernels

Is there some relation between the base kernels κ_{walk} and κ_{subtree} ?

WL subtree kernel as a 2-layer walk kernel

Let $\mathcal{M}(u, u')$ be the set of exact matchings of subsets of the neighborhoods of two nodes u and u'. For any $u \in G$ and $u' \in G'$ such that $|\mathcal{M}(u, u')| = 1$,

$$\kappa_{\text{subtree}}(u, u') = \delta(\varphi_{\text{walk}}(u), \varphi'_{\text{walk}}(u')), \tag{2}$$

where φ_{walk} is the feature map of κ_{walk} satisfying $\varphi_{\text{walk}}(u) = \sum_{p \in \mathcal{W}_k(G,u)} \varphi_\delta(p)$.

- A sufficient condition for $|\mathcal{M}(u, u')| = 1$: u and u' have **same degrees** and both of them have **distinct neighbors**.
- If we replace φ_{path} instead of φ_{walk} we capture **subtrees** without repeated nodes !

Motivation: link between walk and WL subtree kernels

Is there some relation between the base kernels κ_{walk} and κ_{subtree} ?

WL subtree kernel as a 2-layer walk kernel

Let $\mathcal{M}(u, u')$ be the set of exact matchings of subsets of the neighborhoods of two nodes u and u'. For any $u \in G$ and $u' \in G'$ such that $|\mathcal{M}(u, u')| = 1$,

$$\kappa_{\text{subtree}}(u, u') = \delta(\varphi_{\text{walk}}(u), \varphi'_{\text{walk}}(u')), \tag{2}$$

where φ_{walk} is the feature map of κ_{walk} satisfying $\varphi_{\text{walk}}(u) = \sum_{p \in \mathcal{W}_k(G,u)} \varphi_\delta(p)$.

Can we go beyond subtrees to higher order patterns ?

Motivation: link between walk and WL subtree kernels

Is there some relation between the base kernels κ_{walk} and κ_{subtree} ?

WL subtree kernel as a 2-layer walk kernel

Let $\mathcal{M}(u, u')$ be the set of exact matchings of subsets of the neighborhoods of two nodes u and u'. For any $u \in G$ and $u' \in G'$ such that $|\mathcal{M}(u, u')| = 1$,

$$\kappa_{\text{subtree}}(u, u') = \delta(\varphi_{\text{walk}}(u), \varphi'_{\text{walk}}(u')), \tag{2}$$

where φ_{walk} is the feature map of κ_{walk} satisfying $\varphi_{\text{walk}}(u) = \sum_{p \in \mathcal{W}_k(G,u)} \varphi_\delta(p)$.

Can we go beyond subtrees to higher order patterns ?
Composing path kernels !

Model interpretation of GCKN

- By construction, $\Psi_{\text{GCKN}}(G)$ only depends on G through its set of paths $\mathcal{P}_k(G)$

$$\min_{\mathcal{P}' \subseteq \mathcal{P}_k(G)} L(\hat{y}, \langle \Psi_{\text{GCKN}}(\mathcal{P}'), w \rangle) + \mu |\mathcal{P}'|, \tag{3}$$

- This problem can be relaxed by introducing a mask M with values in $[0; 1]$

$$\min_{M \in [0;1]^{|\mathcal{P}_k(G)|}} L(\hat{y}, \langle \Psi_1(\mathcal{P}_k(G) \odot M), w \rangle) + \mu \|M\|_1, \tag{4}$$

Results for GCKN on graphs with discrete node attributes

Dataset	MUTAG	PROTEINS	PTC	NCI1	IMDB-B	IMDB-M	COLLAB
size	188	1113	344	4110	1000	1500	5000
classes	2	2	2	2	2	3	3
avg ♯nodes	18	39	26	30	20	13	74
avg ♯edges	20	73	51	32	97	66	2458
LDP	88.9 ± 9.6	73.3 ± 5.7	63.8 ± 6.6	72.0 ± 2.0	68.5 ± 4.0	42.9 ± 3.7	76.1 ± 1.4
WL subtree	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	$\mathbf{86.0 \pm 1.8}$	73.8 ± 3.9	50.9 ± 3.8	78.9 ± 1.9
AWL	87.9 ± 9.8	-	-	-	74.5 ± 5.9	51.5 ± 3.6	73.9 ± 1.9
RetGK	90.3 ± 1.1	75.8 ± 0.6	62.5 ± 1.6	84.5 ± 0.2	71.9 ± 1.0	47.7 ± 0.3	81.0 ± 0.3
GNTK	90.0 ± 8.5	75.6 ± 4.2	67.9 ± 6.9	84.2 ± 1.5	76.9 ± 3.6	52.8 ± 4.6	$\mathbf{83.6 \pm 1.0}$
GCN	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0	74.0 ± 3.4	51.9 ± 3.8	79.0 ± 1.8
PatchySAN	92.6 ± 4.2	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9	71.0 ± 2.2	45.2 ± 2.8	72.6 ± 2.2
GIN	89.4 ± 5.6	76.2 ± 2.8	64.6 ± 7.0	82.7 ± 1.7	75.1 ± 5.1	52.3 ± 2.8	80.2 ± 1.9
GCKN-walk-unsup	92.8 ± 6.1	75.7 ± 4.0	65.9 ± 2.0	80.1 ± 1.8	75.9 ± 3.7	53.4 ± 4.7	81.7 ± 1.4
GCKN-path-unsup	92.8 ± 6.1	76.0 ± 3.4	67.3 ± 5.0	81.4 ± 1.6	75.9 ± 3.7	53.0 ± 3.1	82.3 ± 1.1
GCKN-subtree-unsup	95.0 ± 5.2	$\mathbf{76.4 \pm 3.9}$	$\mathbf{70.8 \pm 4.6}$	83.9 ± 1.6	$\mathbf{77.8 \pm 2.6}$	$\mathbf{53.5 \pm 4.1}$	83.2 ± 1.1
GCKN-3layer-unsup	$\mathbf{97.2 \pm 2.8}$	75.9 ± 3.2	69.4 ± 3.5	83.9 ± 1.2	77.2 ± 3.8	53.4 ± 3.6	83.4 ± 1.5
GCKN-subtree-sup	91.6 ± 6.7	76.2 ± 2.5	68.4 ± 7.4	82.0 ± 1.2	76.5 ± 5.7	53.3 ± 3.9	82.9 ± 1.6

Results for GCKN on graphs with continuous node attributes

Dataset	ENZYMES	PROTEINS	BZR	COX2
size	600	1113	405	467
classes	6	2	2	2
attr. dim.	18	29	3	3
avg ♯nodes	32.6	39.0	35.8	41.2
avg ♯edges	62.1	72.8	38.3	43.5
RBF-WL	68.4 ± 1.5	75.4 ± 0.3	81.0 ± 1.7	75.5 ± 1.5
HGK-WL	63.0 ± 0.7	75.9 ± 0.2	78.6 ± 0.6	78.1 ± 0.5
HGK-SP	66.4 ± 0.4	75.8 ± 0.2	76.4 ± 0.7	72.6 ± 1.2
WWL	73.3 ± 0.9	$\mathbf{77.9 \pm 0.8}$	84.4 ± 2.0	78.3 ± 0.5
GNTK	69.6 ± 0.9	75.7 ± 0.2	85.5 ± 0.8	79.6 ± 0.4
GCKN-walk-unsup	73.5 ± 0.5	76.5 ± 0.3	85.3 ± 0.5	80.6 ± 1.2
GCKN-path-unsup	$\mathbf{75.7 \pm 1.1}$	76.3 ± 0.5	85.9 ± 0.5	81.2 ± 0.8
GCKN-subtree-unsup	74.8 ± 0.7	77.5 ± 0.3	85.8 ± 0.9	81.8 ± 0.8
GCKN-3layer-unsup	74.6 ± 0.8	77.5 ± 0.4	84.7 ± 1.0	$\mathbf{82.0 \pm 0.6}$
GCKN-subtree-sup	72.8 ± 1.0	77.6 ± 0.4	$\mathbf{86.4 \pm 0.5}$	81.7 ± 0.7