Divide and Conquer Networks

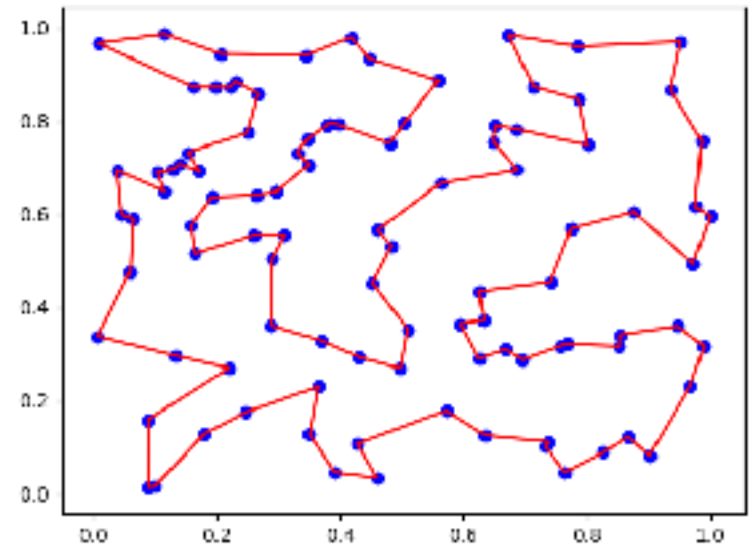
Joan Bruna and Alex Nowak

Courant Institute of Mathematical Sciences
Center for Data Science
NYU

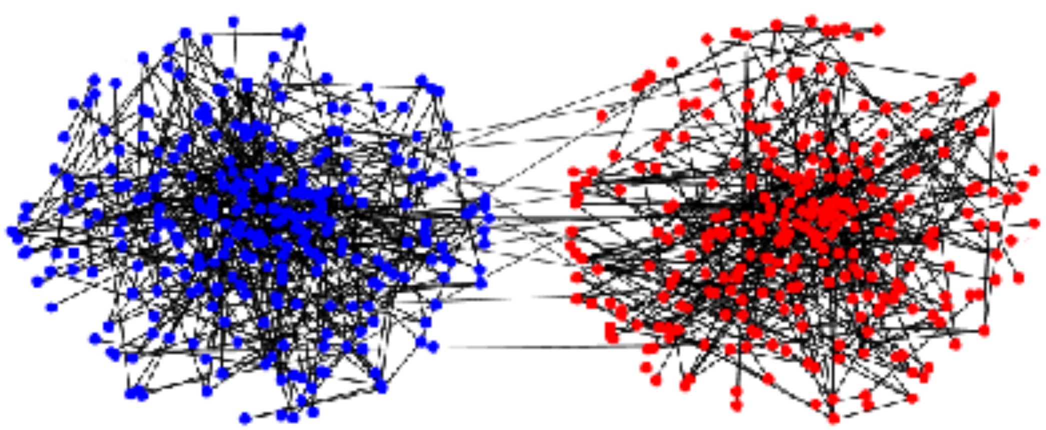
Motivation

- We consider discrete geometric/algorithmic tasks:

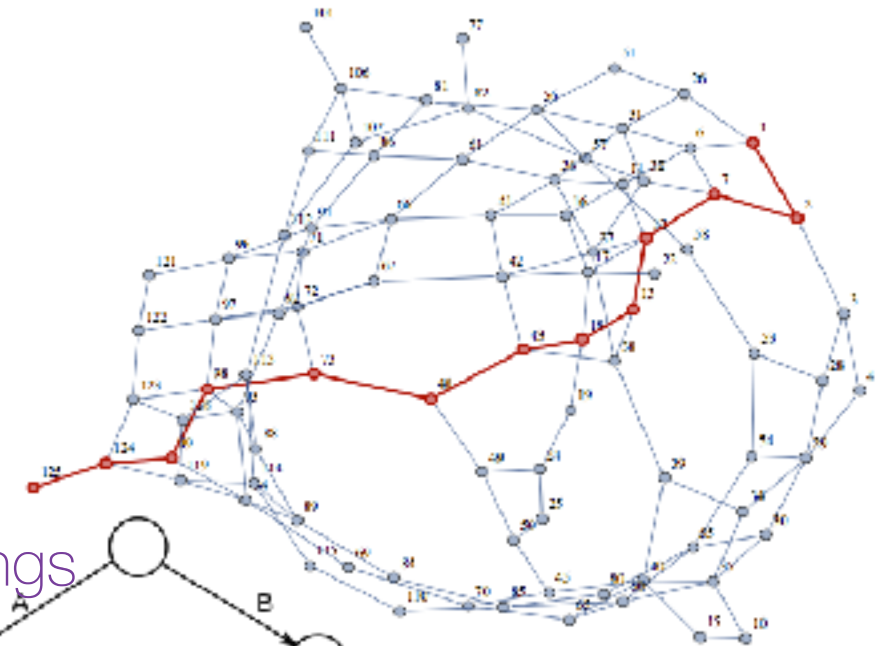
Travelling Salesman



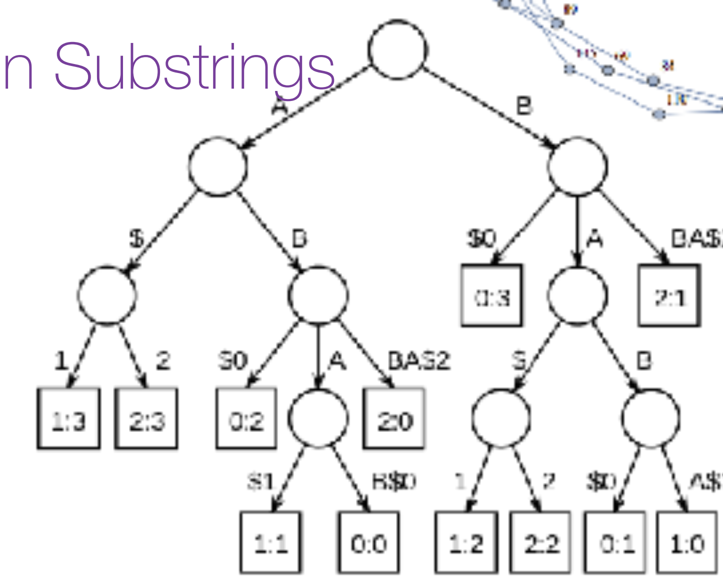
Community Detection



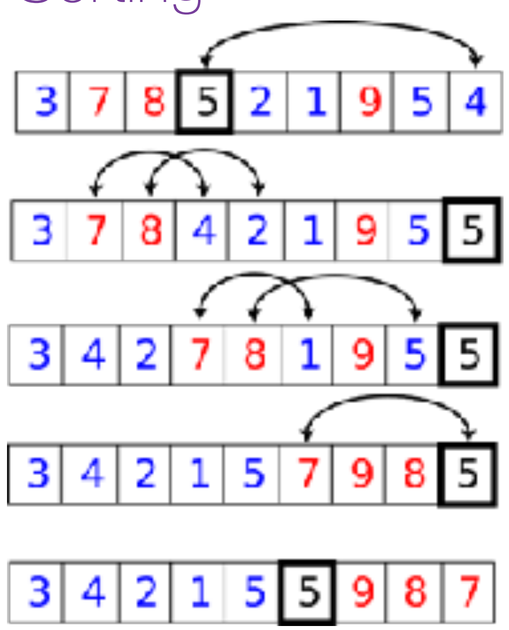
Shortest Paths



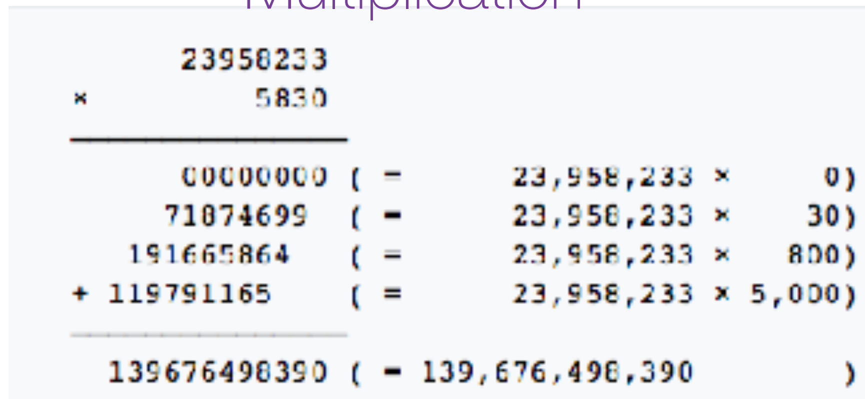
Common Substrings



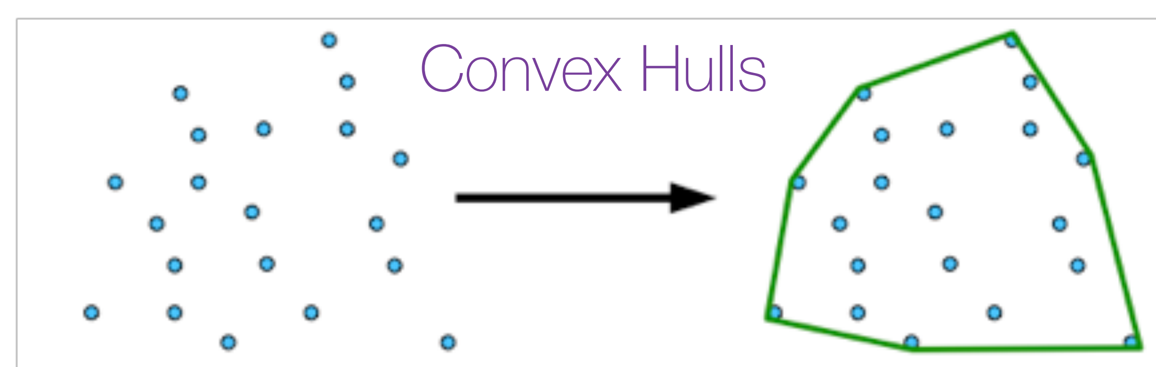
Sorting



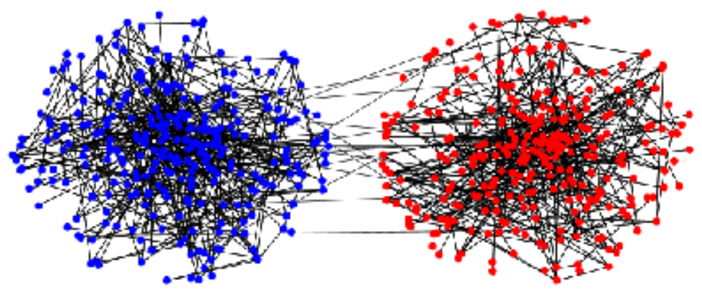
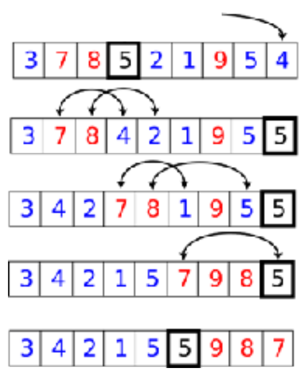
Multiplication

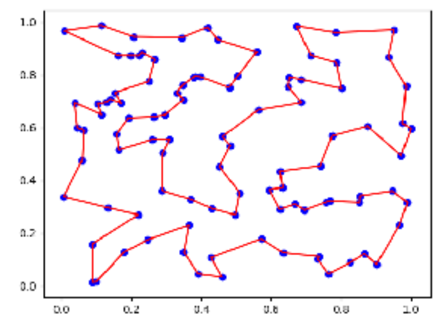
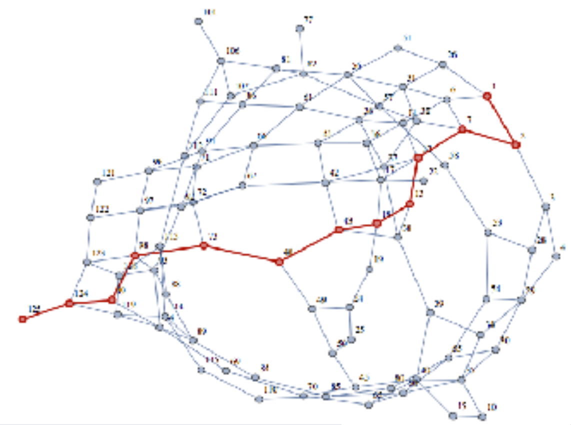
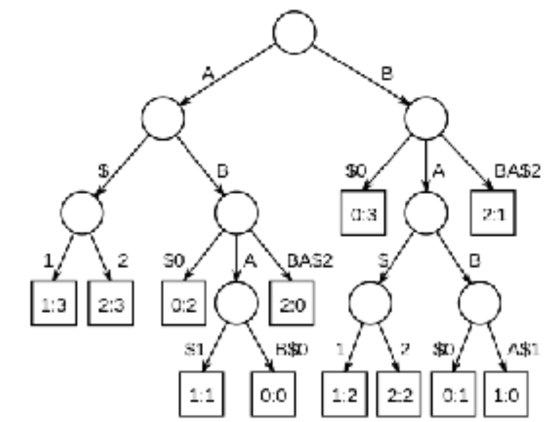
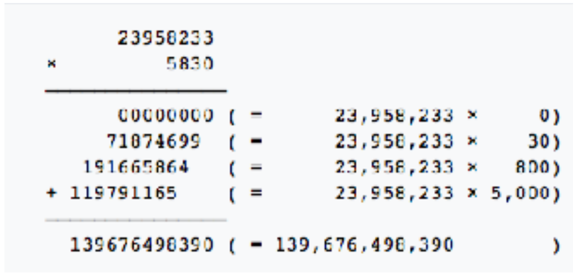


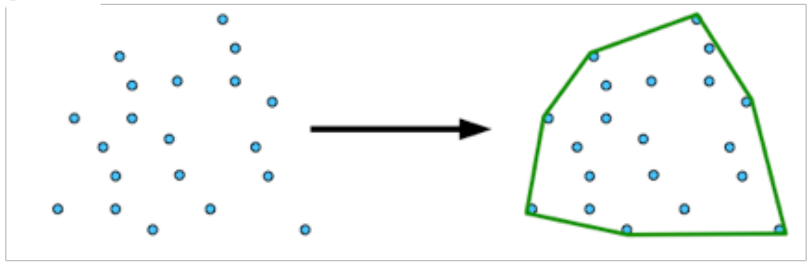
Convex Hulls



Motivation

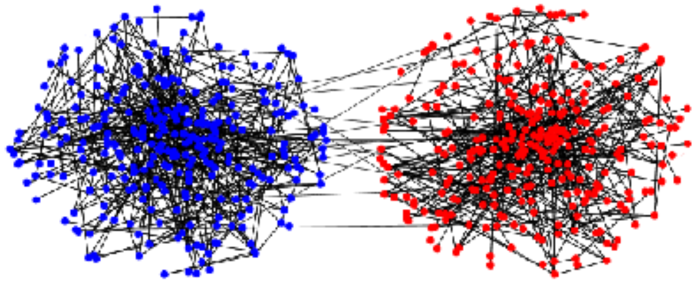



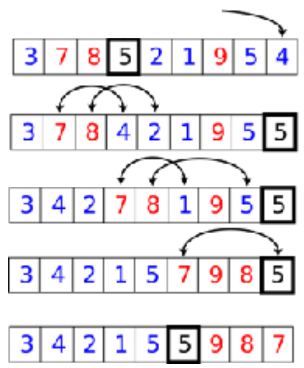





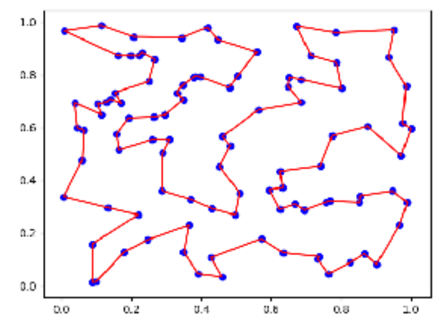


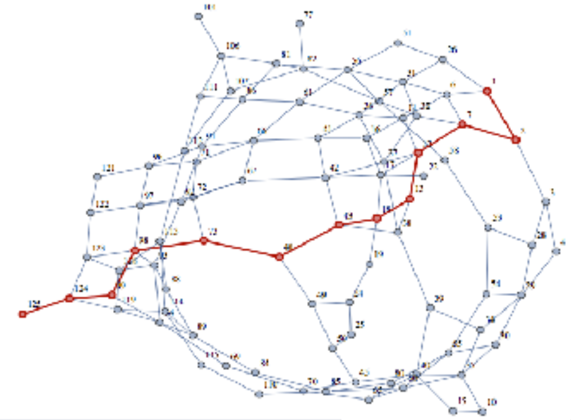
- Tasks with known optimal worst-case complexity (e.g. sorting): can we leverage data distribution to obtain faster algorithms?
- Tasks with unknown optimal complexity (e.g. multiplication): can we learn how to obtain the fastest algorithm?
- NP-hard tasks (e.g. TSP): can we learn efficient approximations?

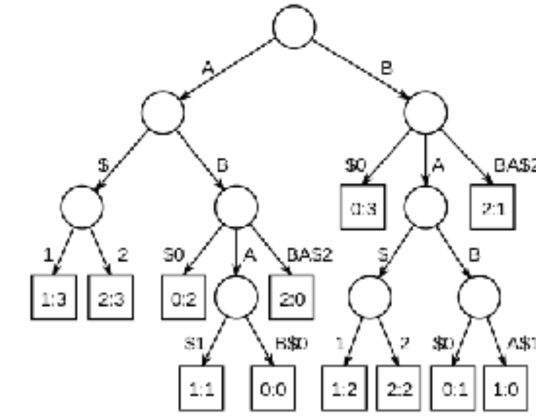
Motivation

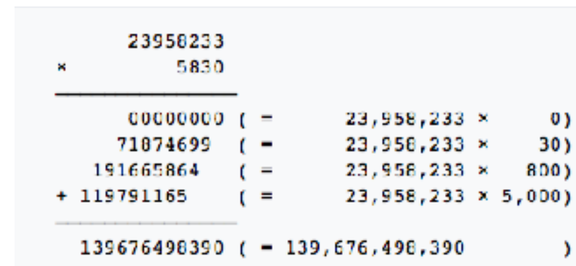


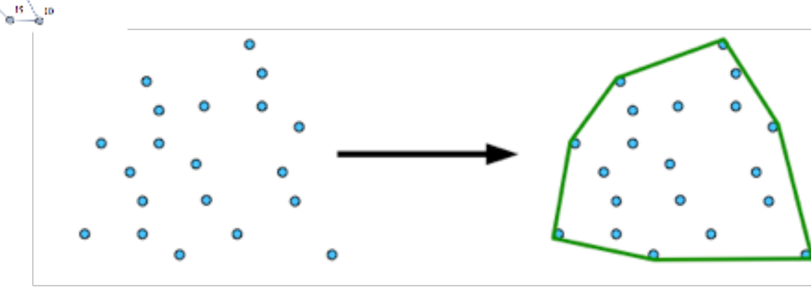








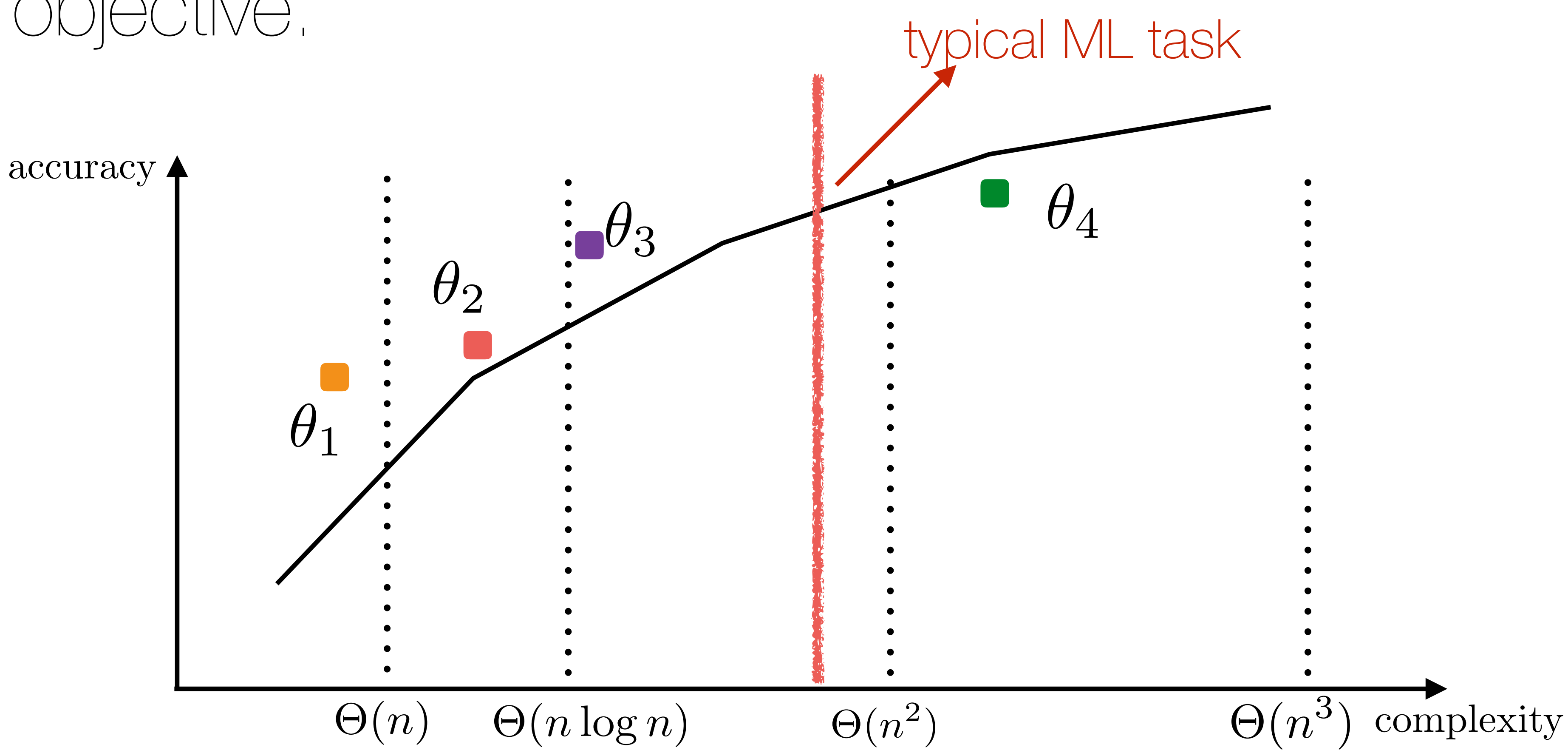




- Tasks with known optimal worst-case complexity (e.g. sorting): can we leverage data distribution to obtain faster algorithms?
- Tasks with unknown optimal complexity (e.g. multiplication): can we learn how to obtain the fastest algorithm?
- NP-hard tasks (e.g. TSP): can we learn efficient approximations?
- **Goal**: learn how to *solve* these tasks *efficiently* from only input-output examples.
  - Need to generalize relative to input size.

Optimizing for Accuracy and Complexity

- We want to add the computational complexity as part of the learning objective.



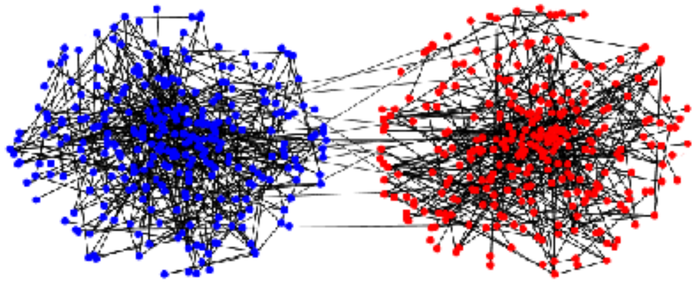
$$\mathcal{L}(\theta) = \mathcal{E}(\theta) + \lambda \mathcal{C}(\theta) \; .$$

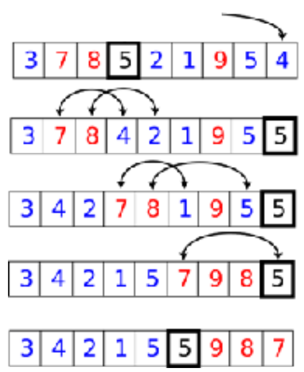
enforces solving the task with high accuracy

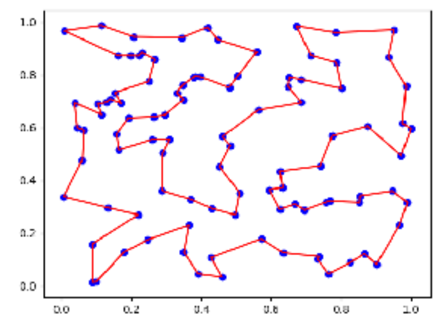
enforces solving the task with small complexity

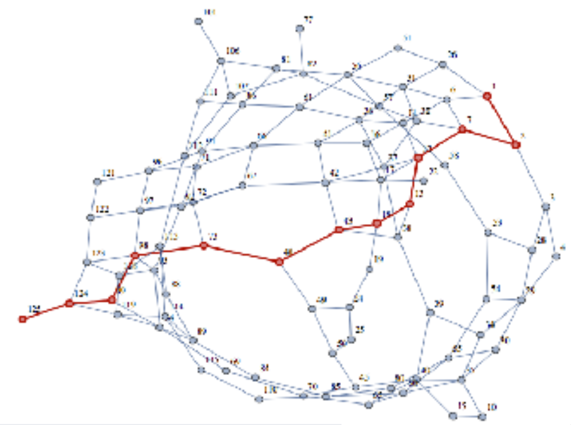
- Q: How to parametrize dynamic computations?

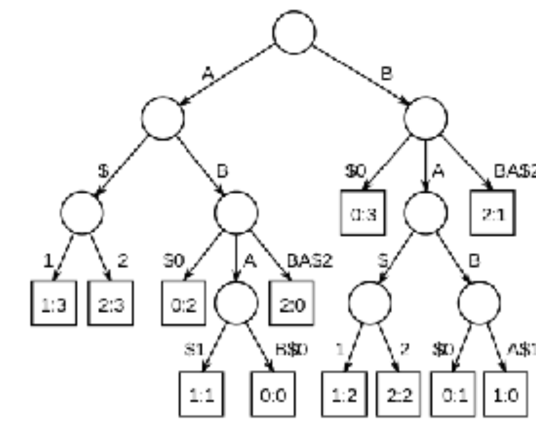
Scale Invariant Algorithmic Tasks

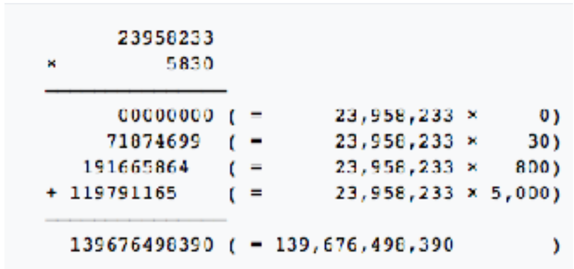


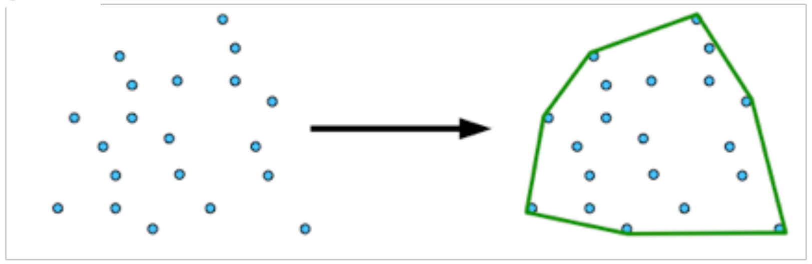






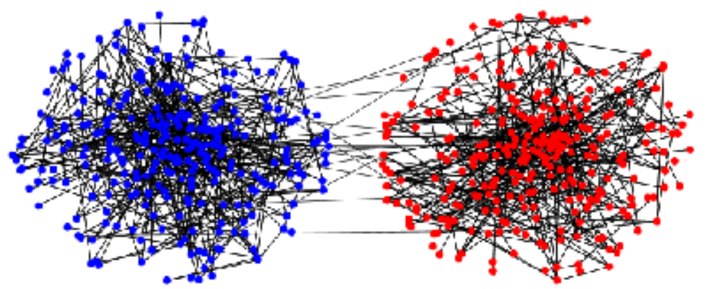


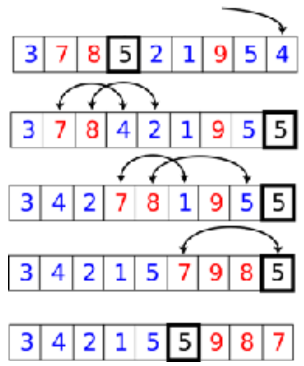


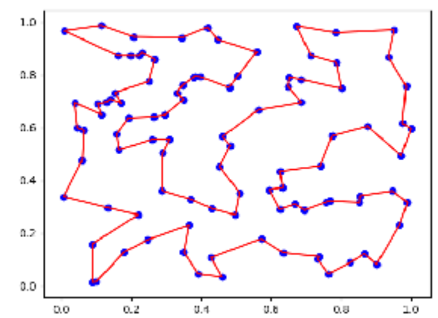


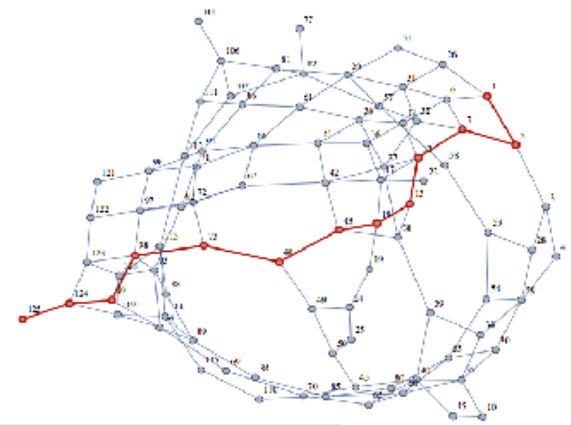
- Many algorithmic and geometric tasks are ***self-similar across scales***:
  - The solution can be expressed in terms of smaller input solutions.
  - This is the basis of dynamic programming and *recursive* algorithms.
  - Controlling the recursion = controlling computational complexity.

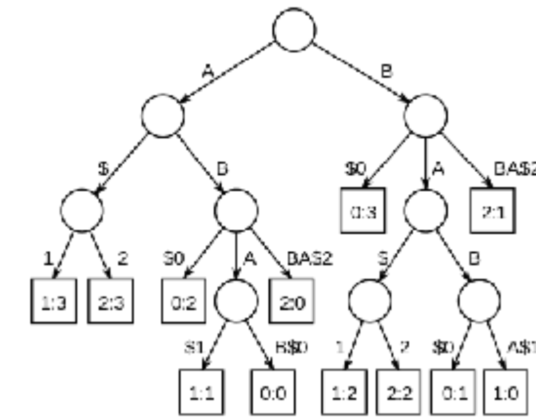
Scale Invariant Algorithmic Tasks

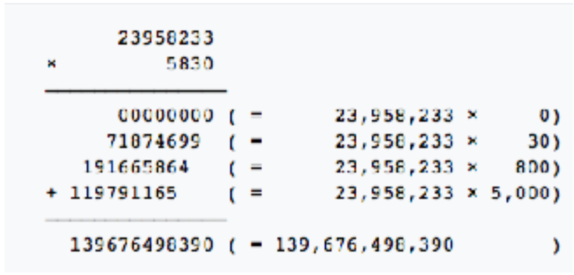


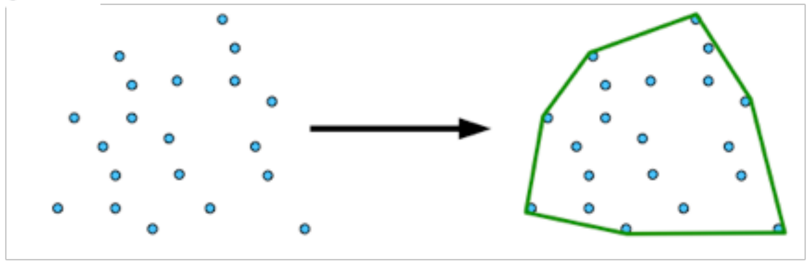












- Many algorithmic and geometric tasks are ***self-similar across scales***:
  - The solution can be expressed in terms of smaller input solutions.
  - This is the basis of dynamic programming and *recursive* algorithms.
  - Controlling the recursion = controlling computational complexity.
- We propose ***Divide and Conquer Networks***: a dynamic neural architecture that learns how to solve tasks using recursion.

Divide and Conquer Networks

- Consider general tasks $\mathcal{T}$ that map an input X to output $\mathcal{T}(X)$.

  $\textbf{input size } |X| = n$

- We consider a recursive decomposition of the task:

$$\mathcal{T}(X) = \mathcal{M}(\mathcal{T}(S_1(X)), \mathcal{T}(S_2(X))) \ ,$$

$$|S_i(X)| < n \ , \ X = S_1(X) \cup S_2(X) \ .$$

Divide and Conquer Networks

- Consider general tasks $\mathcal{T}$ that map an input X to output $\mathcal{T}(X)$.
  $\text{input size } |X| = n$
- We consider a recursive decomposition of the task:

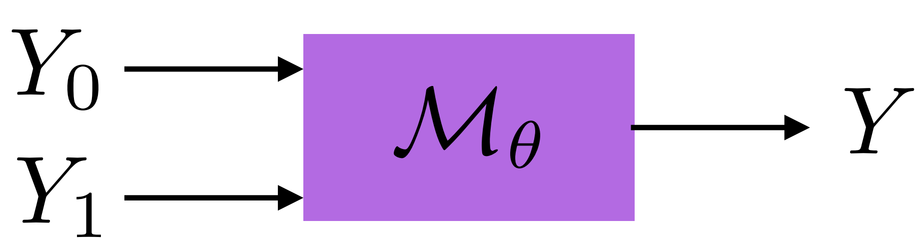
$$\mathcal{T}(X) = \mathcal{M}(\mathcal{T}(S_1(X)), \mathcal{T}(S_2(X))) \ ,$$

$$|S_i(X)| < n \ , \ \ X = S_1(X) \cup S_2(X) \ .$$

- Rather than learning $\mathcal{T}$ directly with e.g. a sequence-to-sequence model, we learn two basic operations:
  - How to ***split*** a given input into two disjoint subsets:

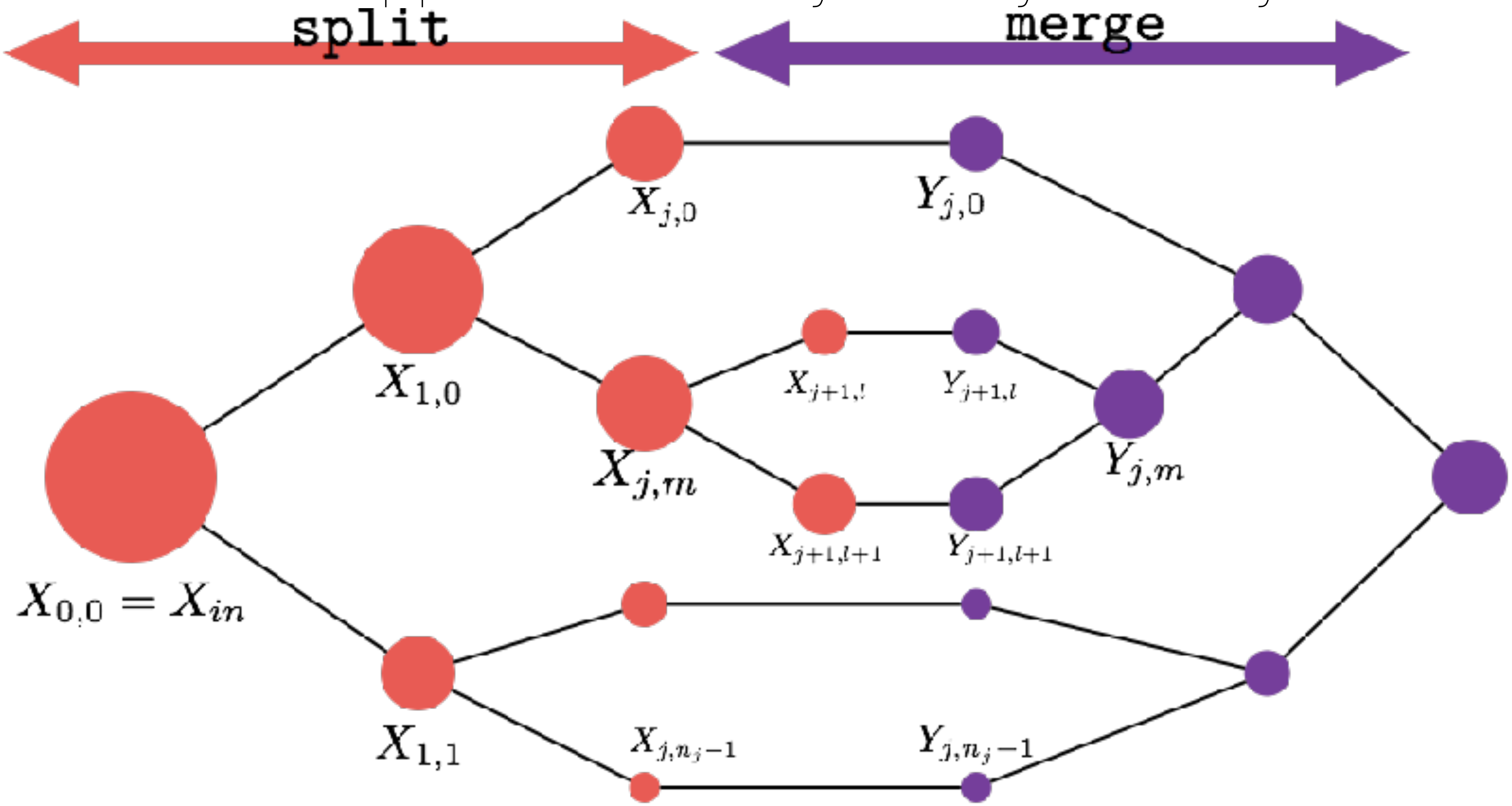
$$X = X_0 \sqcup X_1$$

  - How to ***merge*** two partially solved tasks into a larger one:



Divide and Conquer Networks

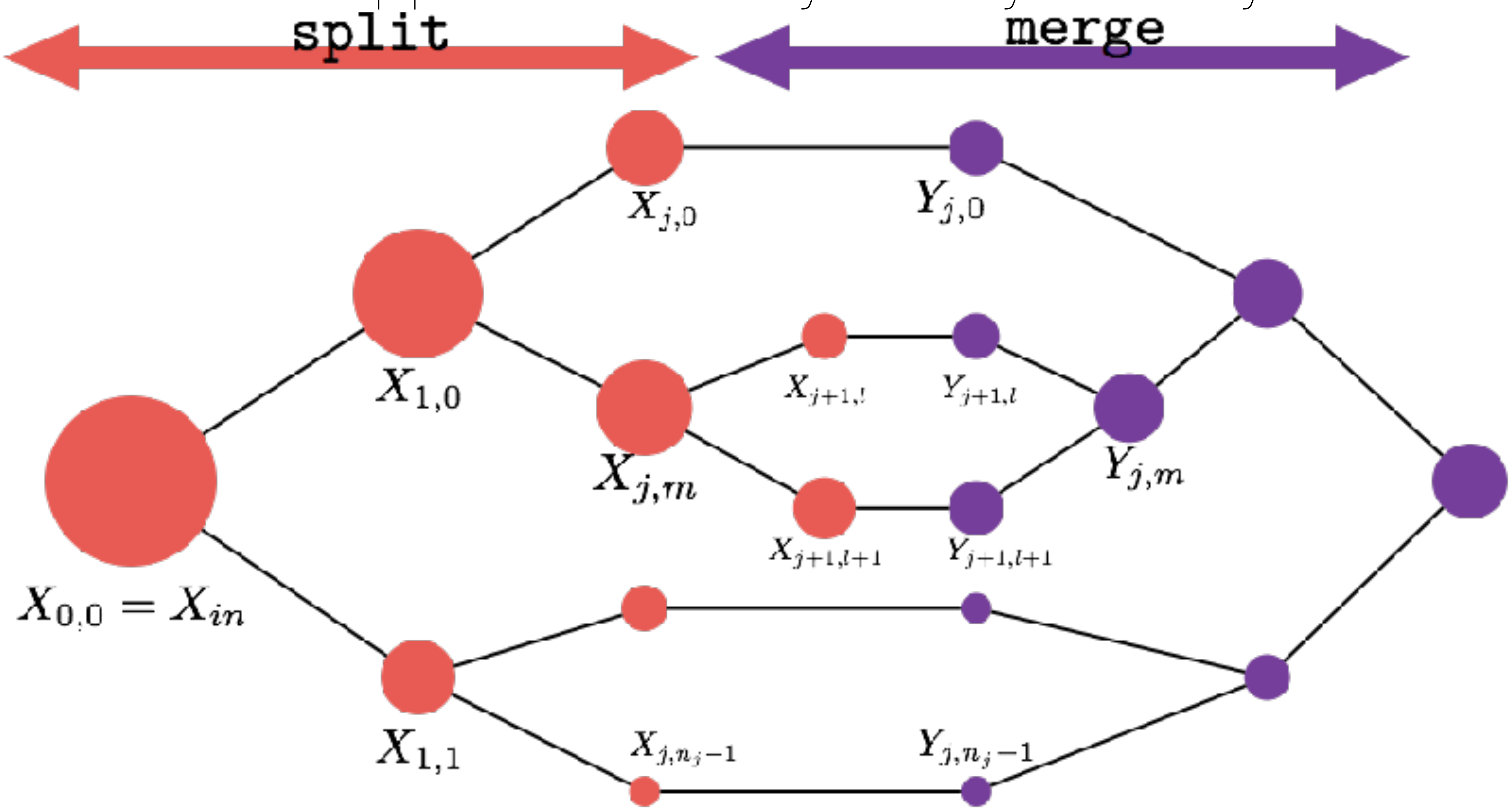
- These blocks are applied recursively and dynamically:



  – Each input generates a different “execution tree”.

Divide and Conquer Networks

- These blocks are applied recursively and dynamically:

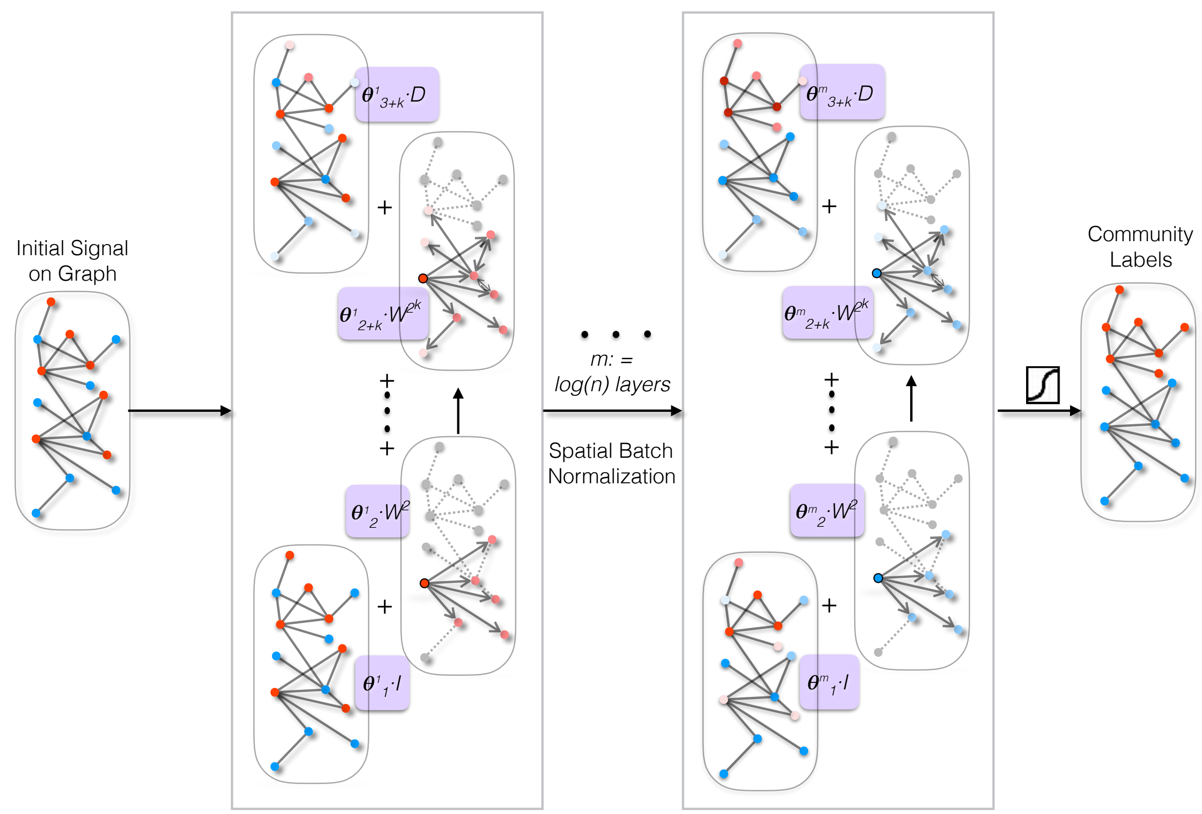


  – Each input generates a different "execution tree".

- Q: How to parametrize those operations?
- How to train the model end-to-end?

Split Model

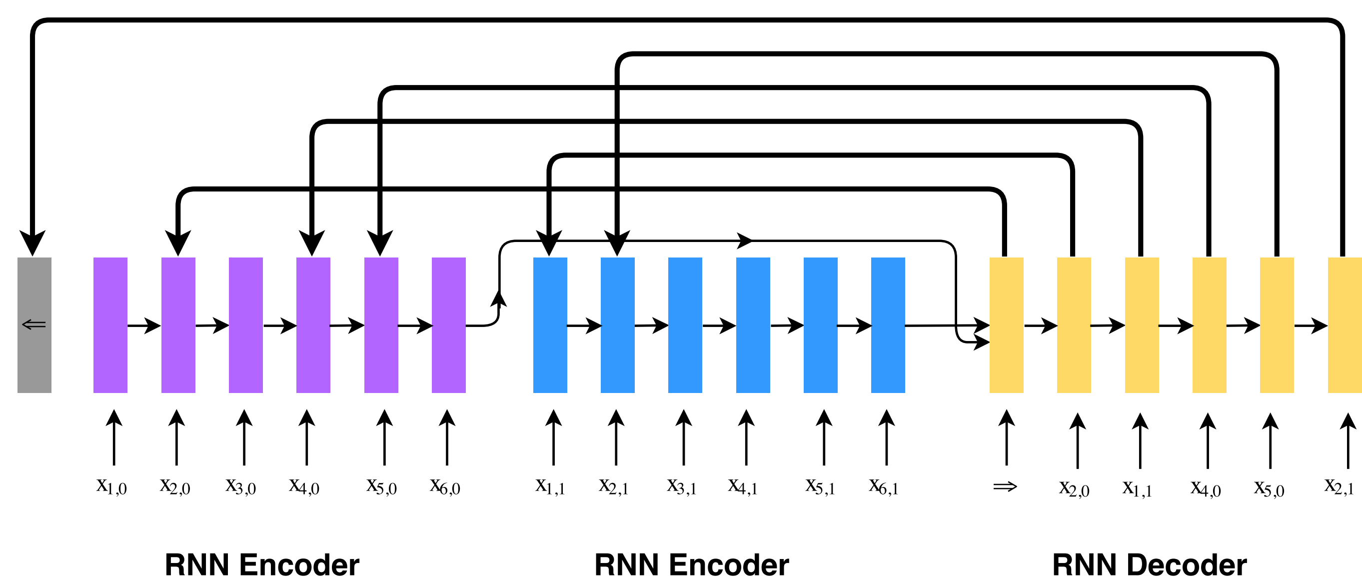
- We consider split models that take as inputs either graphs or sets.
- For graphs, we consider Graph Neural Networks
  - First introduced in [Scarselli et al.'09], [Gori et al'05].
  - Later simplified in [Li et al.'15], [Duvenaud et al.'15] [Sukhbaatar et al.'16].
  - Intimately related to convolutional neural networks using Graph Laplacians [Bruna et al.'16].



- For sets, we consider *set2set* models [Vinyals et al '15], [Sukhbaatar et al.'16]

Merge Model

- Since solutions may be partially ordered, we use a seq2seq model with attention [Bahdanau et al].
  - Generic block with $\Theta(n^2)$ complexity, but can be modulated by regularization.
- In the case where outputs are subsets of input (e.g. convex hull, TSP, sorting), the model is the so-called *Pointer-net* [Fortunato, Vinyals,'15].



  - In that case, cascading merge blocks produces a product of stochastic permutation matrices at each scale.

Training the Model

- Training set: $\{(X^l, Y^l)\}_{l \leq L}$

  $\mathcal{P}(X)$: partition tree associated to X

  Accuracy loss:

θ: split parameters

ϕ: merge parameters

$$\mathcal{E}(\theta, \phi) = \frac{1}{L} \sum_{l \leq L} \mathbb{E}_{\mathcal{P}(X) \sim S_\theta(X)} \log p_\phi(Y^l \mid \mathcal{P}(X^l)) \ .$$

- Merge training:
  - In a pointer network, $p(Y \mid X) = \Gamma X$, Γ: stochastic matrix.
  - In our case,

$$p(Y \mid X) = \left(\prod_{j=0}^{J} \Gamma_j \right) X, \ \Gamma_j\text{: parameter sharing across scales.}$$

Training the Model

- Training set: $\{(X^l, Y^l)\}_{l \leq L}$

  $\mathcal{P}(X)$: partition tree associated to X

  Accuracy loss:

θ: split parameters

ϕ: merge parameters

$$\mathcal{E}(\theta, \phi) = \frac{1}{L} \sum_{l \leq L} \mathbb{E}_{\mathcal{P}(X) \sim S_\theta(X)} \log p_\phi(Y^l \mid \mathcal{P}(X^l)) \ .$$

- Split training:
  - split parameters are separated from the output by sampling steps.
  - we use the policy gradient estimator:

$$\nabla_\theta \mathcal{E}(\theta, \phi) \approx \frac{1}{LK} \sum_{l \leq L} \sum_{\mathcal{P}(X^l)^{(k)} \sim S_\theta(X^l)} \log p_\phi(Y^l \mid \mathcal{P}(X^l)^{(k)}) \nabla_\phi \log p_\theta(\mathcal{P}(X^l)^{(k)} | X^l)$$

  - variance is reduced thanks again to parameter sharing across scales.
  - although other options (e.g relaxing sampling with softmax) possible too.

Computational Complexity as Regularization

- The average case complexity of splitting in our model is

$$C_S(n) = C_S(\alpha_S n) + C_S((1-\alpha_S)n) + \Theta(n) \ ,$$

α_S: average fraction of elements sent to each split output.

- We verify that $C_S(n) \simeq \dfrac{n \log n}{\log \alpha_S^{-1}} \ .$

Computational Complexity as Regularization

- The average case complexity of splitting in our model is

$$C_S(n) = C_S(\alpha_S n) + C_S((1-\alpha_S)n) + \Theta(n) \ ,$$

α_S: **average fraction of elements sent to each split output.**

- We verify that $C_S(n) \simeq \dfrac{n \log n}{\log \alpha_S^{-1}}$.

- Thus computational complexity is controlled by pushing α_S close to 0.5:

$$\mathcal{C}(\theta) = \left(n^{-1} \sum_{i \le n} p_\theta(z_i \mid X) - \frac{1}{2} \right)^2 \ .$$

Experiments

- Sorting
- Convex Hull
- Clustering
- Community Detection on Networks
  - with L. Li (UC Berkeley)
- Quadratic Assignment Problem and TSP
  - with A. Nowak, S. Villar and A. Bandeira (NYU).

Sorting Real Numbers

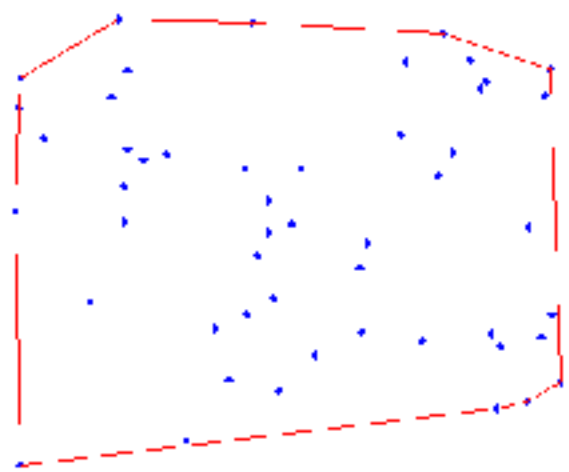
- Toy-task to assess capacity of models.
- Baseline: Pointer Network applied directly to the input.
- DCN (no split). Fix split block to generate a balanced binary tree independent of X.
- DCN (no merge): Fix Merge block to the identity.
- DCN (Joint): Train both blocks.
- We use weak supervision and computational regularization. Train for n=8,16. Test results:

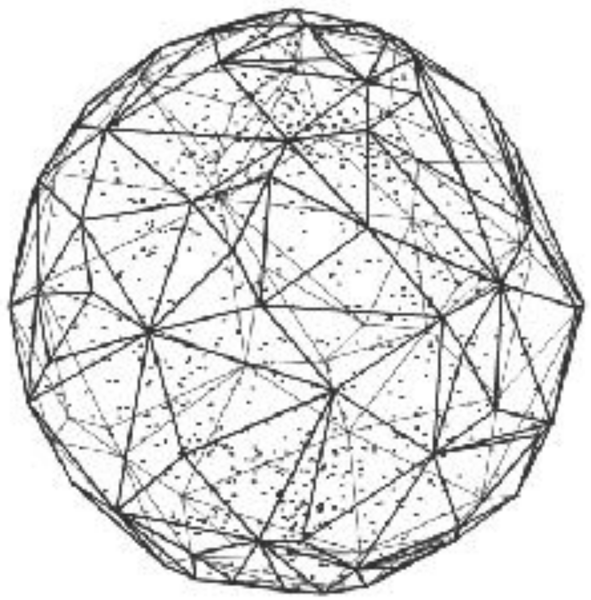
	Baseline	DCN (no Split)	DCN (no Merge)	DCN (Joint)
n=8	80.1	90	**100**	**100**
n=16	31	67	**100**	**100**
n=64	0	0	**99**	0

  – Joint model does not generalize because merge block does not have the appropriate complexity for this task ($\Theta(n^2)$ vs $\Theta(n \log n)$).

Convex Hull

- Given n points in $\mathbb{R}^d$, find the extremal set of points of the polytope of minimum area that contains them all.
  - Task can be solved efficiently in $o(n \log n)$ in the plane



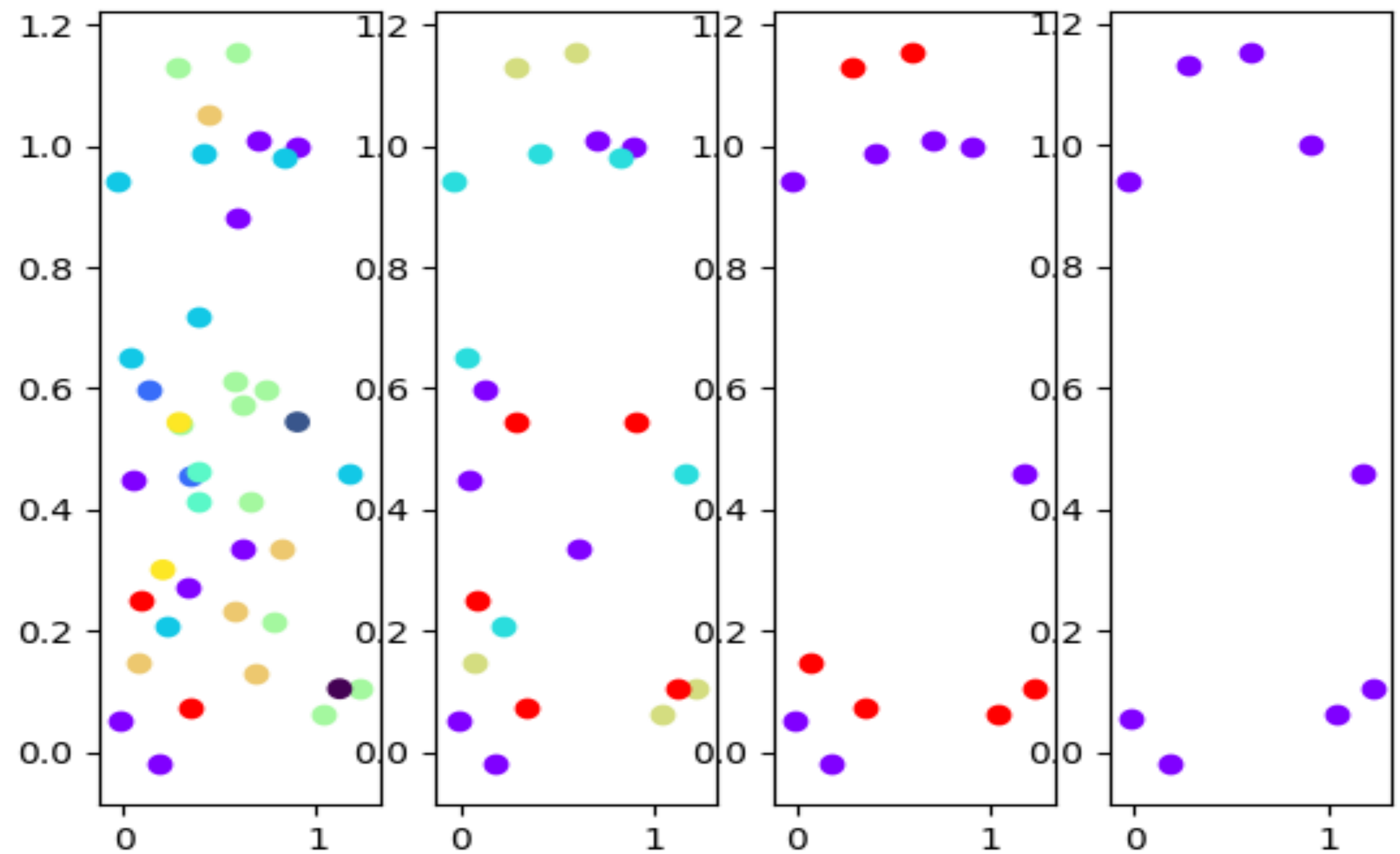


	n=25	n=50	n=100	n=200
Baseline	81.3	65.6	41.5	13.5
DCN Random Split	59.8	37.0	23.5	10.29
DCN	88.1	83.7	73.7	52.0
DCN + Split Reg	**89.8**	**87.0**	**80.0**	**67.2**

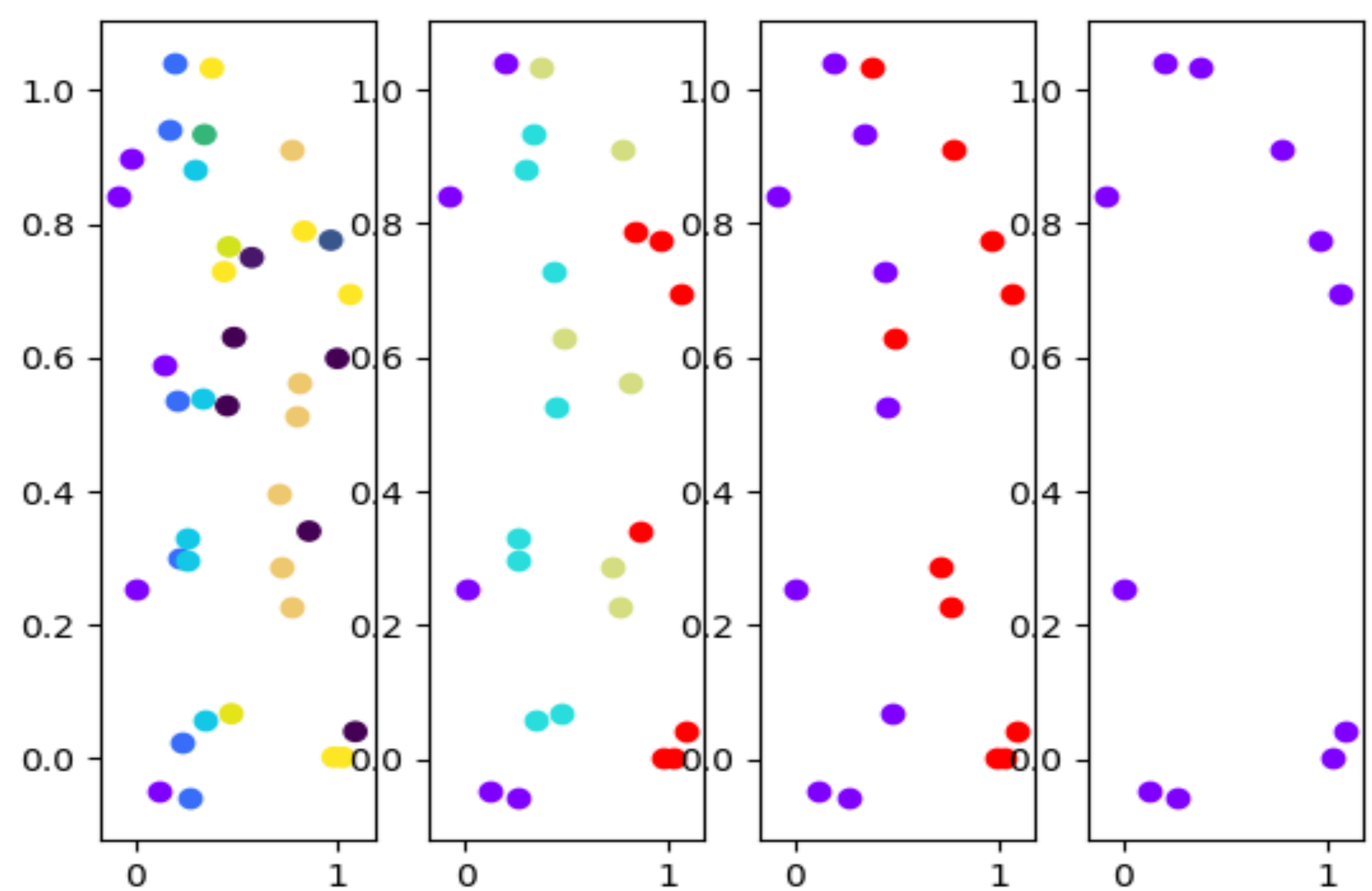
Baseline: seq-2-seq model (PtrNet)

Convex Hull

- 2 Scales (Random Split):



- 2 Scales (Learnt Split):



Euclidean K-Means

- Given $\boldsymbol{n}$ points of $\mathbb{R}^{\boldsymbol{d}}$, solve the following combinatorial optimization problem:

$$\min_{\mathcal{P}(X)} \sum_{i \in \mathcal{P}(X)} n_i \sigma_i^2$$

σ_i^2: variance of subset i of partition projected on an (unknown) subspace.
n_i: cardinality of subset i.

Euclidean K-Means

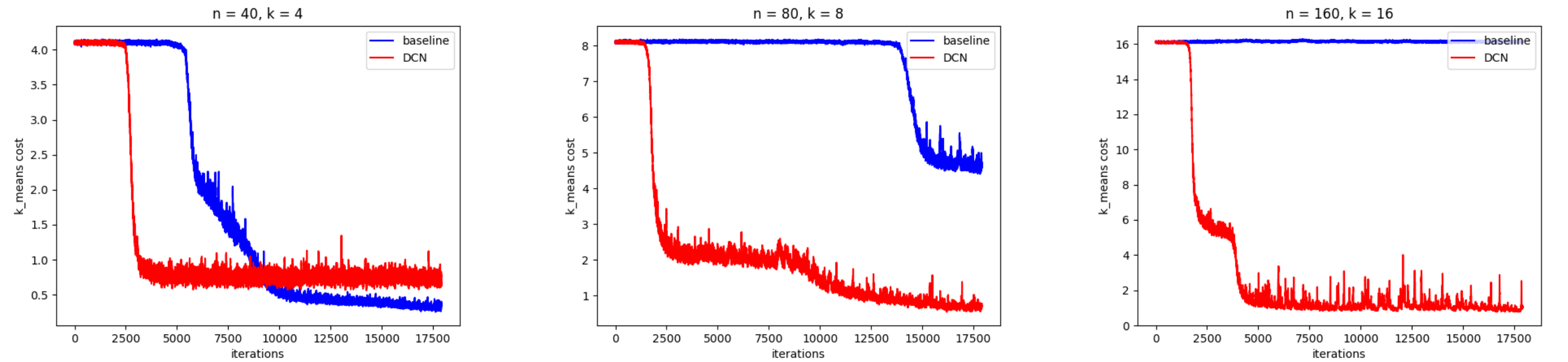
- Given n points of $\mathbb{R}^d$, solve the following combinatorial optimization problem:

$$\min_{\mathcal{P}(X)} \sum_{i \in \mathcal{P}(X)} n_i \sigma_i^2$$

σ_i^2: variance of subset i of partition projected on an (unknown) subspace.
n_i: cardinality of subset i.

  – Here the output is only specified with split parameters.

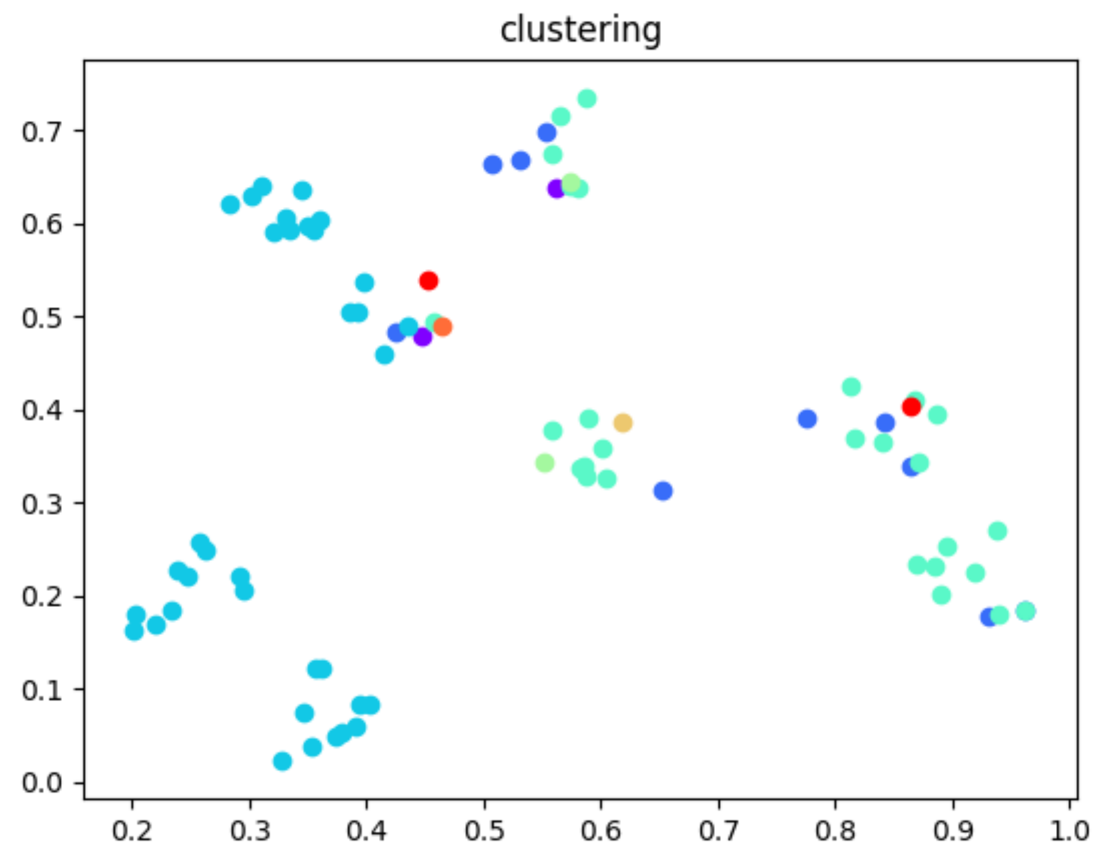
	k=4, n=40	k=8, n=80	k=16, n=160
Baseline	**0.35**	4.62	16.08
DCN	0.85	**0.86**	**0.86**



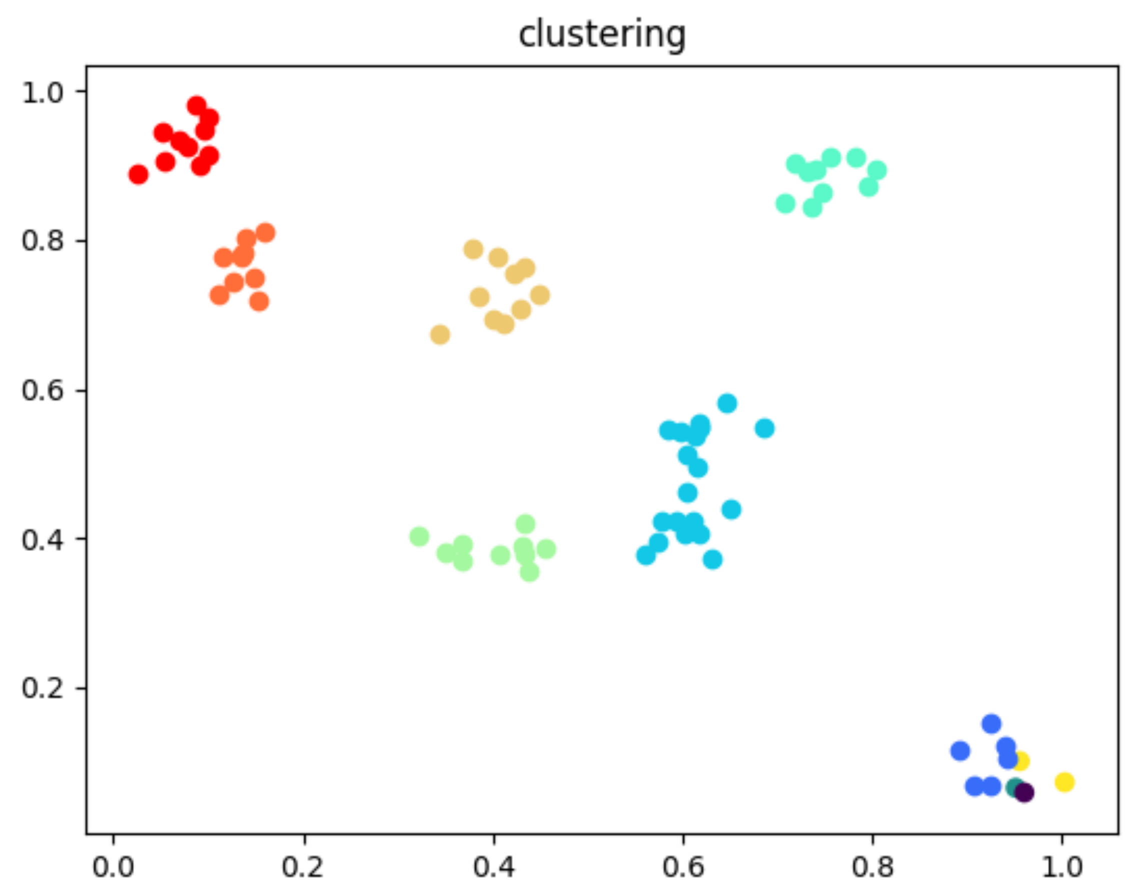
baseline: train a direct policy (J=1)

Euclidean K-Means

- Baseline, k=8 and n=80.



- DCN, k=8 and n=80.



Summary So Far

- So far, we have shown a general trainable architecture to exploit scale invariance in algorithmic/ geometric tasks.
  - Significant gains over baselines agonistic to recursion.
  - Computational and Accuracy Tradeoffs by gradient descent.
  - Shown on tasks where we currently have optimal, efficient algorithmic solutions.

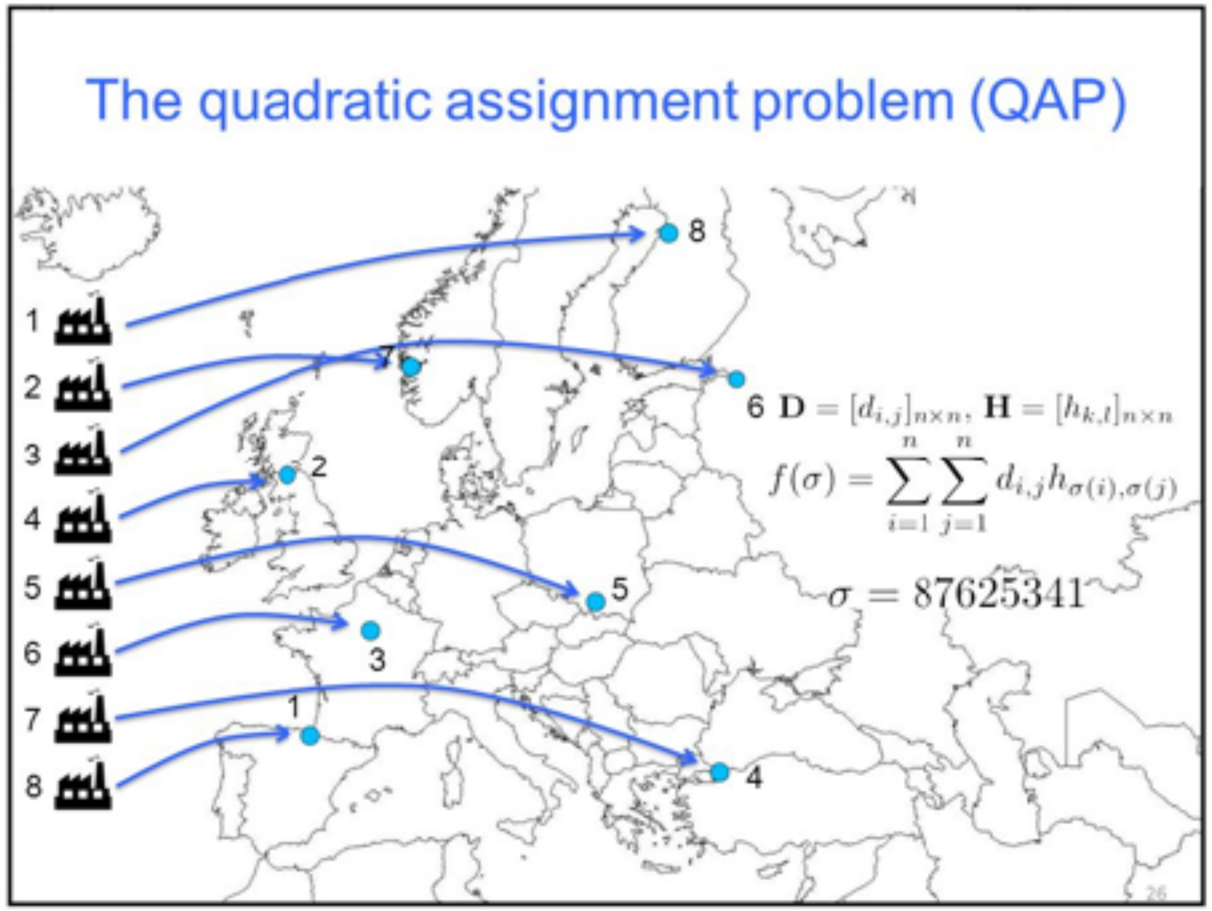
Summary So Far

- So far, we have shown a general trainable architecture to exploit scale invariance in algorithmic/ geometric tasks.
  - Significant gains over baselines agonistic to recursion.
  - Computational and Accuracy Tradeoffs by gradient descent.
  - Shown on tasks where we currently have optimal, efficient algorithmic solutions.

- Current work: use this machinery on tasks that are computationally and statistically *hard,* or where no optimal algorithm is known.
  - Community Detection in Noisy Graphs.
  - Quadratic Assignment Problem
    - Travelling Salesman Problem
  - Matrix Multiplication.

Quadratic Assignment Problem

[with S. Villar (NYU), A. Nowak (NYU) and A. Bandeira (NYU)]



- Find an assignment that optimizes the transportation cost between two graphs:

$$\min_{X \in \Pi_n} \mathrm{Tr}(A_1 X A_2 X^T) \ .$$

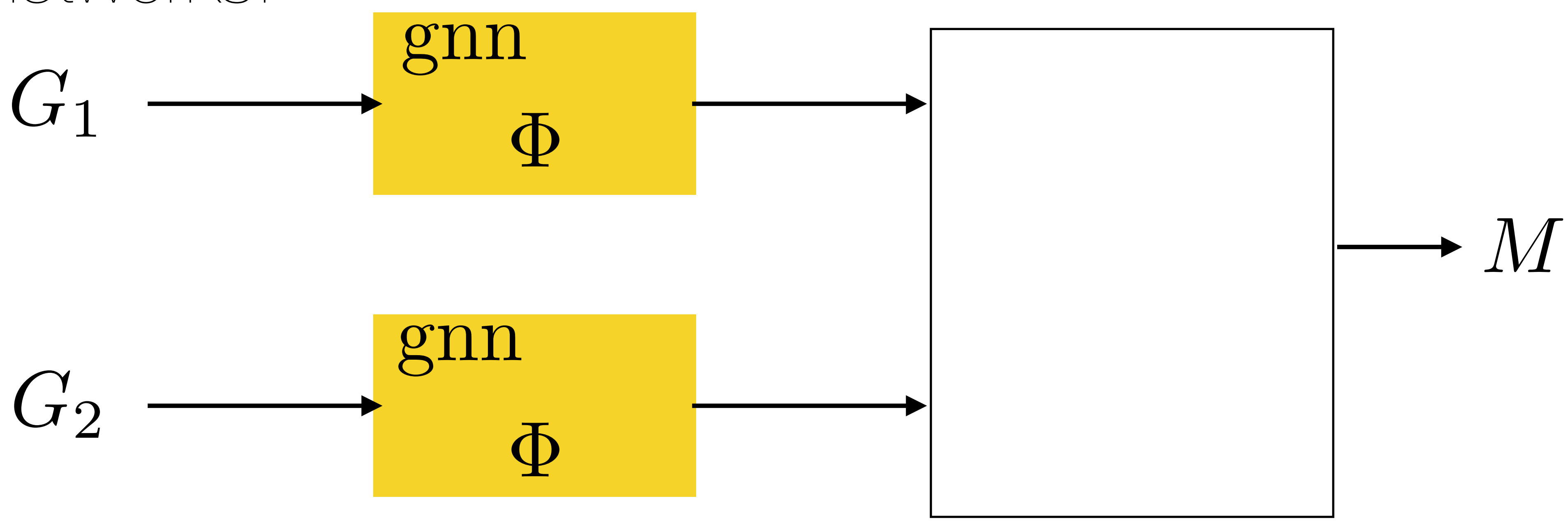
Π_n: space of $n \times n$ permutation matrices.

  - NP-hard
  - Contains the TSP as a particular instance.
  - Relaxations using SDP and Spectral Approaches.

Quadratic Assignment Problem

[with S. Villar (NYU), A. Nowak (NYU) and A. Bandeira (NYU)]

- We learn approximate solutions using siamese graph neural networks:



$$M = \mathrm{softmax}[\Phi(G_1)\Phi(G_2)^T)]$$

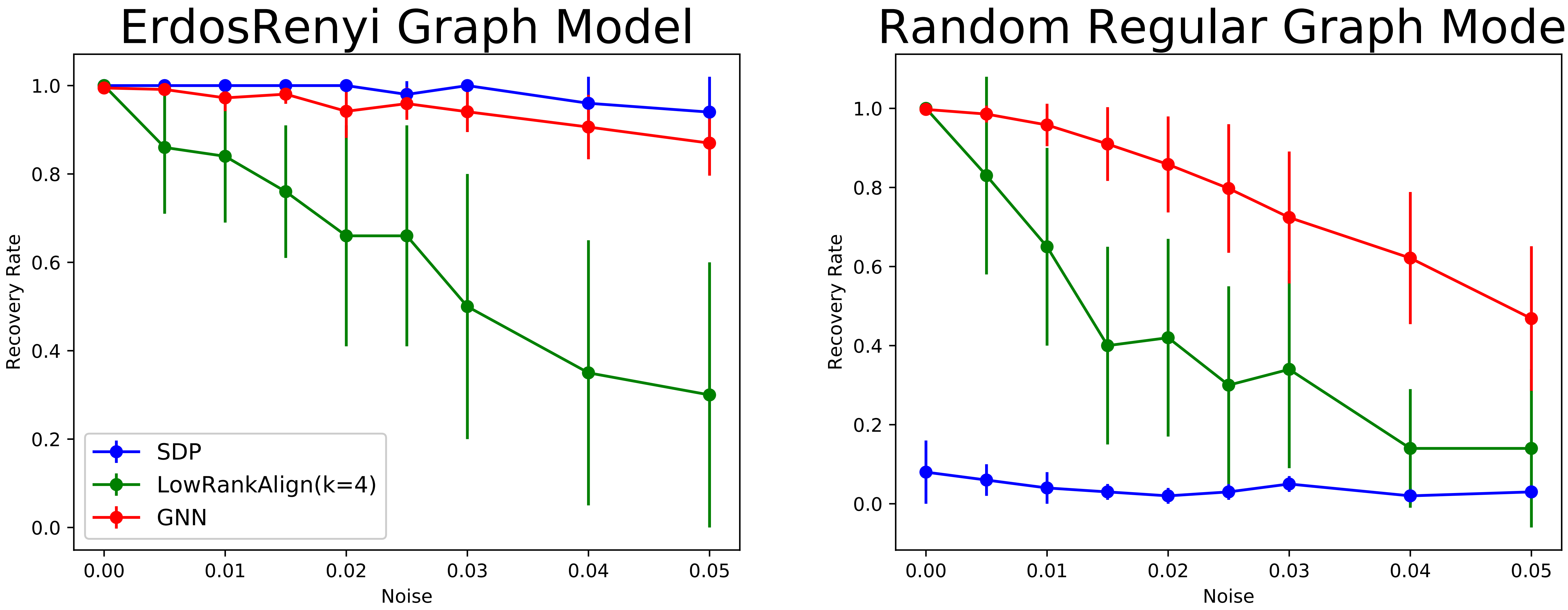
- We train the model to predict the correct permutation matrix on a dataset of the form $G_1 = PG_2 + N$

$N \sim$ Erdos-Renyi

  - $G_2 \sim$ Erdos-Renyi
  - $G_2 \sim$ Random Regular

Quadratic Assignment Problem

[with S. Villar (NYU), A. Nowak (NYU) and A. Bandeira (NYU)]



- Our model runs in $o(n^2)$
- LowRankAlign is $o(n^3)$
- SDP runs in $o(n^4)$

Travelling Salesman Problem

[with S. Villar (NYU), A. Nowak (NYU) and A. Bandeira (NYU)]

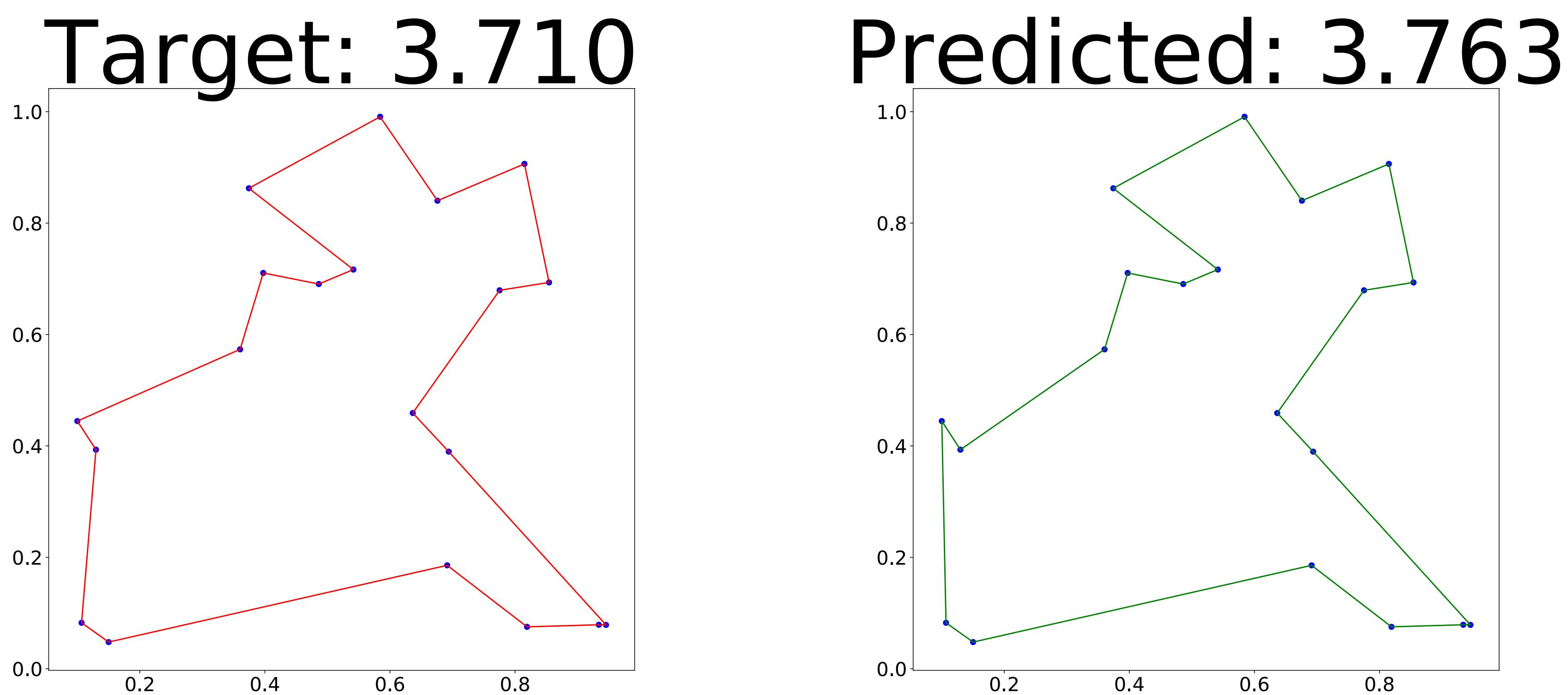
- A particular case of the QAP is the Travelling salesman Problem:

$$\min_{X \in \Pi_n} \mathrm{Tr}(A_1 X A_2 X^T) \ .$$

A_1: dense pairwise distance matrix

A_2: n-cycle adjacency matrix

- We obtain preliminary good results



  - Not yet like best Heuristics in the planar case (Christofides algorithm).
  - Ongoing work.

Conclusions

- Neural Networks and Dynamic Programming.
  - Many tasks in real life contain some form of scale invariance (e.g. navigation, planning, scheduling, ranking, …).
  - Real need for accuracy-complexity tradeoffs.
  - Our architecture is a step towards end-to-end learning that exploits such structure.
  - Challenges ahead: more flexible models allowing wider range of accuracy-complexity tradeoffs.
- Neural Networks and Complexity.
  - Surprising, Intriguing ability to reach detection thresholds known to be hard on the community detection.
  - Very efficient data-driven alternative to attack hard combinatorial optimization problems.
  - Challenges ahead: understand what are the fundamental limits of data-driven

Thank you!

References:
"Divide and Conquer Networks", A. Nowak and J. Bruna, submitted. (https://arxiv.org/abs/1611.02401)
"Community Detection with Graph Neural Networks", J. Bruna and L. Li (https://arxiv.org/abs/1705.08415)
"A Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks", A. Nowak, S. Villar, A. Bandeira, J. Bruna, submitted.