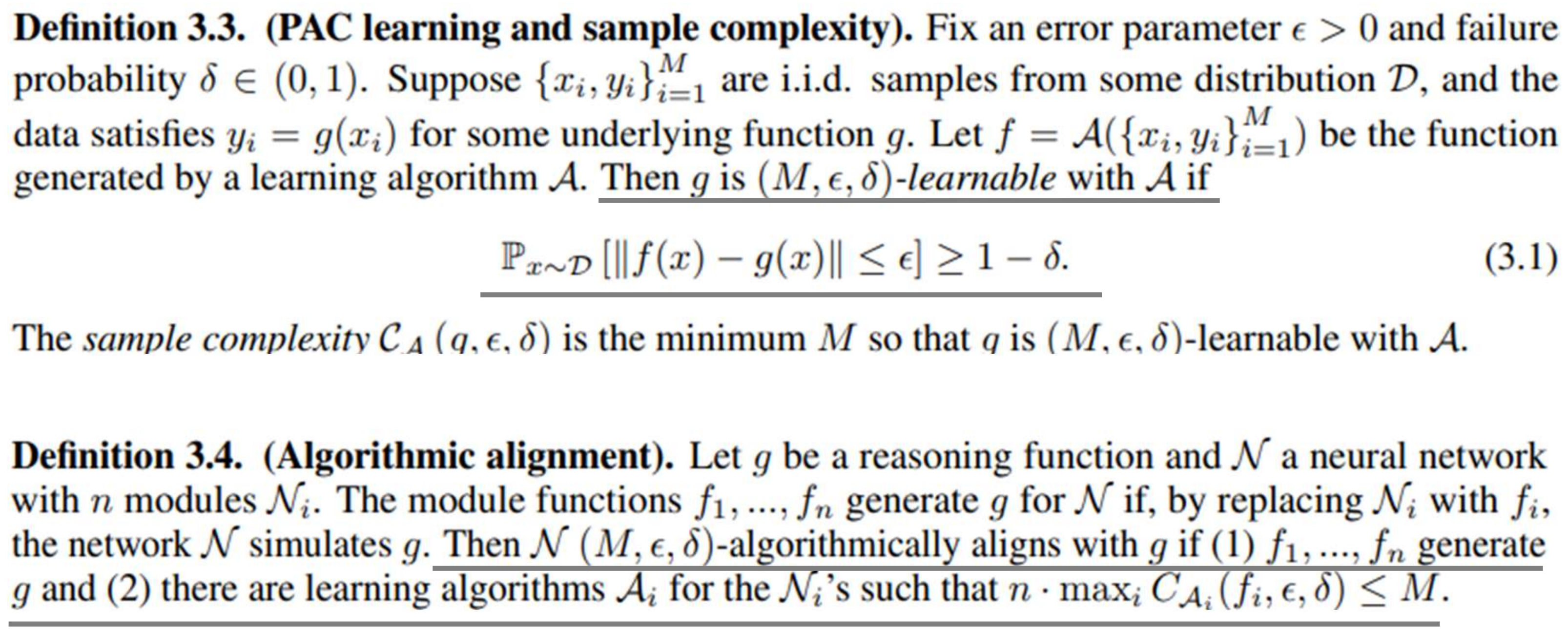
Notes on Reinforcement learning papers related to Algorithmic Alignment

- During my second semester studying graph neural networks and reinforcement learning, I noticed multiple works in GNN reasoning and its relation to algorithms, while similar concepts parallely appeared in RL planning. I liked how these works studied algorithmic inductive bias as opposed to the widely-studied representational one + lead to a field that also empirically works. The slides were refined from my lab seminar material I presented at the end of that semester.
- Additional Refs:
  +) the notes were directly inspired by Petar Velikovic's talks
  https://www.youtube.com/watch?v=1HhTpH3ZXMY&t=882s
  https://www.youtube.com/watch?v=J46Wp2RjQCA
  +) his work showing equivalence between GNN and dynamic programming using integral transform in category theory and improving GNN's edge-centric algorithms solving performance : https://arxiv.org/abs/2203.15544
  +) related talk material at a NeurIPS 2021 tutorial
  https://neurips.cc/media/neurips-2021/Slides/21897.pdf

1. Algorithmic Alignment Definition

- Xu et al. (2020) [1] established formal definition for algorithmic alignment based on PAC learning framework [2]



Definition 3.3. (PAC learning and sample complexity). Fix an error parameter $\epsilon > 0$ and failure probability $\delta \in (0, 1)$. Suppose $\{x_i, y_i\}_{i=1}^{M}$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $y_i = g(x_i)$ for some underlying function g. Let $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^{M})$ be the function generated by a learning algorithm $\mathcal{A}$. Then g is (M, ϵ, δ)-*learnable* with $\mathcal{A}$ if

$$\mathbb{P}_{x \sim \mathcal{D}} \left[\| f(x) - g(x) \| \leq \epsilon \right] \geq 1 - \delta. \tag{3.1}$$

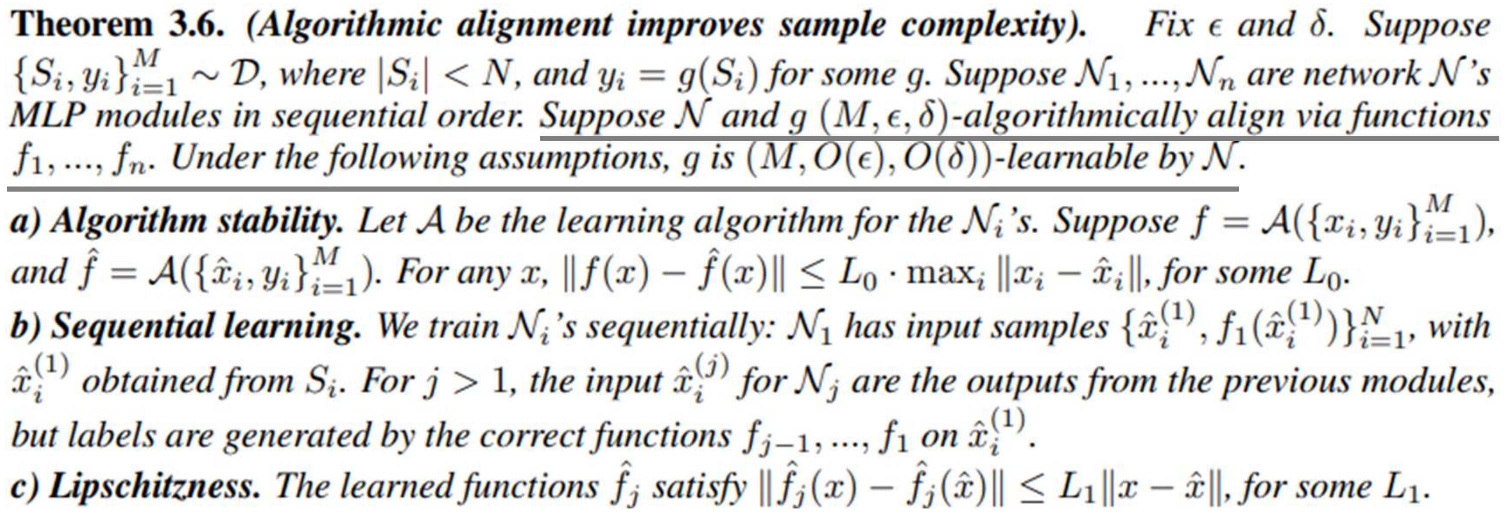
The *sample complexity* $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$ is the minimum M so that g is (M, ϵ, δ)-learnable with $\mathcal{A}$.

Definition 3.4. (Algorithmic alignment). Let g be a reasoning function and $\mathcal{N}$ a neural network with n modules $\mathcal{N}_i$. The module functions $f_1, ..., f_n$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i, the network $\mathcal{N}$ simulates g. Then $\mathcal{N}$ (M, ϵ, δ)-algorithmically aligns with g if (1) $f_1, ..., f_n$ generate g and (2) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that $n \cdot \max_i \mathcal{C}_{\mathcal{A}_i}(f_i, \epsilon, \delta) \leq M$.

[1] Xu, K., Li, J., Zhang, M., Du, S.S., Kawarabayashi, K., Jegelka, S. (2020). What Can Neural Networks Reason About?, ICLR (https://arxiv.org/pdf/1905.13211)
[2] Valiant, L. (1984) A theory of the learnable,. In Proceedings of the sixteenth annual ACM symposiumon Theory of computing

1. Algorithmic Alignment Theory

- If neural network N and a function g are (M, ε, δ)-algorithmically aligned via $f_i\ i \in [\mathrm{n}]$, g is $\left(M, O(\varepsilon), O(\delta)\right)$-learnable by N if f_i satisfies stability, sequential learning and Lipschitzness conditions



Theorem 3.6. ***(Algorithmic alignment improves sample complexity).*** *Fix ϵ and δ. Suppose $\{S_i, y_i\}_{i=1}^{M} \sim \mathcal{D}$, where $|S_i| < N$, and $y_i = g(S_i)$ for some g. Suppose $\mathcal{N}_1, ..., \mathcal{N}_n$ are network $\mathcal{N}$'s MLP modules in sequential order. Suppose $\mathcal{N}$ and g (M, ϵ, δ)-algorithmically align via functions $f_1, ..., f_n$. Under the following assumptions, g is $(M, O(\epsilon), O(\delta))$-learnable by $\mathcal{N}$.*

a) Algorithm stability. *Let $\mathcal{A}$ be the learning algorithm for the $\mathcal{N}_i$'s. Suppose $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^{M})$, and $\hat{f} = \mathcal{A}(\{\hat{x}_i, y_i\}_{i=1}^{M})$. For any x, $\|f(x) - \hat{f}(x)\| \leq L_0 \cdot \max_i \|x_i - \hat{x}_i\|$, for some L_0.*

b) Sequential learning. *We train $\mathcal{N}_i$'s sequentially: $\mathcal{N}_1$ has input samples $\{\hat{x}_i^{(1)}, f_1(\hat{x}_i^{(1)})\}_{i=1}^{N}$, with $\hat{x}_i^{(1)}$ obtained from S_i. For $j > 1$, the input $\hat{x}_i^{(j)}$ for $\mathcal{N}_j$ are the outputs from the previous modules, but labels are generated by the correct functions $f_{j-1}, ..., f_1$ on $\hat{x}_i^{(1)}$.*

c) Lipschitzness. *The learned functions $\hat{f}_j$ satisfy $\|\hat{f}_j(x) - \hat{f}_j(\hat{x})\| \leq L_1 \|x - \hat{x}\|$, for some L_1.*

$\Rightarrow$ The more algorithmically aligned a function is,
the less data is needed to reach the same accuracy for approximating an algorithm
(good/natural inductive bias for algorithm imitation)

1. Algorithmic Alignment Proof by Induction

- With sample size M, if individual module f_i $i \in [\mathrm{n}]$ is guaranteed to have an error less than ε to the true function with probability greater than 1 - δ, sequentially applying the modules for $i \in [\mathrm{n}]$ only at most linearly increases the error ($O(\varepsilon)$) with probability 1-$O(\delta)$ when algorithmic stability, sequential learning and Lipschitzness of the network are satisfied

Many layers (proof by induction)

*First module: by definition of algorithmic alignment

f: learned k+1th module

true k+1th module

$$\|\hat{f}(\hat{z}) - f(z)\| = \|\hat{f}(\hat{z}) - \hat{f}(z) + \hat{f}(z) - f(z)\|$$

z: output of learned kth module

output of true kth module

$$\leq \|\hat{f}(\hat{z}) - \hat{f}(z)\| + \|\hat{f}(z) - f(z)\|$$

$O(\varepsilon)$ by Lipschitz

$$\|\hat{f}(z) - f(z)\| = \|\hat{f}(z) - \tilde{f}(z) + \tilde{f}(z) - f(z)\|$$
$$\leq \|\hat{f}(z) - \tilde{f}(z)\| + \|\tilde{f}(z) - f(z)\|$$
$$\leq L_0 \max \|z_i - \hat{z}_i\| + \epsilon \quad \text{w.p.} \geq 1 - \delta$$

$O(\varepsilon)$ by Algorithm stability

ε by algorithmic alignment

One layer

Bellman-Ford algorithm

```
for k = 1 ... |S| - 1:
  for u in S:
    d[k][u] = min_v d[k-1][v] + cost (v, u)
```

Graph Neural Network

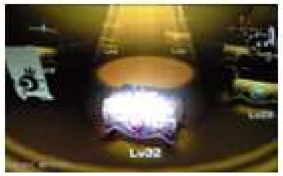
```
for k = 1 ... GNN iter:
  for u in S:
    h_u^(k) = Σ_v MLP(h_v^(k-1), h_u^(k-1))
```

No need to learn for-loops

1. Algorithmic Alignment Empiricals

- Multiple NN architectures (MLP, Deep Sets, GNN) were implemented to solve different kinds of (combinatorial) reasoning tasks where the more algorithmically aligned NN showed better performances given the same sample size

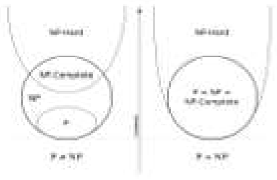
Reasoning Tasks



Summary statistics
What is the maximum value difference among treasures?

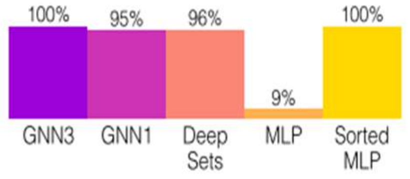
Relational argmax
What are the colors of the furthest pair of objects?

Dynamic programming
What is the cost to defeat monster X by following the optimal path?

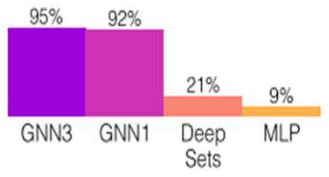


NP-hard problem
Subset sum: Is there a subset that sums to 0?

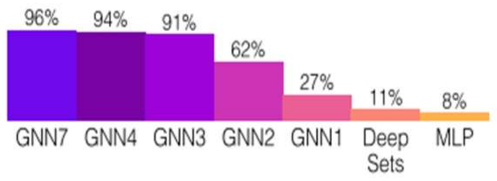
Performance



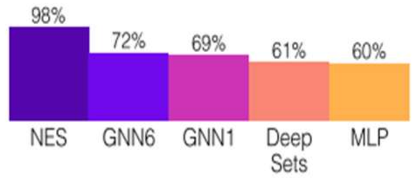
(a) Maximum value difference.



(b) Furthest pair.



(c) Monster trainer.



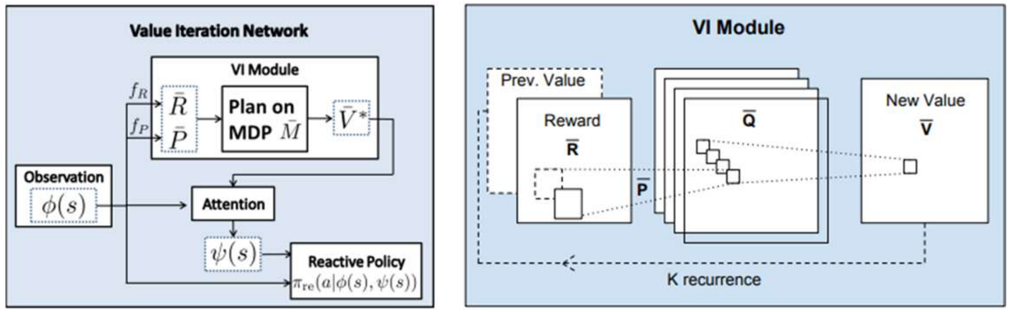
(d) Subset sum. Random guessing yields 50%

2. Architectures related to Algorithmic Alignment

- Earlier work used recurrent CNN or Graph Convolution to imitate Value Iteration and demonstrated empirical improvement

Value Iteration Network [1]

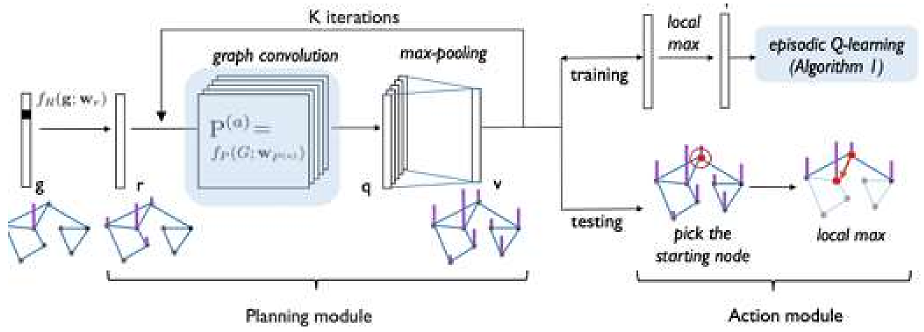
Pioneer work that imitated value iteration with differentiable neural networks (CNN)



- VI Module transforms reward (R) into state value (Q) latent, where comparison across channels (representing actions) results in the optimal value (V) function
- Mostly based on Imitation Learning

Generalized Value Iteration Network [2]

Follow-up work that improved value iteration networks by using Graph Convolution than CNN



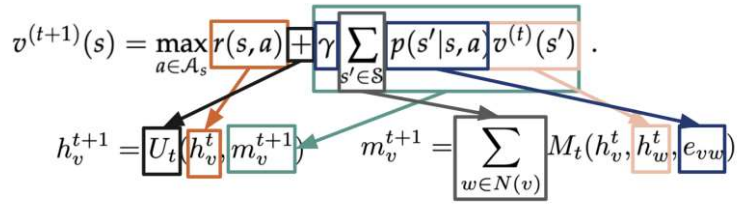
- Planning Module is based on the same logic - CNN was replaced by graph convolution followed by max-pooling
- Episodic Q-learning was adopted for a more stable learning

[1] Tamar, A., Wu, Y., Thomas, G., Levine, S., Abbeel, P. (2016). Value Iteration Networks, NIPS
[2] Niu, S., Chen, S., Guo, H., Targonski, C., Smith, M.C., Kovačević, J. (2017). Generalized Value Iteration Networks: Life Beyond Lattices

2. Architectures related to Algorithmic Alignment

- Graph Neural Induction of Value Iteration [1] made GNN capable of executing VI updates on arbitrary MDP graphs, based on the similarity between GNN message passing and bellman update

Methods



$$v^{(t+1)}(s) = \max_{a\in\mathcal{A}_s} r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) v^{(t)}(s') \ .$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \qquad m_v^{t+1} = \sum_{w\in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

- Build separate graphs per action with
  - node feature: value v(s) and reward r(s,ai)
  - edge feature: discount factor γ and transition p(s'|s,ai)
- Do message passing where
  - M (message passing) aligns with the operation of multiplying p and v and summing them over neighbours s'
  - U (readout) aligns with adding reward to the discounted neighbouring rewards
  - maximization over actions as in
  *note the permutational invariance in both MPNN and VI
- The above alignment doesn't mean that it is directly trained to perform such operations; it's based on supervision on intermediate VI steps (teacher forcing) then rollout at test time

[1] Deac, A., Bacon, P.-L., Tang, J. (2020). Graph Neural Induction of Value Iteration. ICML GRL+

Results

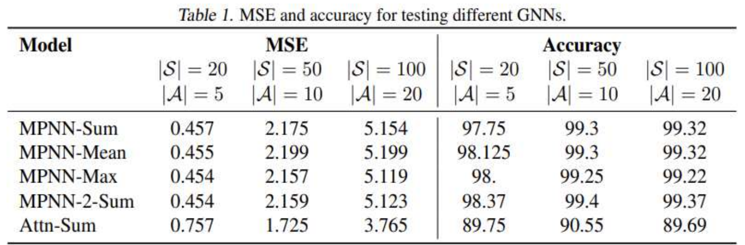


Table 1. MSE and accuracy for testing different GNNs.

Model	MSE			Accuracy		
	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$
MPNN-Sum	0.457	2.175	5.154	97.75	99.3	99.32
MPNN-Mean	0.455	2.199	5.199	98.125	99.3	99.32
MPNN-Max	0.454	2.157	5.119	98.	99.25	99.22
MPNN-2-Sum	0.454	2.159	5.123	98.37	99.4	99.37
Attn-Sum	0.757	1.725	3.765	89.75	90.55	89.69

Generalizes well to enlarged state and action spaces
*Attn-sum's peculiar results may be due to its scale-preservation

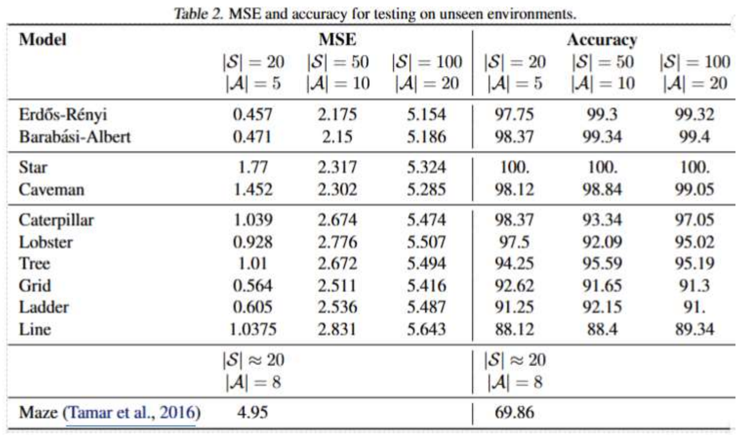


Table 2. MSE and accuracy for testing on unseen environments.

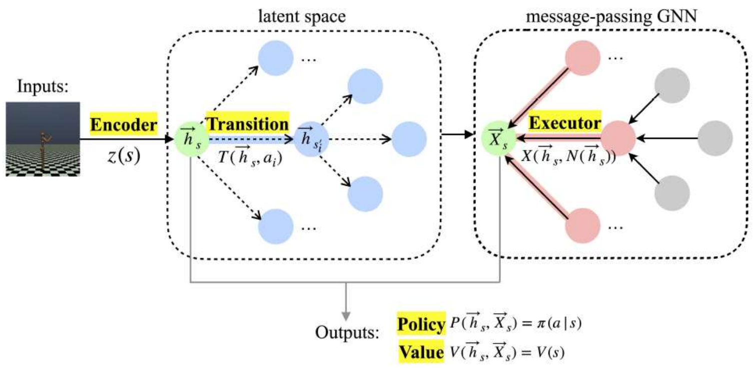
Model	MSE			Accuracy		
	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$	$\lvert\mathcal{S}\rvert = 20$ $\lvert\mathcal{A}\rvert = 5$	$\lvert\mathcal{S}\rvert = 50$ $\lvert\mathcal{A}\rvert = 10$	$\lvert\mathcal{S}\rvert = 100$ $\lvert\mathcal{A}\rvert = 20$
Erdős-Rényi	0.457	2.175	5.154	97.75	99.3	99.32
Barabási-Albert	0.471	2.15	5.186	98.37	99.34	99.4
Star	1.77	2.317	5.324	100.	100.	100.
Caveman	1.452	2.302	5.285	98.12	98.84	99.05
Caterpillar	1.039	2.674	5.474	98.37	93.34	97.05
Lobster	0.928	2.776	5.507	97.5	92.09	95.02
Tree	1.01	2.672	5.494	94.25	95.59	95.19
Grid	0.564	2.511	5.416	92.62	91.65	91.3
Ladder	0.605	2.536	5.487	91.25	92.15	91.
Line	1.0375	2.831	5.643	88.12	88.4	89.34
	$\lvert\mathcal{S}\rvert \approx 20$ $\lvert\mathcal{A}\rvert = 8$			$\lvert\mathcal{S}\rvert \approx 20$ $\lvert\mathcal{A}\rvert = 8$		
Maze (Tamar et al., 2016)	4.95			69.86		

Generalizes well (zero-shot) to unseen MDP graph structures
*worse perf for
- distinct graphs to the training set
- sparse graphs

2. Architectures related to Algorithmic Alignment

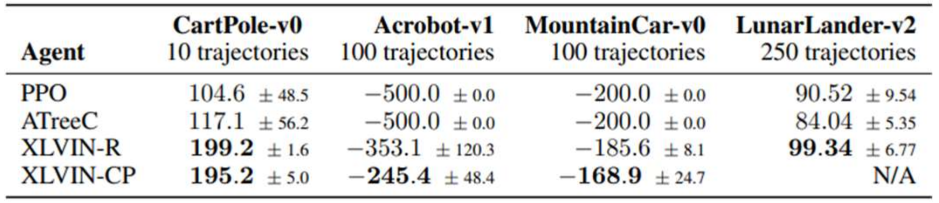
- XLVIN [1] imitated planning in the latent space via GNN, which outperformed baseline by latently reorganizing the environment

Method



- Encoder: consumes state representation and produces hidden state embeddings $h\_s=z(s)$ (CNN)
- Transition: consumes state embedding h_s and action to produce the next state embedding h_s' (pretrained with MLP, TransE)
- Executor: combines neighborhood h_s to produce an updated embedding $x\_s=X(h\_s\ ,\ N(h\_s\ ))$ (pretrained with GNN)
- Policy/Value Tail: consumes hidden and updated state embeddings and outputs action probabilities or state-values $A(h\_s\ ,x\_s)$

Results



Agent	CartPole-v0 10 trajectories	Acrobot-v1 100 trajectories	MountainCar-v0 100 trajectories	LunarLander-v2 250 trajectories
PPO	104.6 ± 48.5	−500.0 ± 0.0	−200.0 ± 0.0	90.52 ± 9.54
ATreeC	117.1 ± 56.2	−500.0 ± 0.0	−200.0 ± 0.0	84.04 ± 5.35
XLVIN-R	**199.2** ± 1.6	−353.1 ± 120.3	−185.6 ± 8.1	**99.34** ± 6.77
XLVIN-CP	**195.2** ± 5.0	**−245.4** ± 48.4	**−168.9** ± 24.7	N/A

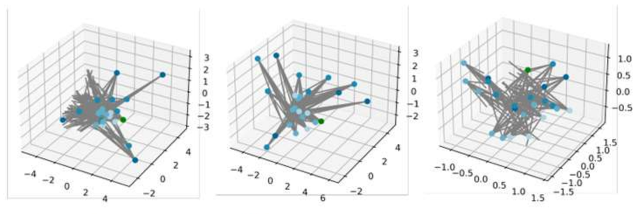


Figure 3: **Top:** A test maze (*left*) and the PCA projection of its TransE state embeddings (*right*), colour-coded by distance to goal (in green). **Bottom:** PCA projection of the XLVIN state embeddings after passing the first (*left*), second (*middle*), and ninth (*right*) level of the continual maze.

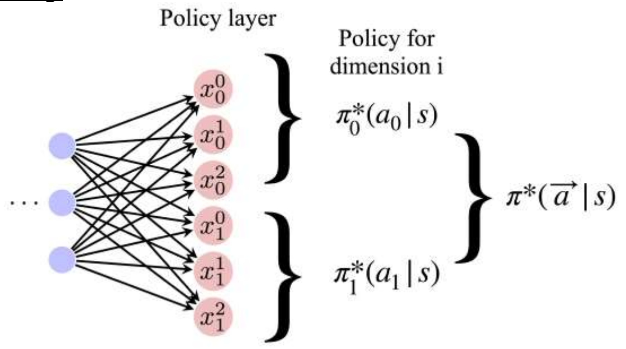
- XLVIN outperformed PPO and ATreeC in classic control problems
- Latent Analysis showed XLVIN reorganizes the grid from shortest path to the longest, in a path finding problem (qualitative evidence for planning imitation)

[1] Deac, A., Veličković, P., Milinković, O., Bacon, P., Tang, J., Nikolić, M. (2021). Neural Algorithmic Reasoners are Implicit Planners, NeurIPS

2. Architectures related to Algorithmic Alignment

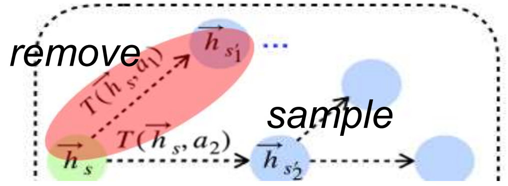
- CNAP[1] extended XLVIN to continuous action space. I have conducted rather detailed methods and results analysis because I played with it around for a while as part of my RL lecture coursework.

Method 1: Action Space Discretization by Factorized Joint Policy



- If action space is discretized without policy adjustment, it will lead to layer dimension (# number of bins)^(dimension of action space), i.e. exponential
- Factorized Joint Policy expresses original policy as a product of each action's policy, ensuring layer dimension to be (# number of bins) x (dimension of action space), i.e. polynomial

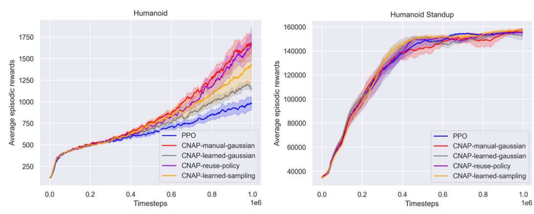
Method 2: Action Space Reduction by Neighbour Sampling (4 tactics)



	Description	Pros	Cons
Manual Gaussian	Choose action in [0, N-1]with Gaussian distribution with mean = N/2 and std=N/4	Sample efficient	Gaussian Distribution may not fit
Learned Gaussian (parametric)	Similar to Manual Gaussian, but use FC to parametrize mean and std	More flexible choice of distribution range	
Reuse Policy	Reuse policy layer in tail module to sample actions	Parameter reuse & Policy distribution alignment	Misalignment in input format
Learned Sampling (parametric)	Use FC to output N x D logits that fully sample actions	Dedicated distribution learning	More parameter s and training

[1] He, Y., Veličković, P., Liò, P., Deac, A. (2022). Continuous Neural Algorithmic Planners

2. Architectures related to Algorithmic Alignment



- High performance than PPO in both classic and complex ctns action space RL problems

Results 1: Classic

Model	**MountainCarContinuous-v0**
PPO Baseline	-4.96 ± 1.24
CNAP-B	55.73 ± 45.10
CNAP-R	**63.41** ± 37.89

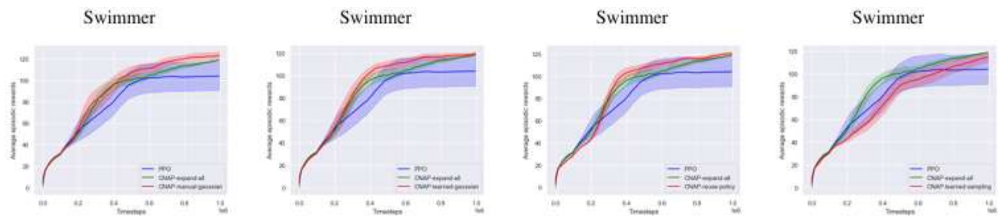
- Performance increases as CNAP-R > CNAP-B > PPO
  * CNAP-B executor pretrained on a binary graph (to simulate bi-directional control of mountain car)
  *CNAP-R executor pretrained on a random Erdos-Renyi graph

Model	Action Bins	**MountainCar-Continuous**
PPO	5	-2.16 ± 1.25
	10	-4.96 ± 1.24
	15	-3.95 ± 0.77
CNAP-B	5	29.46 ± 57.57
	10	55.73 ± 45.10
	15	22.79 ± 41.24
CNAP-R	5	20.32 ± 53.13
	10	**63.41** ± 37.89
	15	26.21 ± 46.44

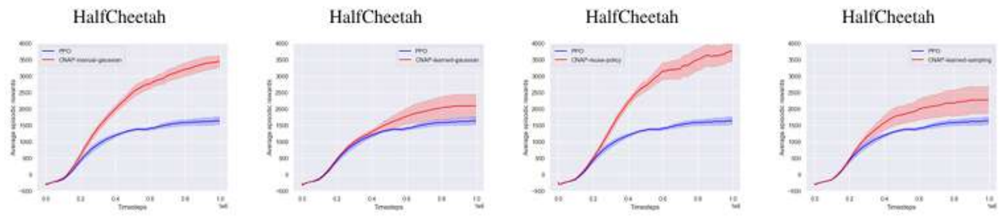
Model	GNN Steps	**MountainCar-Continuous**
CNAP-B	1	55.73 ± 45.10
	2	46.93 ± 44.13
	3	40.58 ± 48.20
CNAP-R	1	**63.41** ± 37.89
	2	34.49 ± 47.77
	3	43.61 ± 46.16

- graph width (# of action bins) of 10 resulted in highest mean rewards in general
  (balance between complexity and sample efficiency)
- graph depth (GNN steps) of 1 resulted in highest mean
  (sample efficiency, precision of translator)

Results 2: Mujoco



(a) Manual-Gaussian (b) Learned-Gaussian (c) Reuse-Policy (d) Learned-Sampling



(e) Manual-Gaussian (f) Learned-Gaussian (g) Reuse-Policy (h) Learned-Sampling

- CNAP outperformed PPO baseline with all 4 sampling tactics across Swimmer, HalfCheetah and Humanoid
- Non-parametric policies (e.g. Manual-Gaussian or Reuse-Policy) led to the largest performance improvement
- CNAP and PPO did not differ in performance in the Humanoid Standup environment; CNAP showed more robust actions