DeepMind

A Tale of Three Implicit Planners and the XLVIN Agent

Petar Veličković

DeepMind/ELLIS CSML Seminar Series
University College London
12 March 2021

DeepMind

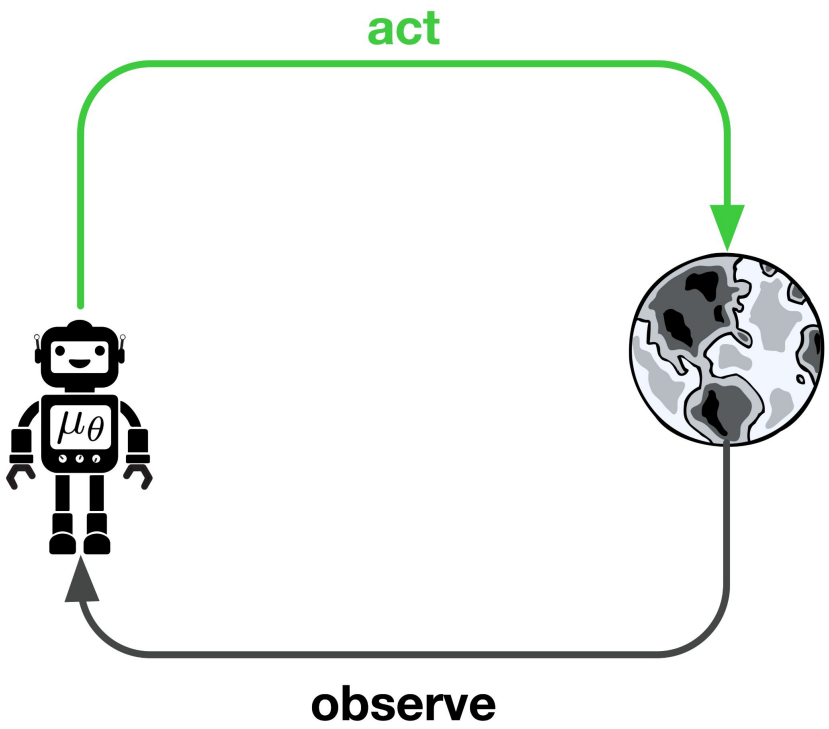
In this talk:

Architectures and algorithms for **implicit planning**

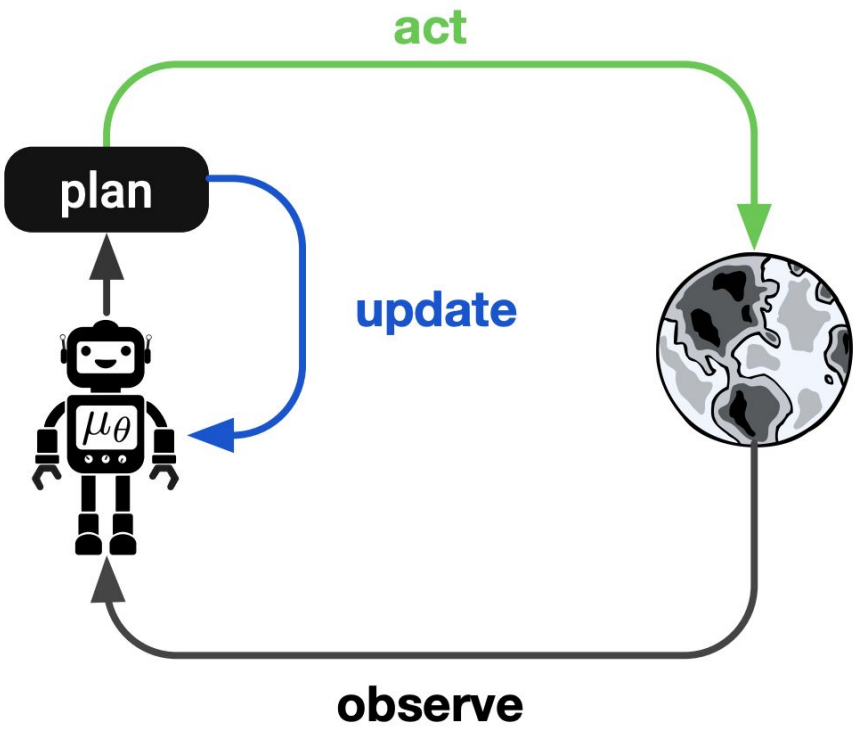
DeepMind

1 What are we doing here?

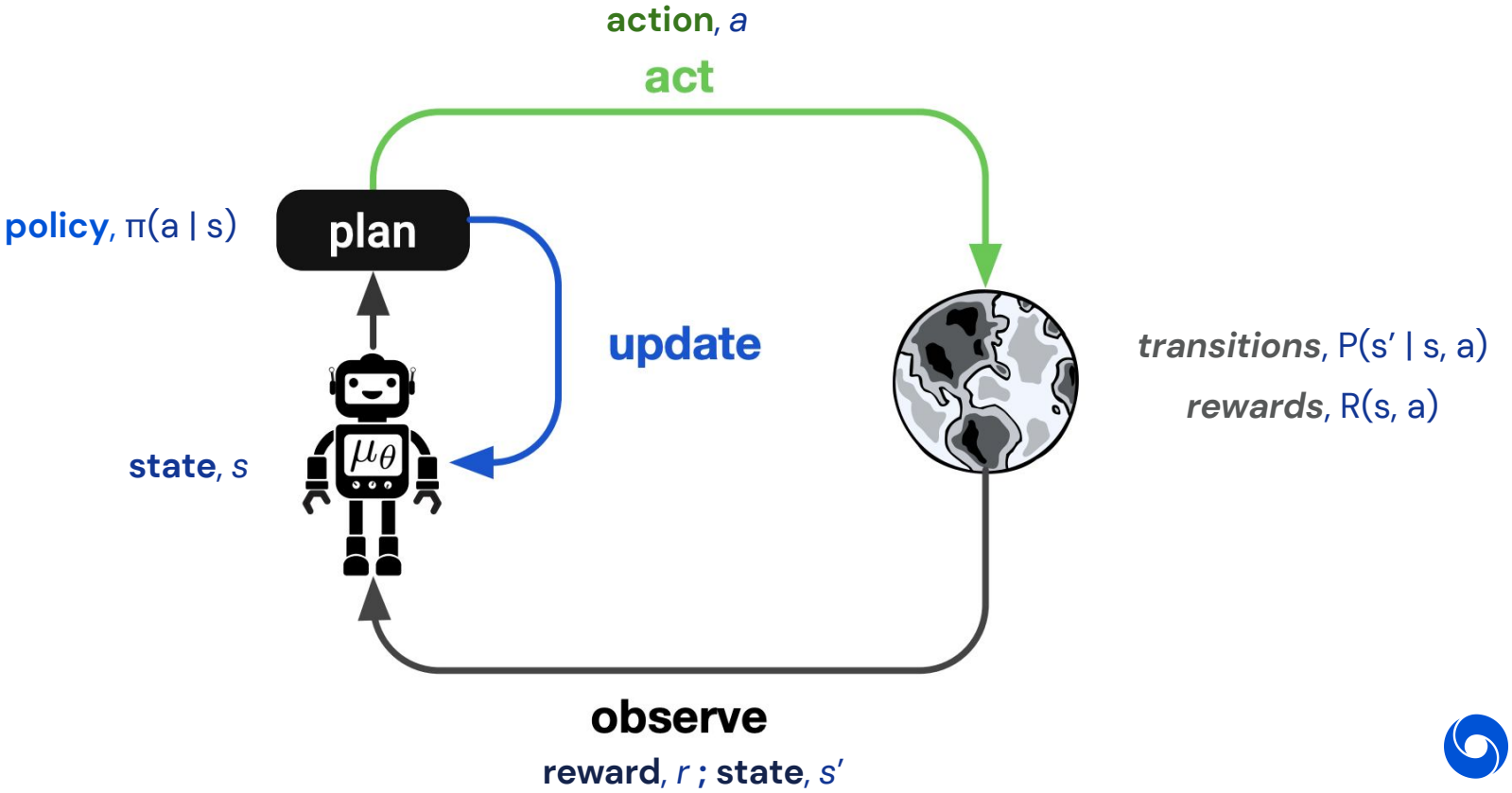
Reinforcement learning (RL) setting



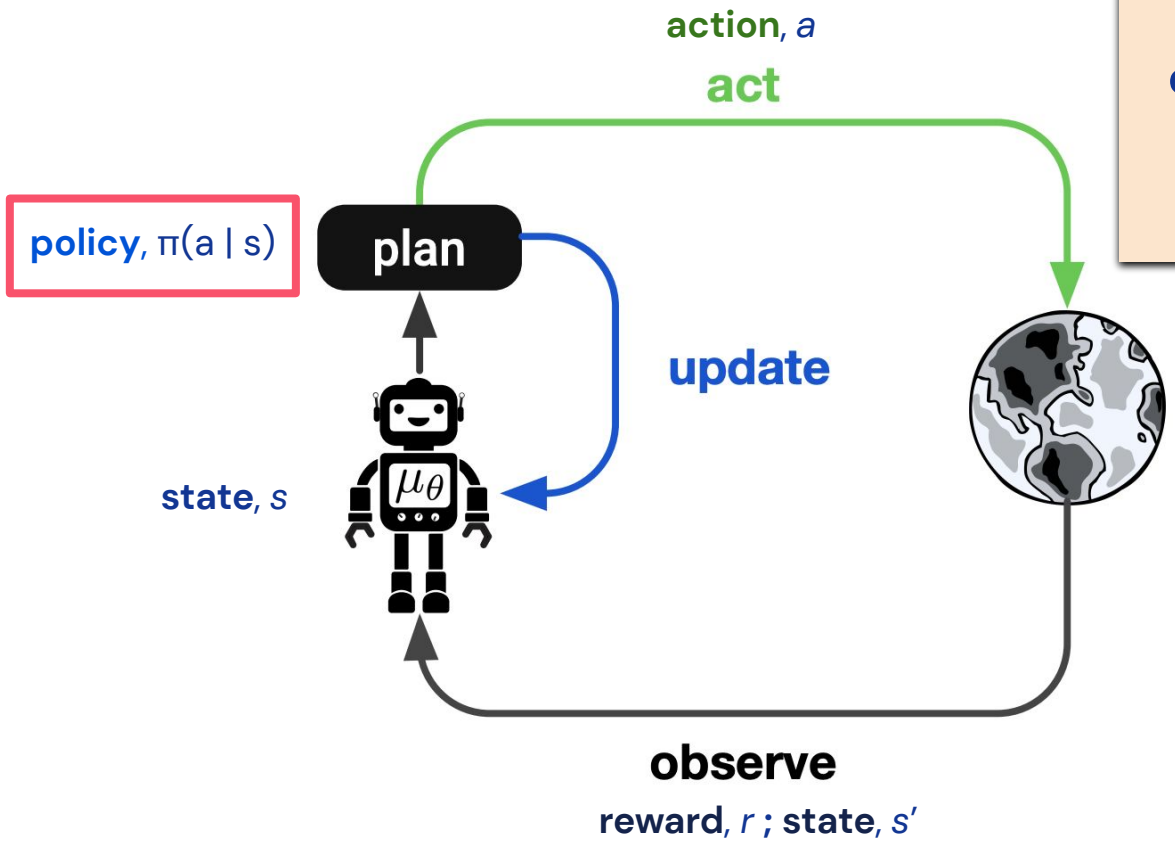
Reinforcement learning (RL) setting (with *planning*)



Reinforcement learning (RL) setting (variables)



Reinforcement learning (RL) setting



Want to optimise:

Discounted cumulative reward

$$G = \sum_{t \geq 0} \gamma^t r_t$$

DeepMind

2 Do you have a plan?

What's in a policy?

- Policies acting purely through adapting to observed rewards are often called **reactive**
- In many cases, they require large quantities of **data** and are slow to **adapt**
- **Planning** ameliorates such issues by maintaining an explicit **model** of the world
  - *State transition model*: $s' \sim f_T(s, a)$
  - *Reward model*: $r \sim f_R(s, a)$
  - Typically trained from **observed trajectories**
- Using these models, a planner can **simulate** the effects of actions before taking them!
  - Comes with many benefits if done properly…

What are the benefits?

- Gains in **data efficiency**: Good model implies fewer interactions are needed to learn to act
- Strong models allow quickly **adapting** to previously unexplored situations
- Being mindful of the consequences of acting enables better **safety**
- Allowing to explicitly account for external factors (e.g. **human** interactions)
- Impactful for **game-playing** (AlphaGo) and across the **sciences** (Segler *et al.*, Nature'18)
- Encouraging theoretically: **perfect** models allow for planning **perfect** policies!

Intro to value iteration

- Value Iteration: *dynamic programming* algorithm for **perfectly** solving an RL environment

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

  where v(*s*) corresponds to the **value** of state *s*.

- **Guaranteed** to converge to *optimal* solution (fixed-point of Bellman optimality equation)!

$$V^\star(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\star(s') \right)$$

  Optimal policy takes actions that **maximise** *expected value*: $\text{argmax}_a \sum_{s'} V^*(s') P(s' \mid s, a)$

- BUT requires **full knowledge** of underlying MDP (P / R)

DeepMind

3 How do you even plan?

Learning the model

- In many cases of interest, P and R will be hidden from us!
  - Hence, we need to learn them through *interacting* with the environment

- Assume access to an **interaction dataset** of trajectories {(*s*, *a*, *r*, *s'*)}
  - *"I performed **action** a in **state** s, observed **reward** r and transitioned to **state** s'."*
  - This can be built up as we learn to act---but need to tradeoff exploring vs exploiting.

- We can then learn the transition function f_T by supervising.
  - Make $f_T(s, a)$ be predictive of s'.
  - Proceed similarly for learning $f_R(s, a)$.

- There are other ways to incorporate assumptions about the env dynamics into f_T/f_R
  - Here we assume a "tabula rasa" approach.

Latent-space transition models

- Especially, we will find models quite attractive if they operate in a **latent** state-space.
  - This particularly plays nicely with **neural network** approximators

- Assume we have encoded our state (e.g. with a NN) into **embeddings**, $z(s) \in \mathbb{R}^k$
  - Our transition model is then of the form $T : \mathbb{R}^k \times A \rightarrow \mathbb{R}^k$
  - Optimised such that $T(z(s), a) \approx z(s')$

- Many popular methods exist for learning T in the context of *self-supervised learning*

- **Contrastive** learning methods try to discriminate (s, a, s') from *negative* pairs $(s, a, \tilde{s})$

Implicit planning

- Planning methods vary in how they exploit this information

- Policies can plan by **explicitly** rolling out the model, and then deciding how to act based on the outcomes of the rollouts (leading to *model-based RL*)
  - This is a **discrete** process, and doesn't always play trivially with neural network-based optimisation

- Here we will focus on learning to plan **implicitly** (leading to *implicit planning*).
  - Support planning computations using (architectural) *inductive biases*
  - But still optimise using **model-free** RL losses e.g. DQN/A3C (~ *"model-free planning"*)

- Easy to compose and blend existing model-free approaches – *"best of both worlds"*

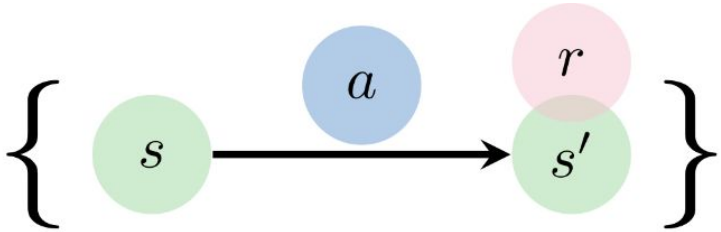
- Major challenge: need to build **differentiable** planning components

DeepMind

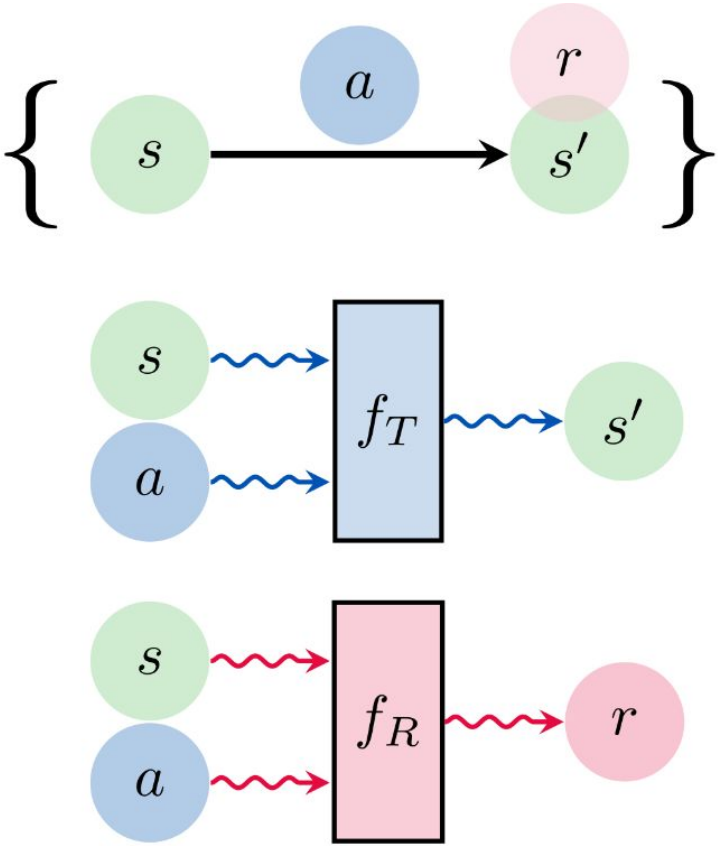
4

What do you mean, the plan must be *differentiable*?

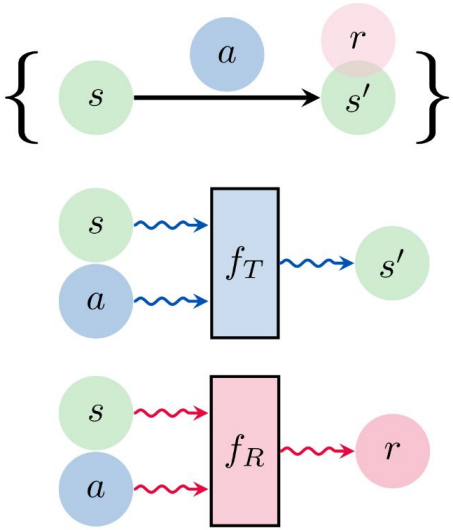
What we have covered so far

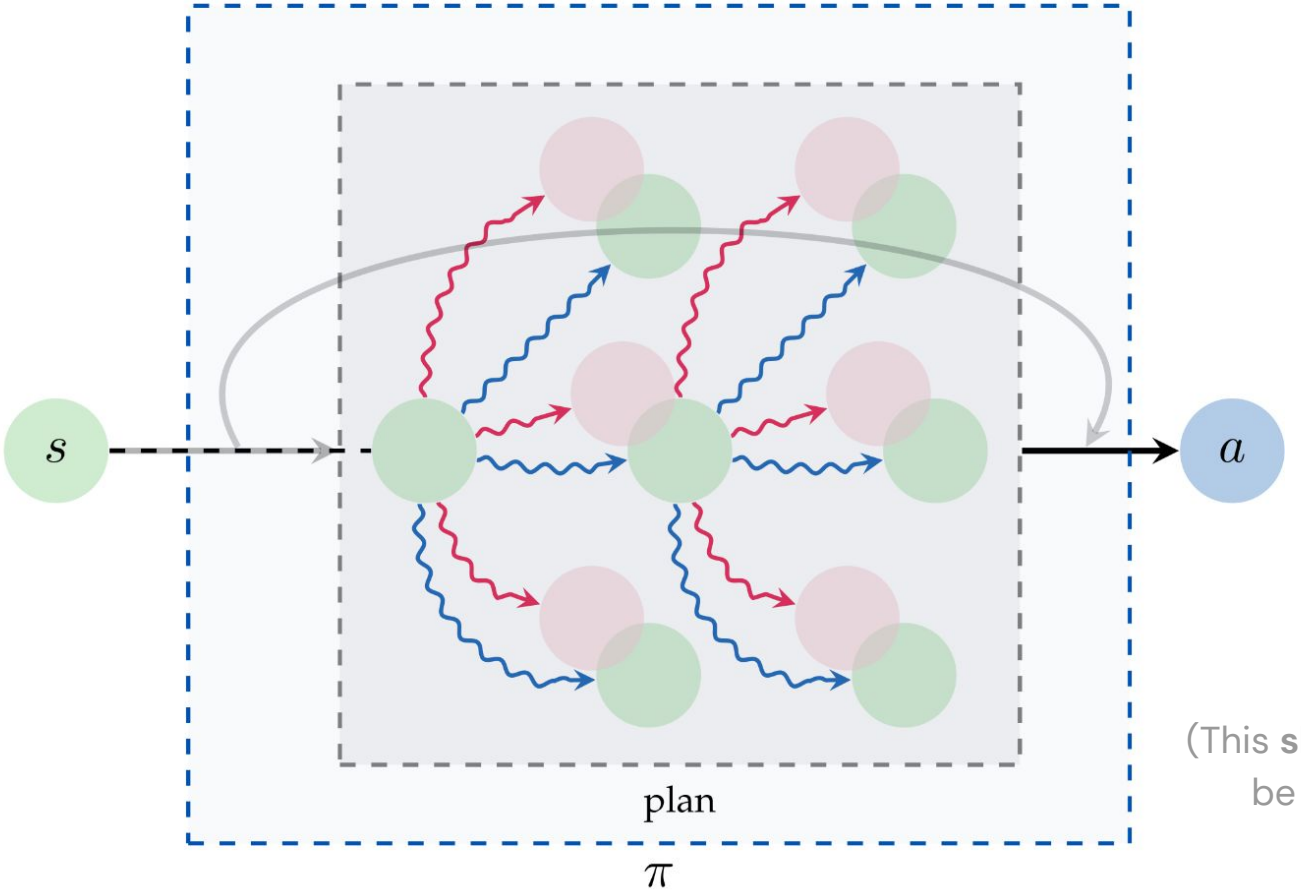


What we have covered so far



What we have covered so far





(This **skip connection** can be quite powerful!)

DeepMind

5

Is it okay to start with something simpler?

Value iteration, revisited

- Now we will focus explicitly on the Value Iteration update rule

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

- At each step, update each state's values, by considering immediate successors

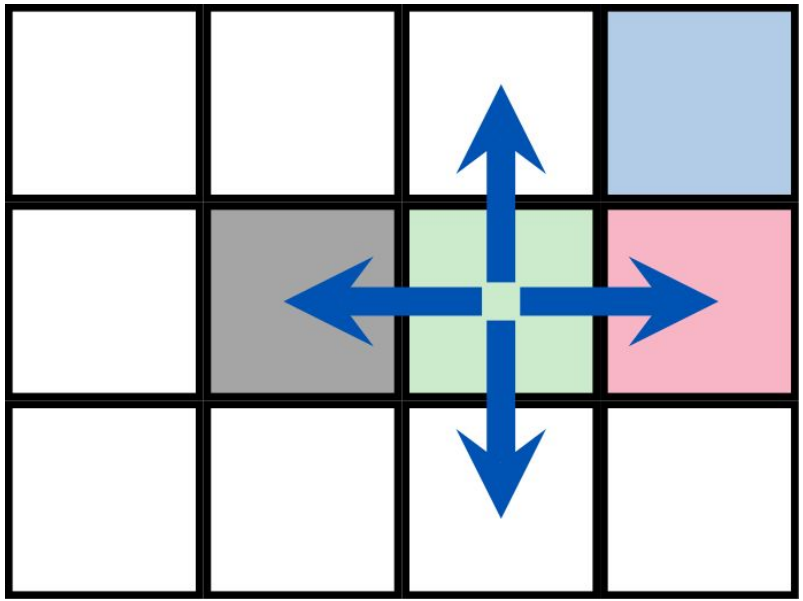
- Such a function is actually **already** differentiable w.r.t. its inputs!
  - But determining successors and aggregating over them may be tricky

- It makes sense to start with special RL environments where this determination is easier...

Value iteration in grid worlds

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

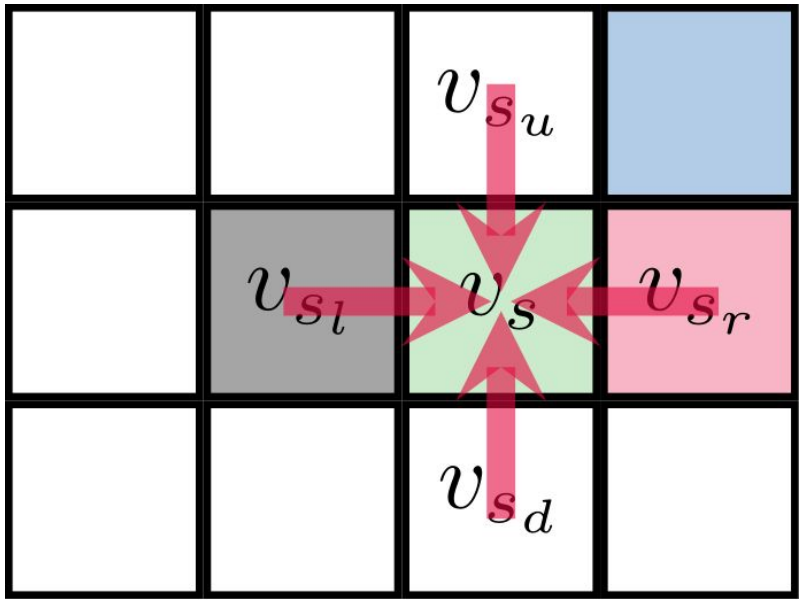
- Each state has **known** neighbours
- Actions are **deterministic**
- In this setting, VI amounts to...



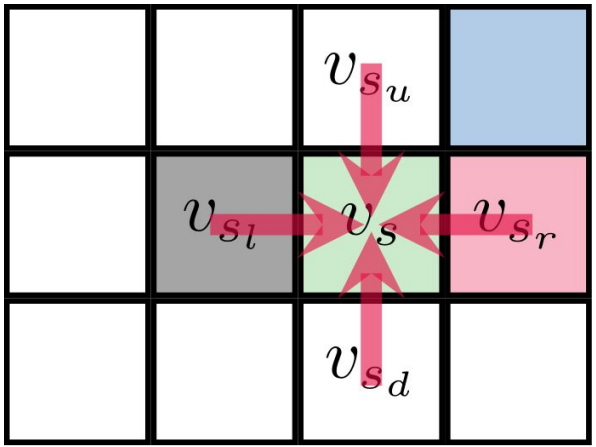
Value iteration in grid worlds

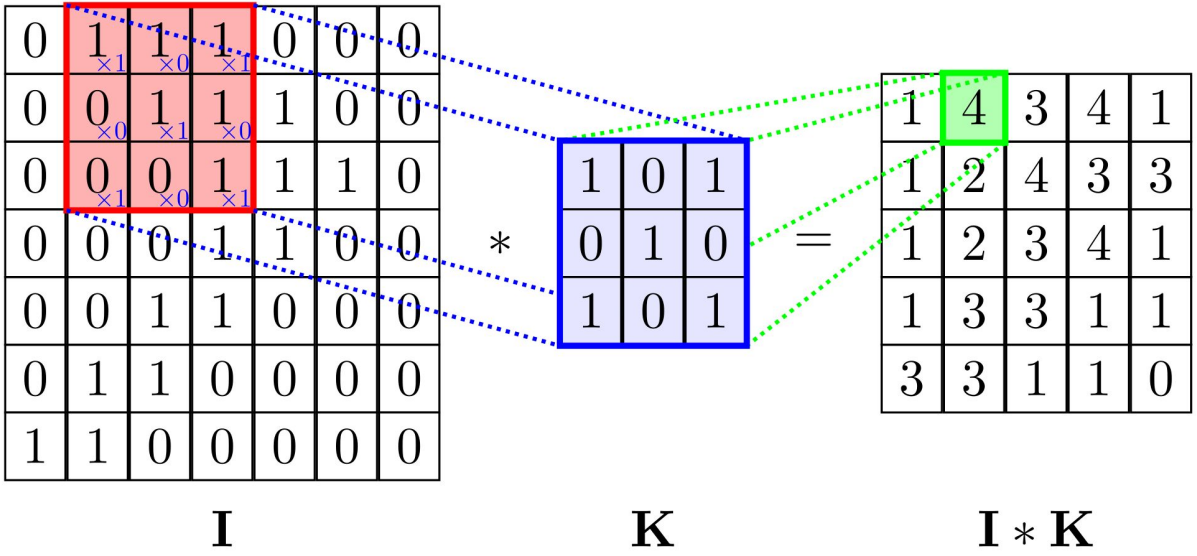
$$v^{(t+1)}(s) = \max_{a \in \mathcal{A}_s} r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^{(t)}(s')$$

- Each state has **known** neighbours
- Actions are **deterministic**
- In this setting, VI amounts to...
- Computing **sums** of **neighbouring** values!
- Does this remind you of something?

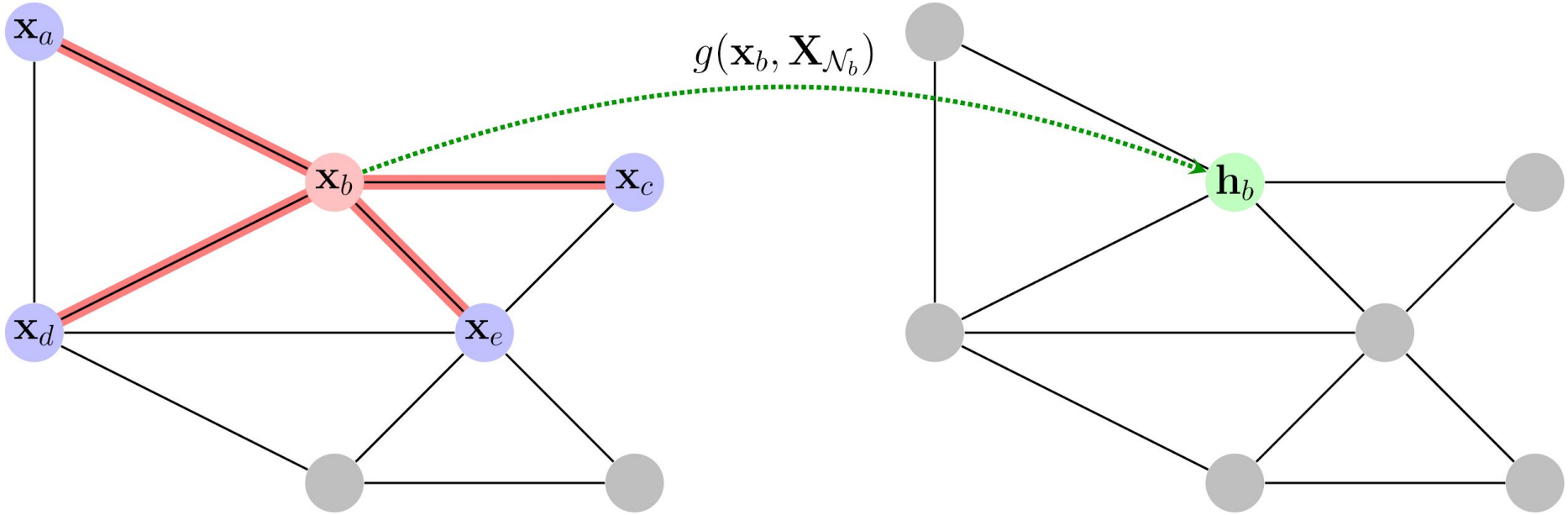


Grid world VI ~ Convolution!



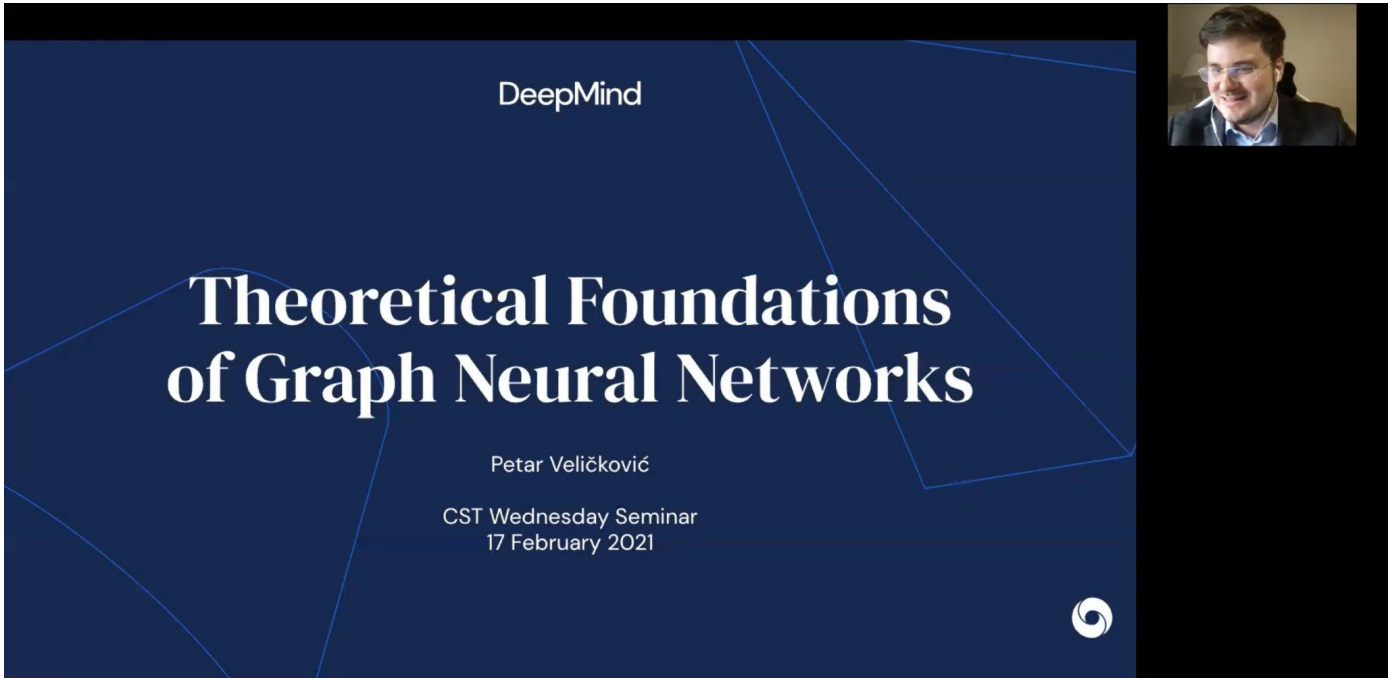


Generic MDP VI ~ *Graph* convolution!



If you'd like to know more

If GNNs are new(ish) to you, I recently gave a useful talk on **theoretical GNN foundations**:
https://www.youtube.com/watch?v=uF53xsT7mjc



Value Iteration Network

- Exactly this idea is leveraged by *Value Iteration Networks* (Tamar *et al.*, NeurIPS'16)
- Assuming the underlying MDP is **discrete**, **fixed** and **known**...
- We can perform VI-style computation by stacking a **shared** convolutional layer
  $\Rightarrow$ We have our differentiable planning module!
- Original VIN paper mainly dealt with grid worlds and hence used CNNs
  - Extended to generic MDPs and GNNs by *GVINs* (Niu *et al.*, AAAI'18)

DeepMind

6

Relax these constraints by tomorrow!

Moving beyond **known** worlds

- Assuming the MDP is **fixed** and **known** was quite helpful
  - We never needed to estimate *transition models*
  - Didn't have to deal with *continuous* state spaces

- To relax these constraints, we learn a transition function T
  - To operate directly over state embeddings z(s)
  - Using it, we may construct a *"local MDP"* around a current state

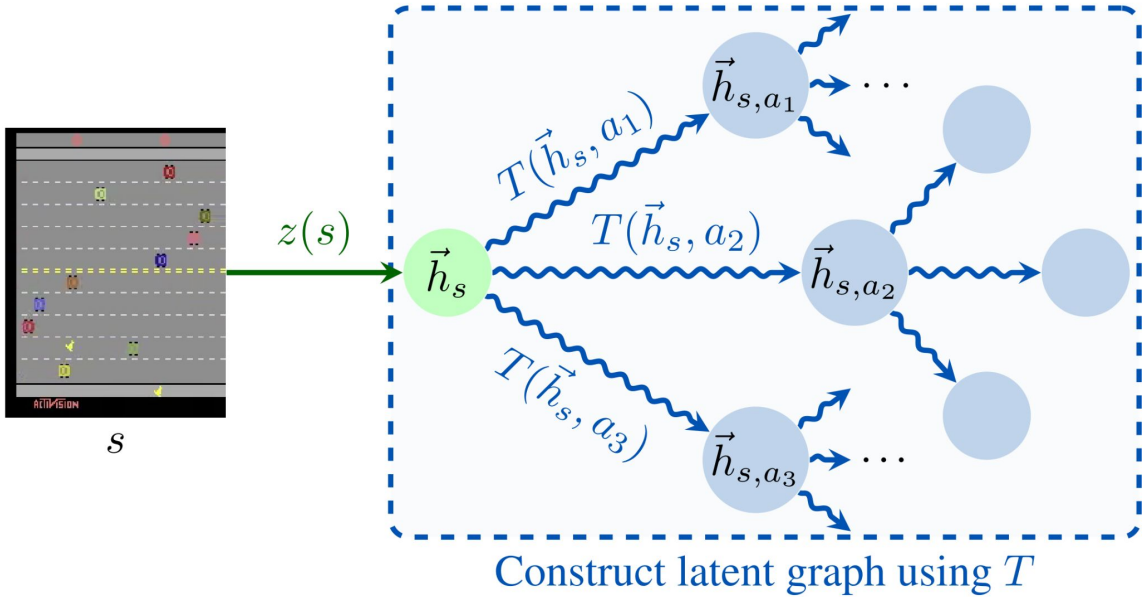
- We already discussed how to train T in prior sections

Using a transition model to **expand**

We can use the transition model on **every** action, to be exhaustive (*~breadth-first search*)

Doesn't **scale** with large action spaces / thinking times; $O(|A|^K)$

Can find more interesting *rollout policies*, e.g. by **distilling** well-performing **model-free** ones.



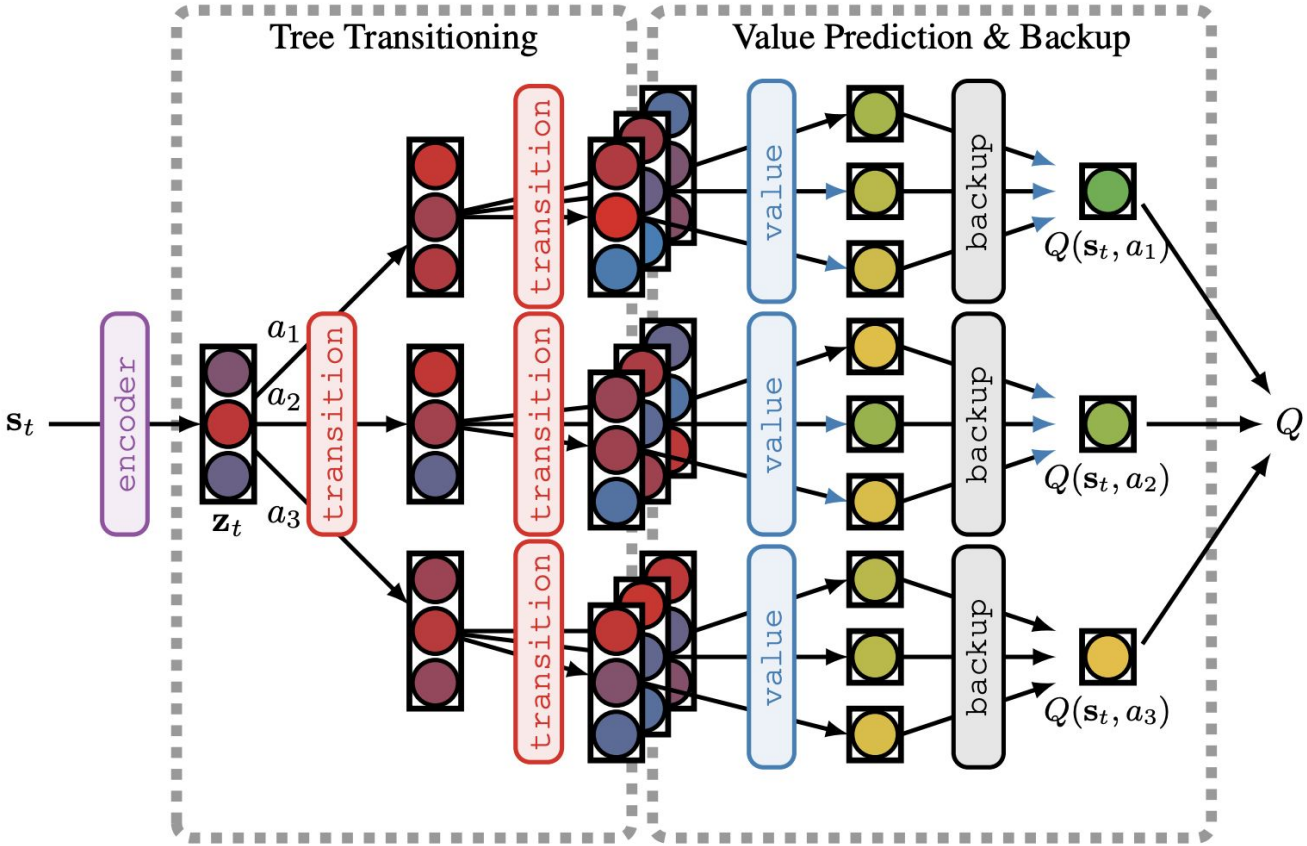
TreeQN / ATreeC

- Assume that we have reward/value models, giving us scalar **values** in every expanded node
- We can now **directly** apply a VI-style update rule!

$$Q(\mathbf{z}_{l|t}, a_i) = r(\mathbf{z}_{l|t}, a_i) + \begin{cases} \gamma V(\mathbf{z}^{a_i}_{d|t}) & l = d-1 \\ \gamma \max_{a_j} Q(\mathbf{z}^{a_i}_{l+1|t}, a_j) & l < d-1 \end{cases}$$

- Can then use the computed Q-values **directly** to decide the policy
- Exactly as leveraged by models like TreeQN / ATreeC (Farquhar *et al.*, ICLR'18)
  - Also related: Value Prediction Networks (Oh *et al.*, NeurIPS'17)

TreeQN / ATreeC in action



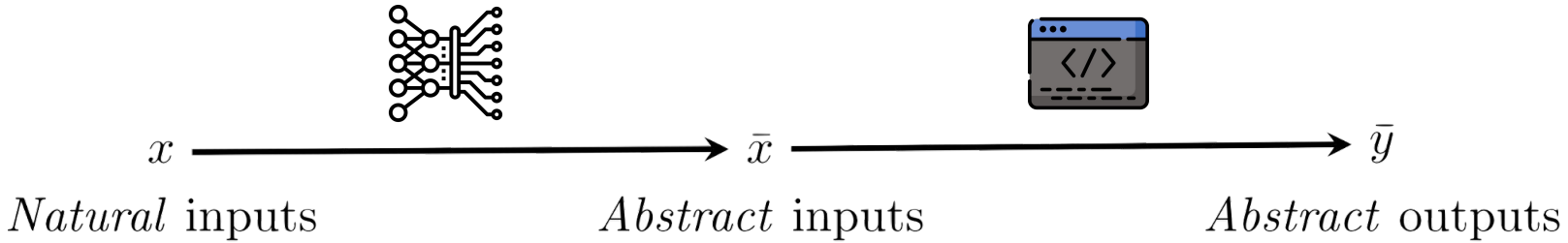
DeepMind

7

Uh oh, there's *bottlenecks* in our plan

High-level view

- It's good to take a recap and realise what we have done so far
  - We mapped our **natural** inputs (e.g. pixels) to the space of abstract inputs
  - (local MDP + reward values in every node)
  - This allowed us to execute VI-style algorithms **directly** on the abstract inputs



- The VI update is differentiable, and hence so is our entire implicit planner.

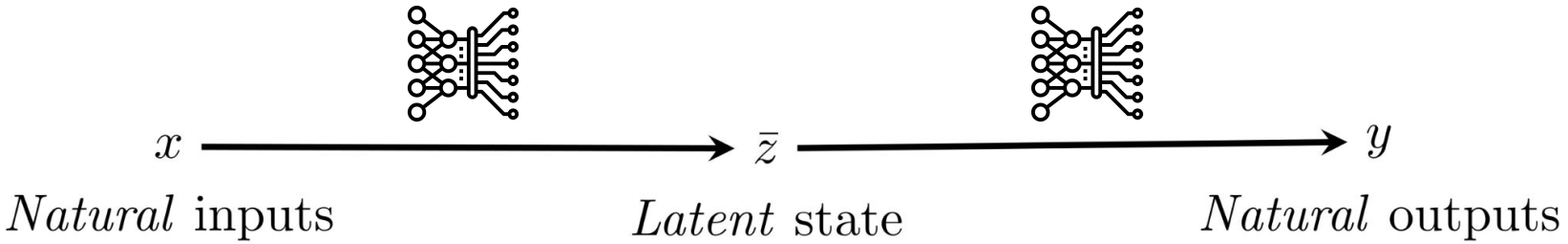
Algorithmic ***bottleneck***

- Real-world data is often incredibly *rich*
- We still have to compress it down to **scalar values**
- The VI algorithmic solver:
  - **Commits** to using this scalar
  - Assumes it is **perfect**!
- If there are insufficient training data to properly estimate the scalars…
- We hit ***data efficiency*** issues again!
  - Algorithm will give a **perfect** solution, but in a ***suboptimal*** environment

Breaking the bottleneck

- Neural networks derive great flexibility from their **latent** representations
  - They are inherently *high-dimensional*
  - If any component is poorly predicted, others can step in and compensate!

- To *break the bottleneck*, we replace the VI update with a **neural network**!



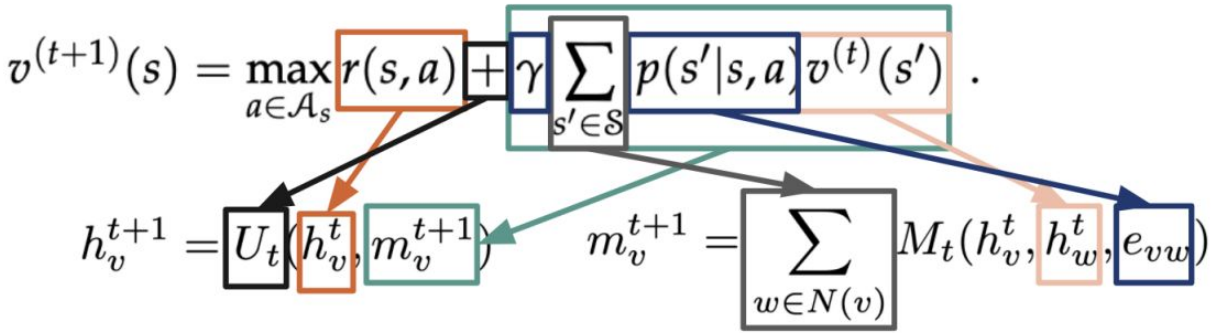
- As before, we can use **graph neural networks** to perform VI-aligning computations.

Algorithmic reasoning

- GNN over state representations *aligns* with VI, but may put **pressure** on the planner
  - Same gradients used to *construct* correct graphs **and** make VI computations

- To alleviate this issue, we choose to **pre-train** the GNN to perform value iteration-style computations (over many **synthetic** MDPs), then deploying it within our planner

- This exploits the concept of *algorithmic alignment* (Xu *et al.*, ICLR'20)



Algorithmic reasoning

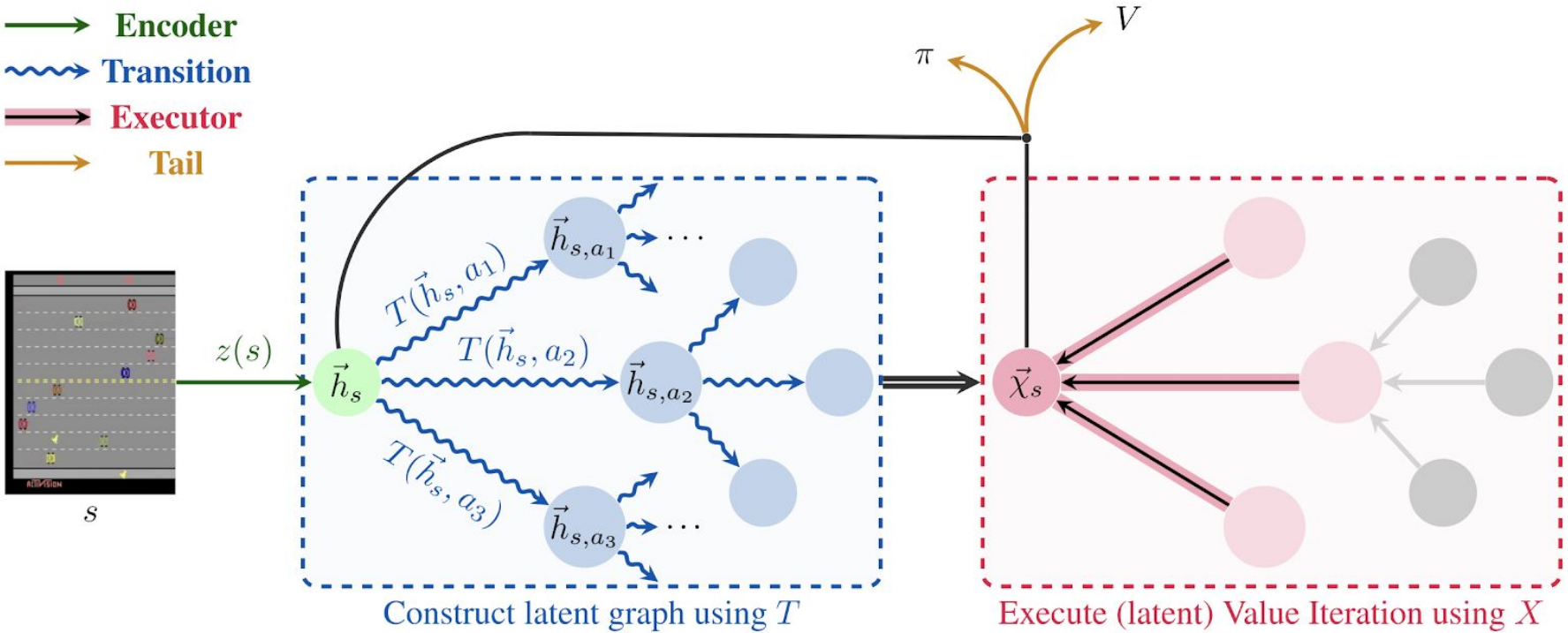
- GNN over state representations *aligns* with VI, but may put **pressure** on the planner
  - Same gradients used to *construct* correct graphs **and** make VI computations

- To alleviate this issue, we choose to **pre-train** the GNN to perform value iteration-style computations (over many **synthetic** MDPs), then deploying it within our planner

- This exploits the concept of *algorithmic alignment* (Xu *et al.*, ICLR'20)

- Relying on prescriptions on how to build effective extrapolating GNN reasoners, provided by *"Neural Execution of Graph Algorithms"* (Veličković *et al.*, ICLR'20)

Putting it all together!



XLVIN (Deac *et al.*, NeurIPS'20 DeepRL)

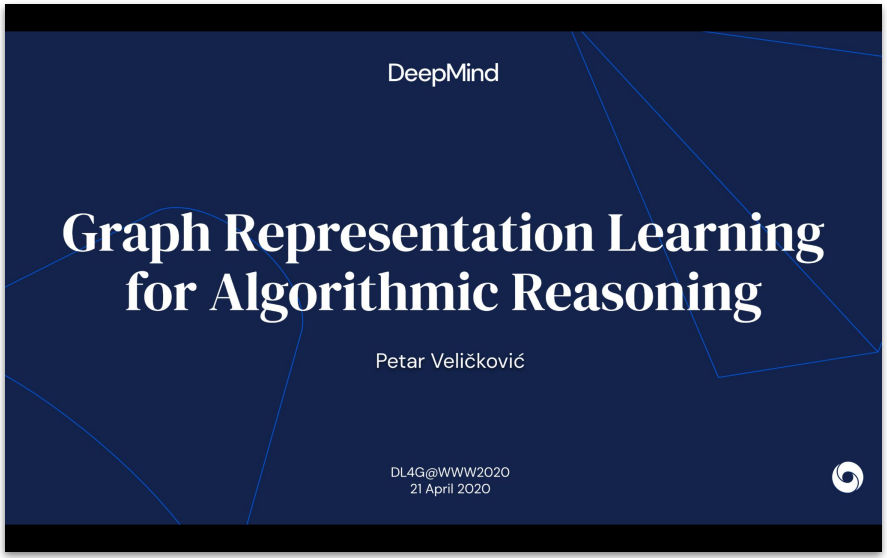
XLVIN Components

- **Encoder** (**z**: $S \to \mathbb{R}^k$) provides state representations
- **Transition** (**T**: $\mathbb{R}^k \times A \to \mathbb{R}^k$) simulates effects of actions in *latent* space
  - **Pre-trained & Fine-tuned** on the TransE loss (observed trajectories)
- **Executor** (**X**: $\mathbb{R}^k \times \mathbb{R}^{|A| \times k} \to \mathbb{R}^k$) simulates a planning algorithm (Value Iteration) in *latent* space
  - **Pre-trained** to execute VI on synthetic MDPs of interest, then **frozen**
- **Policy / Value Head**, computing action probabilities and state-values given embeddings
  - Use **PPO** as the policy gradient method

The entire procedure is end-to-end differentiable, does not impose any assumptions on the structure of the underlying MDP, and has the capacity to perform computations directly aligned with value iteration. Hence our model can be considered as a generalisation of VIN-like methods to settings where the MDP is not provided or otherwise difficult to obtain.

Further insight

If you would like to know more details about constructing good GNN executors:



https://www.youtube.com/watch?v=IPQ6CPoluok

https://drive.google.com/file/d/1_EQ9Yu7VEkvrHaVHl_WbT5ABvxrSNY-s/view?usp=sharing

Algorithmic reasoning *survey*

Combinatorial optimization and reasoning with graph neural networks

Quentin Cappart[1], Didier Chételat[2], Elias Khalil[3], Andrea Lodi[2], Christopher Morris[2], and Petar Veličković[*4]

[1]Department of Computer Engineering and Software Engineering, Polytechnique Montréal
[2]CERC in Data Science for Real-Time Decision-Making, Polytechnique Montréal
[3]Department of Mechanical & Industrial Engineering, University of Toronto
[4]DeepMind

Combinatorial optimization is a well-established area in operations research and computer science. Until recently, its methods have focused on solving problem instances in isolation, ignoring the fact that they often stem from related data distributions in practice. However, recent years have seen a surge of interest in using machine learning, especially graph neural networks (GNNs), as a key building block for combinatorial tasks, either as solvers or as helper functions. GNNs are an inductive bias that effectively encodes combinatorial and relational input due to their permutation-invariance and sparsity awareness. This paper presents a conceptual review of recent key advancements in this emerging field, aiming at both the optimization and machine learning researcher.

Our *43-page* survey on GNNs for CO!

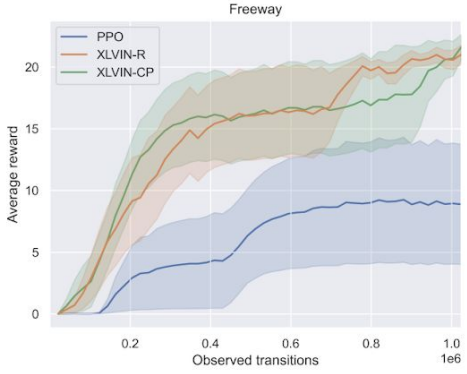
https://arxiv.org/abs/2102.09544

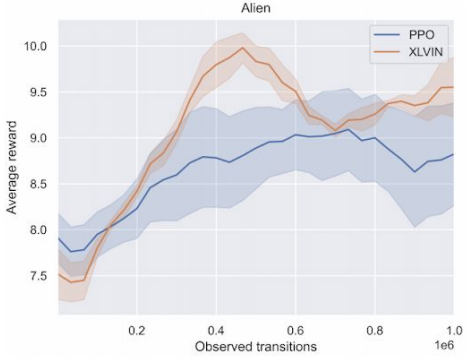
Section 3.3. details algorithmic reasoning, with comprehensive references.

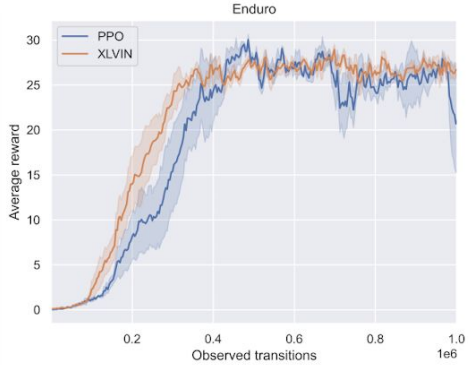
DeepMind

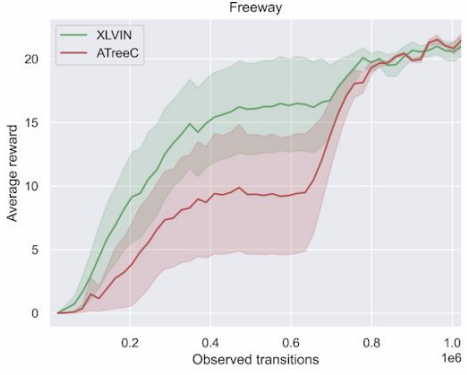
8 So... did our plan work?

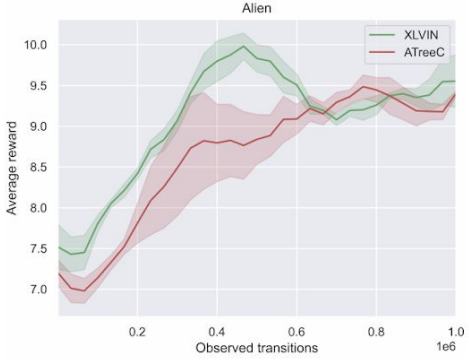
Results on *low-data* Atari

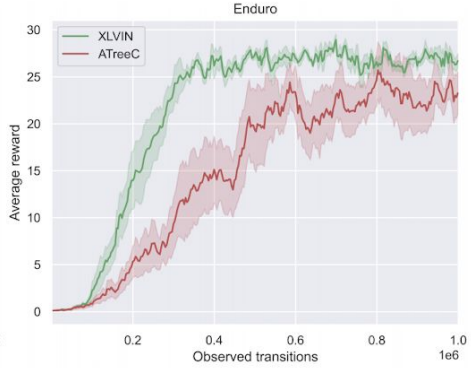












DeepMind

Thank you!

Questions?

petarv@google.com | https://petar-v.com

With many thanks to Andreea Deac, Ognjen Milinković, Pierre-Luc Bacon,
Jian Tang, Mladen Nikolić, Jess Hamrick, Feryal Behbahani and Charles Blundell