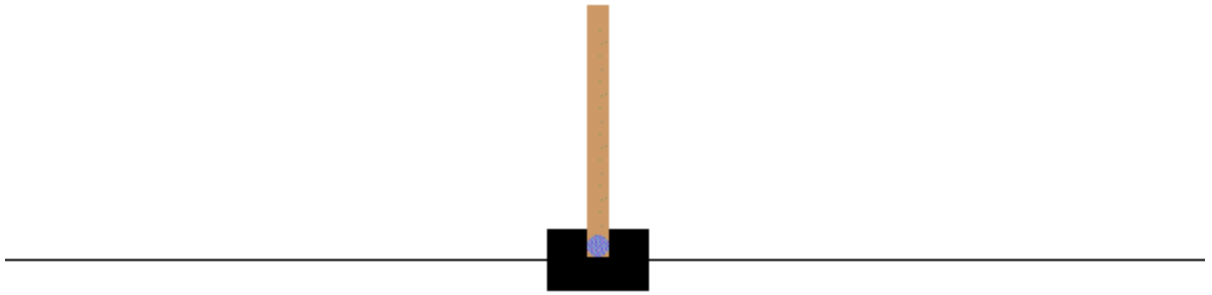
GNNs for Reinforcement Learning

Deep Reinforcement Learning

- Deep Reinforcement Learning is a field of machine learning in which we train a neural network model, called a policy, that is trained to assign higher probability to actions in states that maximise long term expected reward.
- For simple environments, we can use a simple neural network like a multi-layer perceptron (MLP) + a softmax function after the final layer which takes in a vector representation of the environment's current state and returns a vector of probabilities for each of the possible actions in that state.

A Familiar Example



State

$$\begin{bmatrix} Cart\ Position \\ Cart\ Velocity \\ Pole\ Angle \\ Pole\ Angular\ Velocity \end{bmatrix}$$

Model

MLP

Actions

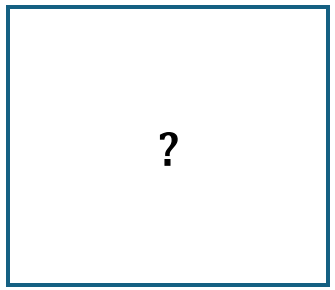
$$\begin{bmatrix} Probability\ of\ Pushing\ Cart\ Left \\ Probability\ of\ Pushing\ Cart\ Right \end{bmatrix}$$

A New Example

State

$$\begin{bmatrix} Enemy\ Unit\ 1 \\ Enemy\ Unit\ 2 \\ Enemy\ Unit\ 3 \\ \vdots \end{bmatrix}$$

Model

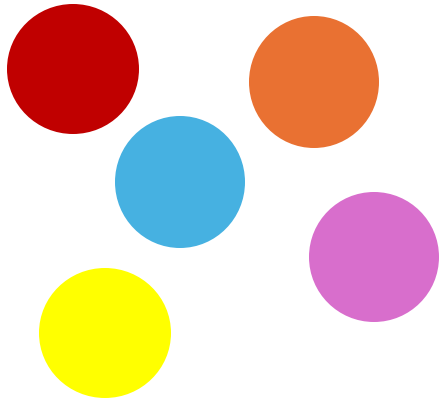


Actions

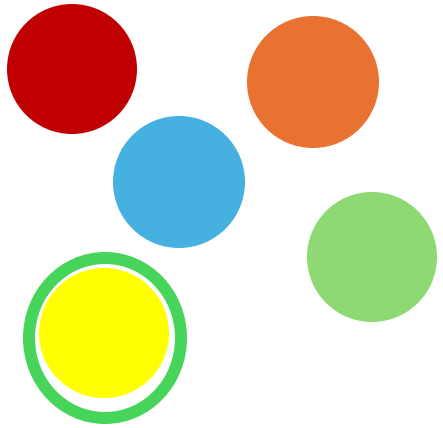
$$\begin{bmatrix} Attack\ Enemy\ Unit\ with\ Friendly\ 1 \\ \vdots \end{bmatrix}$$

The General Idea

State



Actions



Masking

- If it is possible, then we can create a policy which outputs the probability of every conceivable action in a state and then set the probability of forbidden actions to zero.
- For example, in chess there are **4164** ways of moving any piece on the board if they were unimpeded. In most positions, one or more pieces will have some moves unavailable to them due to other pieces being in the way
- The policy outputs a length **4164** vector of log-probabilities, the illegal actions are set to **-1e10** and the softmax function is applied.

Shortcomings

- Masking is a very efficient way of creating a policy which can be applied to environments in which the number of legal actions varies with time.

However,

- To apply masking, we require a priori knowledge of the largest possible number of actions we can take in a state in the environment.

Deep Sets, GNNs and Transformers

- These are three types of neural network that can operate on sets rather than fixed dimensional inputs (Recurrent Neural Networks also have this property but have an implicit sequential bias)
- The common element between these three network architectures is that they all contain some way of aggregating over elements in the input set to obtain a representation of the set as a whole.
- They differ in the manner in which relations between elements are considered.

Deep Sets

- Deep sets lie at one end of the spectrum, they do not consider the relationships between elements of the set before the aggregation is applied.

$$f(\{x_i\}) = \rho\left(\bigoplus_i (\phi(\boldsymbol{x}_i)\right)$$

- If we're using this as a value function, a function that assigns a scalar value to each state representing the expected future reward, then the function can be expected to capture some relationship between elements of the state.
- If we derive a policy function from this architecture, then we want to forgo the aggregation.

Transformers

- At the other end of the spectrum, we have transformers in which the attention mechanism captures pair-wise interactions between elements in the input set.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

- We can remove the positional encoding layer in the transformer to make it permutation invariant.

GNNs

- The original transformer architecture is essentially a Graph Attention Network Applied to a fully connected graph between elements in a set.
- If we use a GNN based function in reinforcement learning, we can impose a graph structure on our problem, which can be useful for making the model more computationally lightweight and more likely to learn relationships between elements that matter.
- This graph structure represents our implicit knowledge about which relationships we expect to have higher importance in a given problem.

Janossy Pooling

- Janossy Pooling handles the construction of permutation invariant functions over sets of arbitrary size by averaging over all possible permutations of the input set.

$$f(X) = \frac{1}{|S_n|} \sum_{\pi \in S_n} \phi\left(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\right)$$

- This will typically be computationally intractable and approximations will be used instead

Summary

- Here are 3 (4) examples of permutation invariant functions that can be applied to sets of arbitrary size.
- They can be generalized as GNNs applied to a spectrum of “graph” structures of the state (an unconnected graph for deep sets, a fully connected graph in the case of transformers and then any flavour of GNN sitting in the intermediate spaces)
- So, consider the application of GNNs to environments in reinforcement learning in which we have an unordered set of elements.
- I haven’t mentioned this yet but this kind of model enables generalization of the policy to problem instances larger than those seen in training

Routing Problems

- These type of models can be naturally applied to routing problems.
- If we take input data in the form of customer locations, then we can construct a graph on this data using a Euclidean K nearest neighbour algorithm
- Policies trained in this manner can be applied to problem instances larger than those present in the training data and show good generalization ability
- Of particular interest is how we can apply a model like this to dynamic routing problems.

Conclusions

- The alphastar work mentioned earlier is an example of using an attention mechanism to learn relationships between elements within a reinforcement learning policy.
- For reinforcement learning to be useful in environments that are not subject to controls, having a policy like these can adapt more easily to the presence of additional elements in a state seems important.

References

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... & Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *nature*, *575*(7782), 350-354.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). Deep sets. Advances in neural information processing systems, 30.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2020). Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637.

Murphy, R. L., Srinivasan, B., Rao, V., & Ribeiro, B. (2018). Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. arXiv preprint arXiv:1811.01900.

Kool, W., Van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems!. arXiv preprint arXiv:1803.08475.

Joshi, C. K., Cappart, Q., Rousseau, L. M., & Laurent, T. (2022). Learning the travelling salesperson problem requires rethinking generalization. Constraints, 27(1), 70-98.