Reinforcement Learning: Function Approximation

Earl Wong

Classical

- Many classical problems in RL originated from discrete state and action spaces.
- For example: NxN grid world consisted of N*N discrete states and at most, 8 discrete actions (up, down, left, right and diagonals).
- Other problems with continuous states and / or actions, could (also) be cast into discrete state and / or action spaces.
- For example, continuous valued cart pole states (car position, car velocity, pole angle, pole velocity) could be discretized through binning.

Classical

- Adding reward R to the state / action picture, we have the (S, A, R) components of a Markov Decision Process (MDP).
- Adding transition probabilities P to the picture, Bellman equations, state and action value functions, and policies quickly follow.
- Hence, a wide variety of RL problems can now be addressed.
- The underlying limitation though, was the curse of dimensionality.
- For example, how much could grid world be scaled up (from 4x4 to NxN) before becoming impractical?

Classical

- Some relief could be obtained, by solving / moving the problem to a model free environment.
- Now, rollouts could be used to determine state and action value functions, and optimal policies.
- However, model free RL was not the “end all”, since the “tabular” world constraints remained.
- Question: How do we move out of “tabular” world?

Math to the Rescue

- Elementary school students know that the equation for a parabola is y = a*(x*x).

- Now, for every input / output pair, instead of having a lookup table that said: map input -1 to output 2, map input 0 to output 0, etc. …

- … we could use a mathematical expression -> y = 2*(x*x), to achieve the same functionality.

Math to the Rescue

- But what if “tabular” world could not be modeled by a parabola?
- Say, x = 1 mapped to 20, instead of 2.
- Or, what if the input domain for “tabular” world consisted of 100 input variables, instead of just 1?
- Then what?

Math to the Rescue - Again

- Fortunately, the family of non-linear mathematical functions / expressions is huge.

- Fortunately, the family of multivariate non-linear mathematical functions / expressions is even larger.

- Is there a single, construct that can embody the above, and SIGNIFICANTLY more?

- Yes. That construct is a neural network.

Math to the Rescue - Again

- The classical Universal Approximation Theorem says: A feed forward neural network with 1 hidden layer and a sufficient number of neurons, coupled with non linear activation units, can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary precision.

Math to the Rescue - Again

- So that’s it.
- We are done.
- Problem solved.
- Not quite …
- Determining the weights for a neural network (to achieve good function approximation) can be challenging.

Math to the Rescue - Again

- And it typically is, for problems beyond toy problems.
- Empirically, it was determined that deep networks achieved better results than shallow networks - despite the claim stated the Universal Function Approximation Theorem.
- Deep neural networks required new training methods to encourage neural network learning.
- Dropout, batch norm, skip connections, etc.
- However, great results could be obtained.

Success Stories

- Neural networks have been successfully applied to many problems.

- One famous one, was MNIST digit recognition.

- “Gradient-Based Learning Applied to Document Recognition”, LeCun, et. al. Proceedings IEEE, 1998

- Another famous one, was image classification.

- “ImageNet classification with deep convolutional neural networks”, Krizhevsky, et. al., Advances in NIPS, 2012

The Three Re-occurring Questions

- What architecture do you use?
- What data do you use for training?
- How do you successfully train the architecture with the given data?

Architecture

- Deep Learning moved learning from the realm of “small” and shallow neural networks, to “large” and deep neural networks.

- These deep neural networks ranged from CNN / MLP combinations with hundreds of millions of parameters, to present day (2025) deep transformer networks with 500 billion to 1 trillion parameters.

- In reinforcement learning, we use deep neural networks to learn a policy and / or value functions.

Data

- In reinforcement learning, the data can come in different forms.
- The data can be real world rollouts of an agent acting in an environment.
- The data can be demonstrations of an agent acting in an environment.
- The environment could be real or simulated.
- The data can be collected and reused.
- If the data is reused, the data may have resulted from an agent using a different policy.
- etc.

Training / Learning

- Assuming that you have the right architecture and the right data, how do you train an agent to learn the correct policy?
- What loss function(s) and objectives do you apply?
- How do you prevent the network from completely collapse during training?
- How do you prevent the network from pre-maturely converging to a local minima? i.e. How do you keep exploration alive?
- How do you incorporate “normalization” into the training process?
- etc.

Example: DQN

- Q Learning was a well established technique, based on tabular data.
- Deep Q Learning moved Q Learning into the function approximation space.
- i.e. A tabular look up table was replaced by a deep neural network.
- This deep network determined the appropriate discrete actions for Atari games, given pixel based, input observations.

DQN

- Architecture: CNN and MLP
- Data: Groups of Atari Video Frames
- Training (Tricks) : 1) target network to increase training stability and 2) replay buffer (to increase sample efficiency and to break unwanted temporal correlations)

DQN

The 3 re-occurring questions:

- Correct architecture: Yes
- Correct and adequate data: Yes
- Learning: MSE loss
- Successful Learning: With tricks = Yes. Without tricks = No.