Reinforcement Learning and the Bandit Problem

(Sahit's Guide To Stealing Hearts)

Sahit Chintalapudi

2/14/18

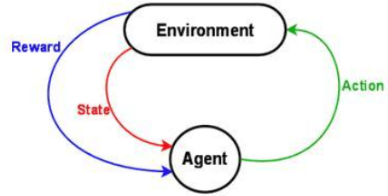
Outline

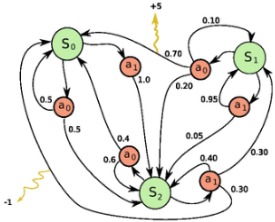
The Bandit Problem

- Consider a slot machine with k arms.
  - Each arm has a different distribution of returns.
  - You don't know which arm can give you highest expected returns.
  - Exploration v. Exploitation
- Who cares?
  - Clinical trials, k possible treatments for a stream of patients.
  - Contextual Bandits, where the world publishes some "context vector", that we use to estimate return distributions.
  - Forcing Valentine's day puns into your talks

Q learning

- Our *agent* is trying to maximize *reward* by learning a function that maps *state, action* pairs to *utility*
$$Q : S \times A \to \mathbb{R}$$
- Reward is kept very low everywhere but terminal states, the agent has to figure out the value for other states
- With our function Q we can execute a policy π by selecting the action that maps to the highest utility
$$\operatorname*{argmax}_{a \in A} Q(S, a)$$





The Bellman Update

- We want to learn a function Q that reflects the reward at the current state as well as (an expectation of) discounted future rewards.
$$Q(s, a) = R(s, a) + \gamma \underset{a \in A}{\text{argmax}} Q(s', a)$$
- Build a Q function by simulating explorations of the state space.
- Choose an action greedily but with some randomness
- Update Q with new information after every transition
$$Q(s, a) = Q(s, a) + \alpha(R(s, a) + \gamma(argmax((s', a) - Q(s, a))$$

Q Networks

- Sounds like we already have a pretty good tool for learning functions
- We want to learn a function that maps states to vectors in $\mathbb{R}^A$
- We don't have to worry about discretization of the state space
- Loss given by MSE

$$L = \sum (r + \gamma max Q(s', a) - Q(s, a))^2$$

- We can now advance this model with some ideas we've seen before as well as some new ideas

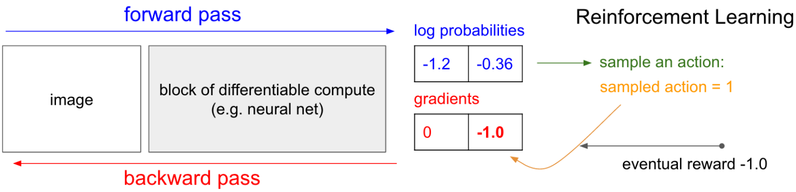
Deep Q Networks

- Unsuprisingly, convolutions over the input give us a better representation of space, so we see better results.
- *Experience replay*: Instead of training on consecutive (s, a, r, s') examples, which drives the model into local minima we randomly sample from old transitions in the training process.
- *Learning Atari games!*

Policy Gradients

- A new take on the RL problem: instead of trying to infer utilities "Q" and then execute a policy based on that, iteratively learn a policy.
- Maximize the total of future expected Rewards

$$\nabla_\theta E[R_t] = E[\nabla_\theta log P(A) R_t]$$

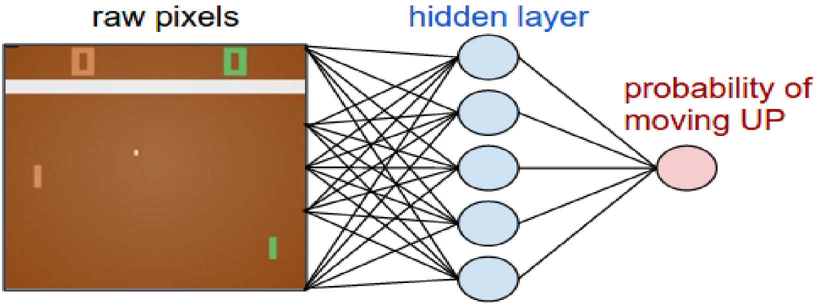


Policy Gradients in the wild

- Think of this as a supervised learning problem where the labels are given by the eventual reward
- We're searching directly in the "policy space", so this approach tends to generalize better
- Let A_i be reward. Our loss takes the form:

$$\sum_i A_i logp(y_i|x_i)$$

- *Walking becomes easy*



Distributional RL

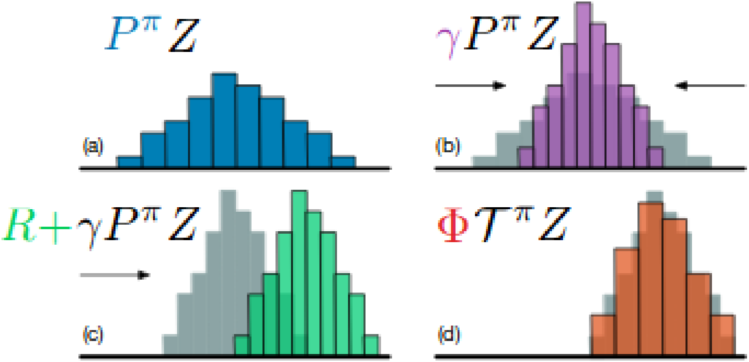
- Remember the Bellman equation? What we're really saying is:

$$Q^\pi(s, a) = \mathbb{E}[R_t] = \mathbb{E}R(s, a) + \gamma\mathbb{E}Q(s', a')$$

- How can we make this better? Let Z be a probability distribution we refer to as the *value distribution*:

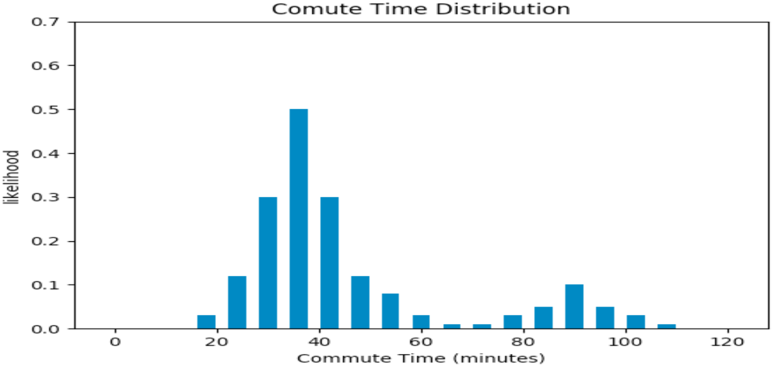
$$Z(x, a) = R(x, a) + \gamma Z(X', A')$$

- The last step below is a projection of $\mathbb{Z}'$ onto *supports* of $\mathbb{Z}$



Why is learning the Distribution a good idea?

- Sometimes the distribution of rewards is *multi-modal*, an expectation can't capture this
- If we're risk averse, we can decide to choose an action that leads to a reward with lesser variance



Recurrent Q Networks

- *Partially Observable Markov Decision Process (POMDP):* We don't have the entire state
- RNNs give our network "attention"
- insert an LSTM block after the last convolutional layer
- In the replay buffer, store experiences of a fixed length (as opposed to just a transition)
- *It's pretty good at DOOM*

Bandit Convex Optimization

- BCO is an interesting subfield of optimization that tries to bound errors on algorithms solving the bandit problem
- We talk about bounds in terms of regret, where regret is given by

$$R_n = max_i \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}$$

Citations 4 dayz/Further Reading

- Arthur Juliani's Medium Posts on DQNs
- Felix Yu's Blog Posts on Distribution RL/Policy Gradients
- Andrej Karpathy's Post on Policy Gradient
- CS 294 at UC Berkeley
- Intel AI explaining DQNs
- Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning.
- Sebastien Bubeck and Nicolo Cesa-Bianchi. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems