Function approximation.

Last lectures

- Calculating $Q(s, a)$ by different methods:
  - Montecarlo.
  - Temporal Differences (TD):
    - On-policy: Sarsa
    - Off-policy: Q-Learning
- Core idea: Store in a lookup table the statistics of how good an action is on every state.

Drawback

- Not really useful to solve really **large** problems:
  - Backgammon: 10^{20} states.
  - Computer Go: 10^{170} states.
  - Chess: 10^{120} states.
  - Helicopter flying: continuous state space.
  - Atoms in the observable universe: 10^{81}.
- **Generalization**: how can we learn about the rest of the world from limited experience?

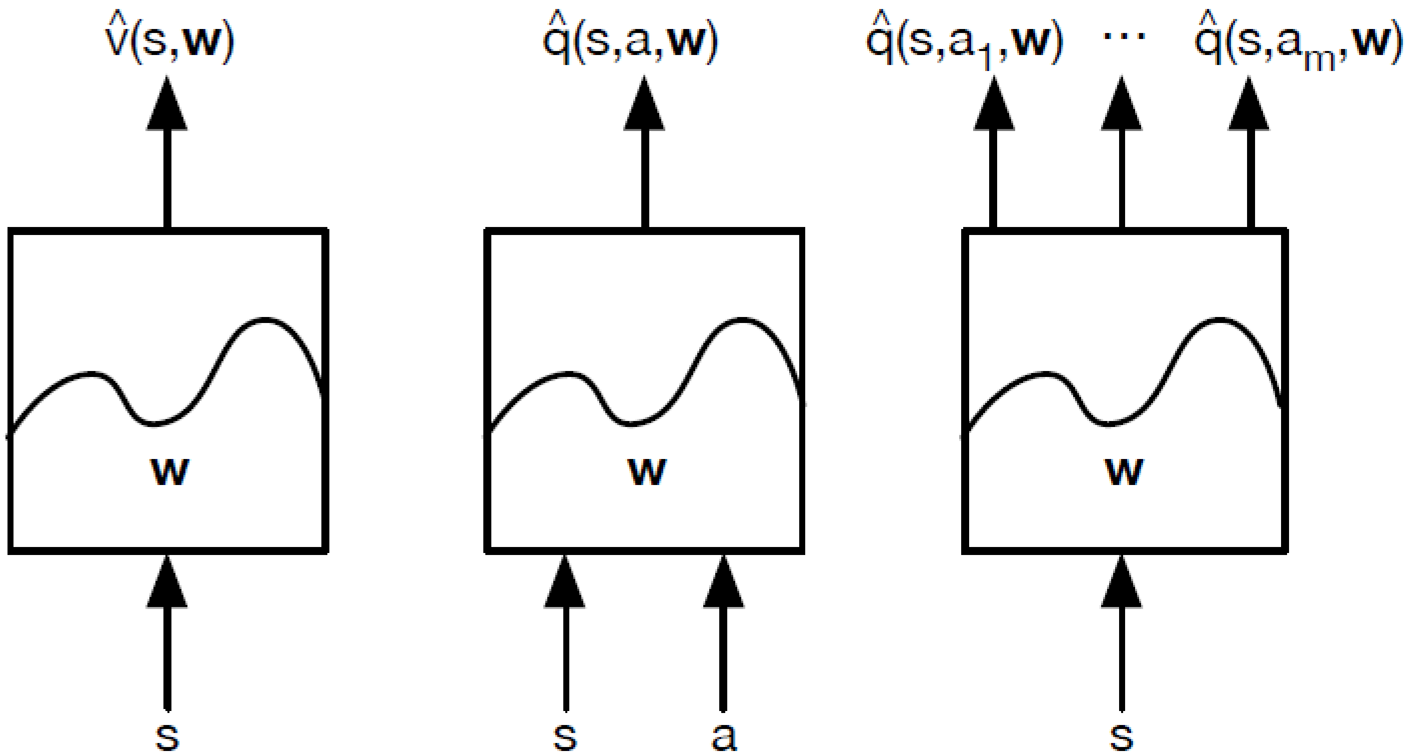
How can we solve such problems?

- **Function approximation**
- The goal is to find a parameter vector θ such that

$$\hat{Q}(s, a, \theta) \approx Q(s, a)$$

- θ might be, for instance:
  - Weights in a neural network.
  - Coefficients for a linear regression model.

Value function approximation?



Function approximators

In principle, you can try anything:

- Linear combination of features.
- Neural networks.
- Random Forests.
- Fourier/wavelet bases.

We focus on differentiable function approximators, as this allows us to define "good" search directions to look at.

So this is supervised learning?

- Not really!
- The data is **not stationary**:
  - A modification of the policy parameter θ would have influence on the rest of the trajectory!

Incremental methods

FrozenLake revisited

- https://gym.openai.com/evaluations/eval_xqVczXQDREisq4xa2Eb5kg

Gradient descent

- Let $J(\theta)$ be a differentiable function of θ.
- The **gradient** of $J(\theta)$ is:

$$\nabla_\theta J(\theta) = \left(\frac{\partial J(\theta)}{\partial \theta_1}, \ldots, \frac{\partial J(\theta)}{\partial \theta_n} \right)^T$$

- To find a local minimum:
  - Change θ in the direction of the gradient:
  - $\Delta\theta := -\frac{1}{2}\alpha \nabla_\theta J(\theta)$

    where α is a parameter.

Value function approximation

Goal: Find a parameter vector θ^* that minimizes:

$$J(\theta) := \mathbb{E}\left[(Q(s,a) - \hat{Q}(s,a,\theta))^2\right]$$

where $Q(s,a)$ is the true value function and $\hat{v}(s,\theta)$ is the approximation.

Error functional

- **Goal**: Find a parameter \\theta^* that minimizes:

$$J(\theta) := \frac{1}{2} \sum_{s \in \mathcal{S}, a \in \mathcal{A}} \left[\left(Q(s,a) - \hat{Q}(s,a,\theta) \right)^2 \right]$$

where $Q(s,a)$ is the true value function and $\hat{Q}(s,a,\theta)$ is the approximation.

By gradient desscent:

$$
\begin{aligned}
\Delta\theta = & -\frac{1}{2}\alpha\nabla_\theta J(\theta) \\
= & \alpha\mathbb{E}\left[(Q(s,a) - \hat{Q}(s,a,\theta))\right]\nabla_\theta\hat{Q}(s,a,\theta)
\end{aligned}
$$

- Drawback:
  - We still need to calculate the expectation! (pass over all states).

Stochastic gradient descent

- Sample the gradient instead!
  - Take only one step.
  - Update your parameter θ according to:
  - $$\Delta\theta = \alpha(Q(s,a) - \hat{Q}(s,a,\theta))\nabla_\theta \hat{Q}(s,a,\theta)$$

Feature vectors

- Represent the full state as **features**:
- These could be
  - Distance from a robot to (each) wall.
  - PCA decomposition

Embedding

We have an embedding of the state space into a smaller dimensional space:

$$s \mapsto \mathbf{x}(s) := (\mathbf{x}_1(s), \mathbf{x}_2(s), \ldots \mathbf{x}_m(s))$$

Linear value function approximation

- Represent the value function as a linear combination of features:

$$\hat{v}(s, \theta) := \sum_{i=1}^{m} \mathbf{x}_i(s)\theta_i$$

The update becomes:

$$\Delta\theta = \alpha(Q(s, a) - \hat{Q}(s, a, \theta))\mathbf{x}(s)$$

where $\mathbf{x}(s) = (\mathbf{x}_1(s), \mathbf{x}_2(s), \ldots \mathbf{x}_m(s))$

Remarks

- Table lookup is a special case of value function approximation.
- We are **cheating**: we do not know the value function, (there is no supervisor), we only have rewards.

Function backups

- DP: $s \mapsto \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \cdot \max_{a' \in \mathcal{A}} \hat{Q}(s', a', \theta)$
- Monte-Carlo: $s \mapsto G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$
- TD: $s \mapsto R_{t+1} + \gamma \hat{Q}(S_{t+1}, A_{t+1} \theta)$

Function backups

- In general, we have something of the form $s \mapsto g$, where g is some target value.
  - Up to now, trivial updates: move the estimated value "a bit more" towards g.
  - Viewing each backup as a *training example* we can use any **supervised learning** method to estimate the value function.

How to stop cheating?

- Instead of the true value function $v(s)$, or the action-value function $Q(s, a)$, we plug in the corresponding updates as in the previous slide.

Convergence

- **Bootstrapping**
  - Updating the value function from other estimates.
- Off-policy methods do not backup state and action pairs with the same function they are estimating!
- At least theoretically, it is possible that the Q $-$learning with function approximation will not converge.
  - In practice, it does.

MountainCar Demo

MountainCar

- https://github.com/dennybritz/reinforcement-learning
- State: 2 parameters (x, y)
- Action: Accelerate backward (0), Stay (1), Accelerate forward (2)
- An episode is solved if you get -110 points over 100 consecutive trials.