# Function Approximation

## Gradient Descent and Linear Models

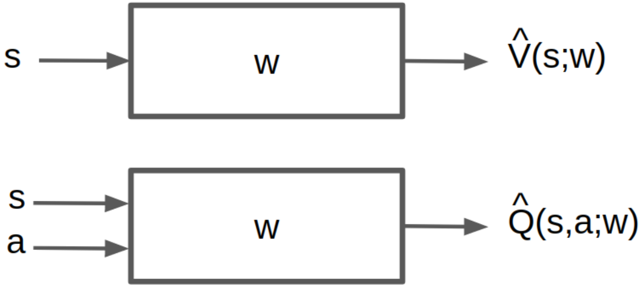
Marius Lindauer

Leibniz Universität Hannover

Winter Term 2021

# Overview

- Represent a (state-action/state) value function with a parameterized function instead of a table




- Which function approximator

# Function Approximators

- Many possible function approximators including
  - linear combinations of features
  - Neural networks
  - Decision trees
  - Nearest neighbors
  - Fourier / wavelet bases
- Focus on differentiable function approximators
- Let's start with linear feature representations

# Recap: Gradient Descent

- Consider a function $J(\vec{w})$ that is differentiable function of a parameter vector $\vec{w}$
- Goal is to find parameter $\vec{w}$ that minimizes $J$
- The gradient of $J(\vec{w})$ is

$$\nabla J(\vec{w}) = \left[\frac{\partial J}{\vec{w}_1} \dots \frac{\partial J}{\vec{w}_n}\right]$$

$$\vec{w}_t = \vec{w}_{t-1} - \alpha \nabla_w J(\vec{w})$$

where $\alpha$ is the learning rate.

# Value Function Approximation for Policy Evaluation with an Oracle

- First assume we could query any state $s$ and an oracle would return the true value for $V^\pi(s)$
- The objective was to find the best approximate representation of $V^\pi$ given a particular parameterized function

# Stochastic Gradient Descent

- Goal: Find the parameter vector $\vec{w}$ that minimizes the loss between a true value function $V^\pi(s)$ and its approximation $\hat{V}^\pi(s; \vec{w})$ as represented with a particular function class parameterized by $\vec{w}$.
- Generally use mean squared error and define the loss as

$$J(\vec{w}) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}^\pi(s; \vec{w}))^2]$$

- Use gradient descent to find a local minimum

$$\Delta\vec{w} = -\frac{1}{2}\alpha\nabla_{\vec{w}J(\vec{w})}$$

- Stochastic gradient descent (SGD) uses a finite number of samples to compute an approximate gradient:

$$\nabla_{\vec{w}J(\vec{w})=\nabla_{\vec{w}}\mathbb{E}_\pi[V^\pi(s)-\hat{V}^\pi(s;\vec{w})]^2}$$
$$= \mathbb{E}_\pi[2(V^\pi(s) - \hat{V}^\pi(s; \vec{w}))\nabla_{\vec{w}\hat{V}(s;\vec{w})}]$$

# Model Free VFA Policy Evaluation

- In practice, we don't actually have access to an oracle to tell true $V^\pi(s)$ for any state $s$
- Now consider how to do model-free value function approximation for prediction / evaluation / policy evaluation without a model

# Model Free VFA Prediction / Policy Evaluation

- Recall model-free policy evaluation
  - Following a fixed policy $\pi$ (or had access to prior data)
  - Goal is to estimate $V^{\pi}$ and/or $Q^{\pi}$
- Maintained a lookup table to store estimates $V^{\pi}$ and/or $Q^{\pi}$
- Updated these estimates after each episode (Monte Carlo methods) or after each step (TD methods)
- New: in value function approximation, change the estimate update step to include fitting the function approximator

# Model Free VFA Prediction / Policy Evaluation

- Use a feature vector to represent a state $s$

$$\vec{x}(s) = \begin{pmatrix} \vec{x}_1(s) \\ \vec{x}_2(s) \\ \dots \\ \vec{x}_n(s) \end{pmatrix}$$

- For table lookups, we have not really needed that because we only needed to know which table entry to look up

# Linear Value Function Approximation for Prediction With An Oracle

- Represent a value function (or state-action value function) for a particular policy with a weighted linear combination of features

$$\hat{V}^\pi(s;\vec{w}) = \sum_{j=1}^{n} \vec{x}_j(s)\vec{w}_j = \vec{x}(s)^T\vec{w}$$

- Objective function is

$$J(\vec{w}) = \mathbb{E}[(V^\pi(s) - \hat{V}^\pi(s;\vec{w}))^2]$$

- Recall weight update:

$$\Delta\vec{w} = -\frac{1}{2}\alpha\nabla_{\vec{w}}J(\vec{w})$$

- Update (- step size $\times$ prediction error $\times$ feature value)

$$\Delta\vec{w} = -\frac{1}{2}\alpha(2(V^\pi(s) - \vec{x}(s)^T\vec{w}))\vec{x}(s)$$