Function Approximation

Gradient Descent and Linear Models

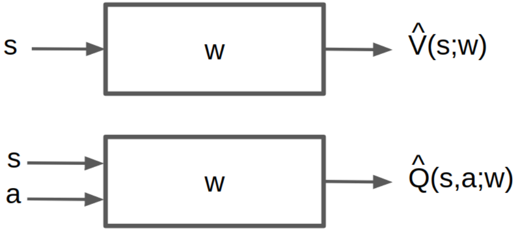
Marius Lindauer

Leibniz Universität Hannover

Automated Machine Learning Hannover

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

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

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

where α 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^π given a particular parameterized function

Stochastic Gradient Descent

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

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

- Use gradient descent to find a local minimum

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

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

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

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

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

Automated Machine Learning Hannover

Model Free VFA Prediction / Policy Evaluation

- Recall model-free policy evaluation
  - Following a fixed policy π (or had access to prior data)
  - Goal is to estimate V^π and/or Q^π
- Maintained a lookup table to store estimates V^π and/or Q^π
- 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

Automated Machine Learning Hannover

Model Free VFA Prediction / Policy Evaluation

- Use a feature vector to represent a state s

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

- 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

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

- Objective function is

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

- Recall weight update:

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

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

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

Automated Machine Learning Hannover

AutoML.org