Google Developers

Experts

근사를 이용한 활성 정책 예측

강화학습

송호연 @chris_loves_ai
sjhshy@gmail.com

가치 함수 근사
Value Function Approximation

대규모 강화학습

Large-Scale Reinforcement Learning

- 강화학습은 대규모의 문제를 해결하는 데 활용할 수 있음
  - Backgammon: 10^20 상태 수
  - Computer Go: 10^170 상태 수
  - Helicopter: 연속 상태 수

가치 함수 근사

Value Function Approximation

- 모든 예측 방법은 특정 상태에서의 가치를 그 상태에 대한 '보강된 가치'
  또는 갱신 목표를 향해 이동시키기 위한 가치 함수 추정값의 갱신으로 설명됨

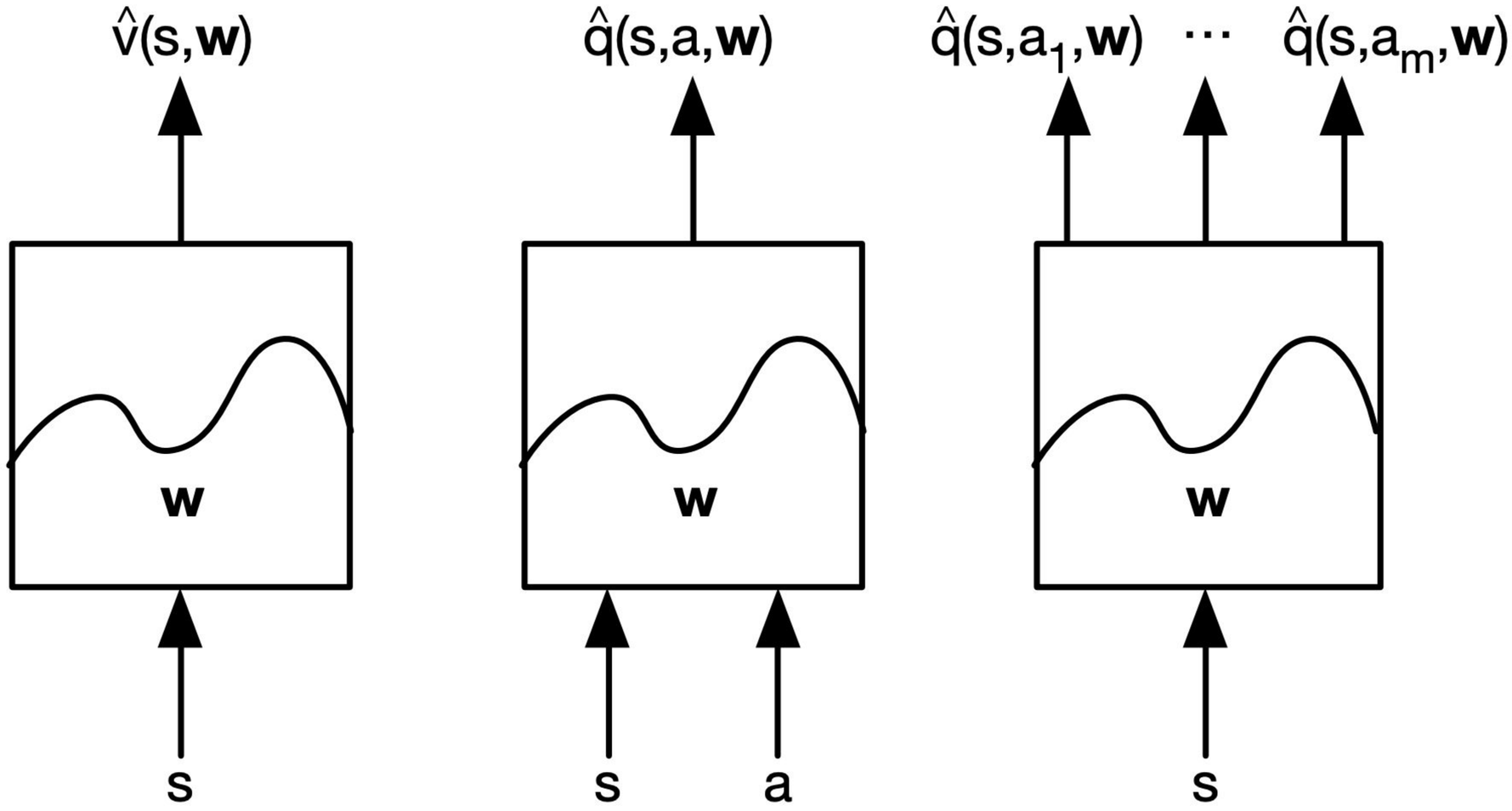
- 입출력 예제를 모사하기 위해 이러한 방식으로 학습하는 기계학습 방법을 지도학습
  방법이라고 하고,
  출력이 u와 같은 숫자일 경우 이 과정은 종종 함수 근사(function approximation)라고도
  한다.

예측 목적

VE, 평균 제곱 오차, Mean Squared Value Error

- **모델(model)**이란 환경이 행동에 어떻게 반응할 것인지를 예측하기 위해 학습자가 사용할 수 있는 모든 것을 의미한다.

예측 목적


$\hat{v}(s,\mathbf{w})$
$\hat{q}(s,a,\mathbf{w})$
$\hat{q}(s,a_1,\mathbf{w})$ ⋯ $\hat{q}(s,a_m,\mathbf{w})$
w
w
w
s
s
a
s

어떤 근사 함수를 사용할까?

planning

- 미분가능한 함수
  - 선형 방정식
  - 인공 신경망
  - 의사결정트리
  - ...

- 또한 우리는 iid가 아니고 정적이지 않은 데이터로부터 학습할 수 있는 방법들을 필요로 함

비선형 함수 근사: 인공 신경망

Artificial Neural Network

ANN은 비선형 함수 근사에 폭넓게 활용된다.

활성화 함수(activation function)

Gradient Descent

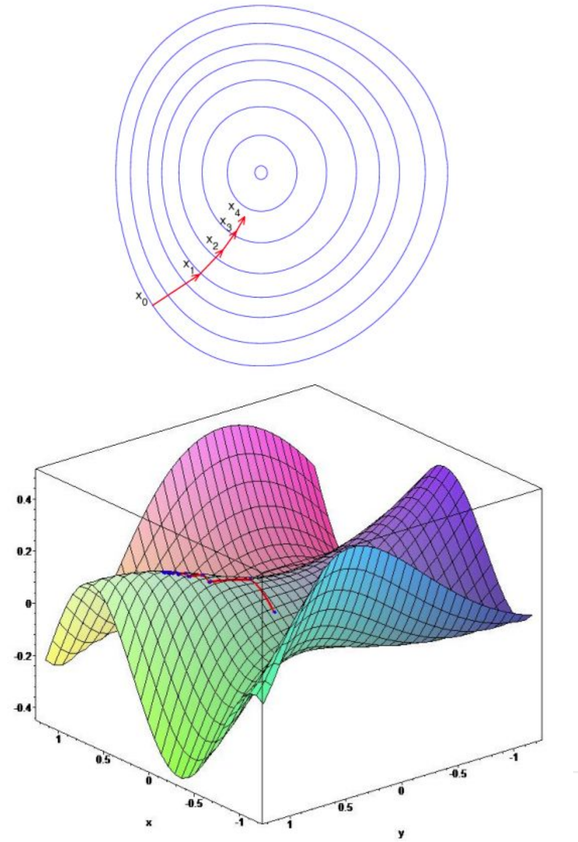
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$
- Adjust $\mathbf{w}$ in direction of -ve gradient

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

where α is a step-size parameter



Feature Vectors

- Represent state by a *feature vector*

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Feature Vectors

- Represent state by a *feature vector*

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Experience Replay in DQN

DQN uses experience replay and fixed Q-targets

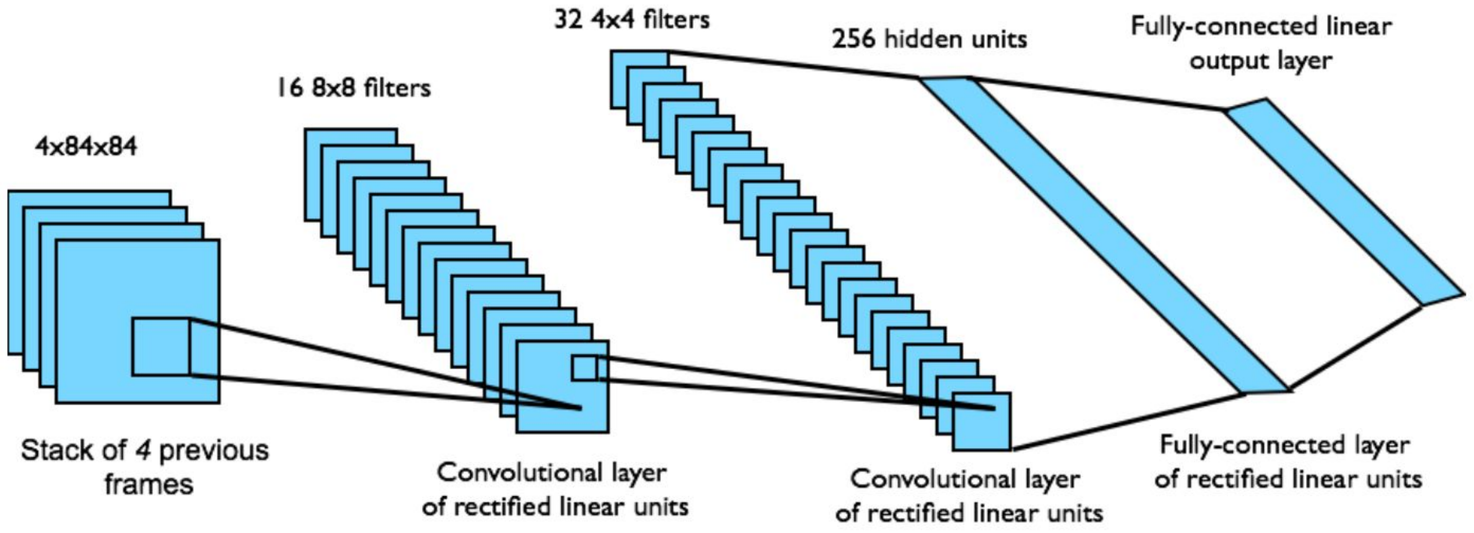
- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]$$

- Using variant of stochastic gradient descent

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

선형 가치 함수 근사
Linear Least Squares Prediction

Linear Least Square Prediction

- Experience replay finds least squares solution
- But it may take many iterations
- Using *linear* value function approximation $\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^\top \mathbf{w}$
- We can solve the least squares solution directly

E.O.D.