RL Algorithms Continued

Q-Learning Algorithm

$$Q^*(s, a) = \qquad E(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a)$$

- Initialize the Q-table

- Until convergence:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \Bigg)}^{\text{temporal difference}}$$

Q-Learning Pseudocode

Algorithm 1: Epsilon-Greedy Q-Learning Algorithm

```
Data: α: learning rate, γ: discount factor, ε: a small number
Result: A Q-table containing Q(S,A) pairs defining estimated
        optimal policy π*
/* Initialization                                              */
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) ← 0;
/* For each step in each episode, we calculate the
   Q-value and update the Q-table                              */
for each episode do
  /* Initialize state S, usually by resetting the
     environment                                               */
  Initialize state S;
  for each step in episode do
    do
      /* Choose action A from S using epsilon-greedy
         policy derived from Q                                 */
      A ← SELECT-ACTION(Q, S, ε);
      Take action A, then observe reward R and next state S';
      Q(S, A) ← Q(S, A) + α [ R + γ max_a Q(S', a) - Q(S, A)];
      S ← S';
    while S is not terminal;
  end
end
```

Algorithm 2: Epsilon-Greedy Action Selection

```
Data: Q: Q-table generated so far, : a small number, S: current
      state
Result: Selected action
Function SELECT-ACTION(Q, S, ε) is
  n ← uniform random number between 0 and 1;
  if n < ε then
    A ← random action from the action space;
  else
    A ← maxQ(S,.);
  end
  return selected action A;
end
```

More Implementation Examples

- A. Rao, T. Jelvis. Foundations of reinforcement learning with applications to finance.

RL for functional approximation

- Tabular Q-Learning does not scale with increase of size of state space - too many states to visit
- E.g. - need to discretize continuous state spaces, not scalable!
- Need to be able to generalize to unseen states
- Let θ be a parameter
- Instead of learning a Q-table, learn a function $Q_\theta(s, a)$ so that for every *s, a*

$$Q_\theta(s, a) = \qquad E(r_{t+1} + \gamma \max_{a'} Q_\theta(s_{t+1}, a') \mid s_t = s, a_t = a)$$

Gradient Descent Version ofQ-Learning

$$Q_\theta(s,a) = \quad E(r_{t+1} + \gamma \max_{a'} Q_\theta(s_{t+1}, a') \,|\, s_t = s, a_t = a)$$

Minimize the loss

$$\ell_\theta(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} (Q_\theta(s,a) - R(s,a,s') - \gamma \max_{a'} Q_\theta(s',a'))^2 =: \mathbb{E}_{s' \sim P(s,a,\cdot)} [\ell_\theta(s,a,s')]$$

Start with initial state $s = s_0$. In iteration $k = 1, 2, ...,$

- Take an action a.
- Observe reward r, transition to state $s' \sim P(\cdot|s,a)$.
- $\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla_{\theta_k} \ell_{\theta_k}(s,a,s')$, where

$$\nabla_\theta \ell_{\theta_k}(s,a,s') = -\delta_k \nabla_{\theta_k} Q_{\theta_k}(s,a)$$
$$\delta_k = r + \gamma \max_{a'} Q_{\theta_k}(s',a') - Q_{\theta_k}(s,a)$$

- $s \leftarrow s'$,

Linear Function Approximation

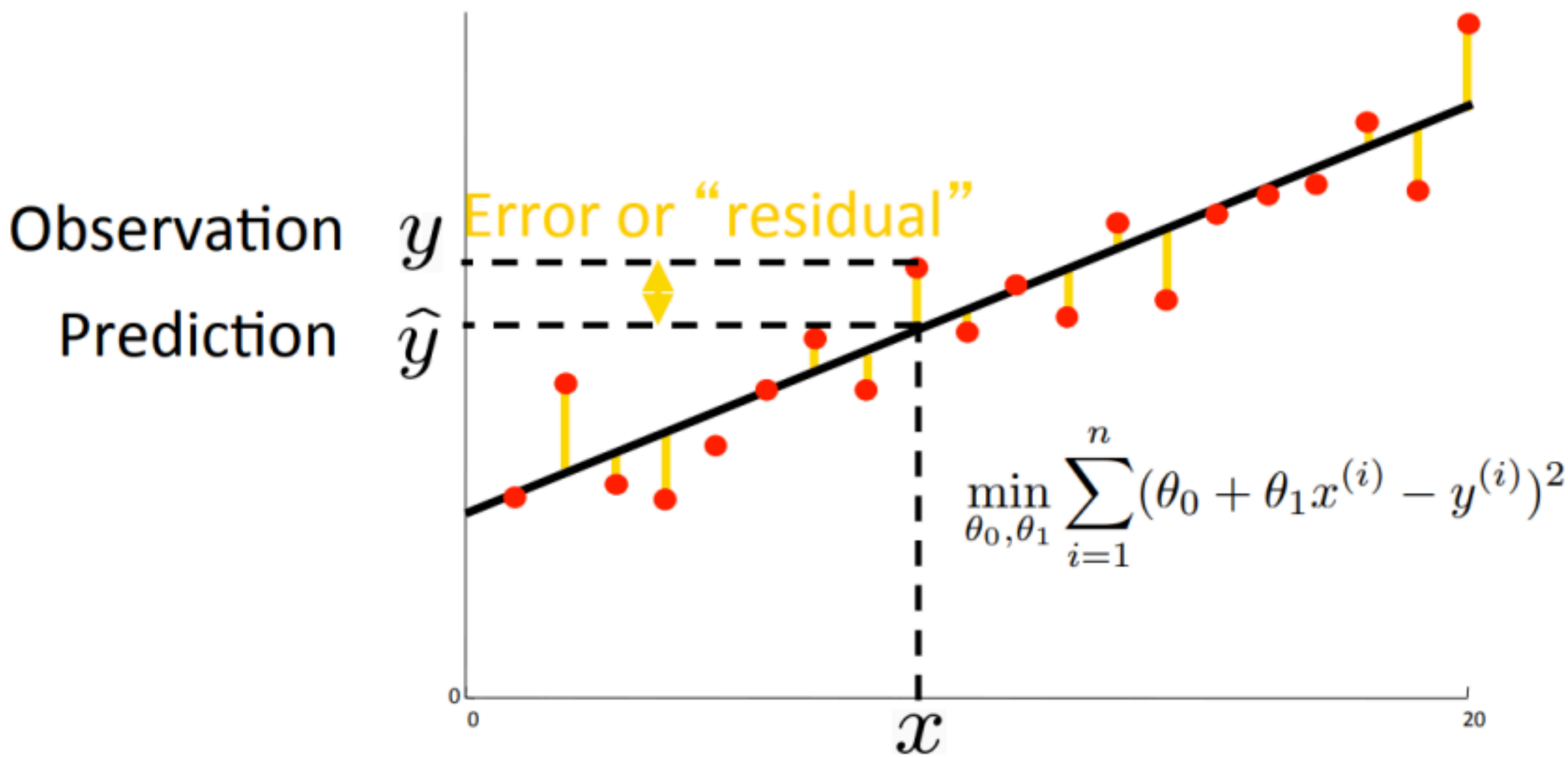
- Learn parameter θ
- Linear function approximation

$$Q(s,a) = \theta_0 \cdot 1 + \theta_1 \phi_1(s,a) + \cdots + \theta_n \phi_n(s,a) = \theta^T \phi(s,a)$$

- Example: The features for state action pair (s,a_i) - two state features, four actions - can be encoded as

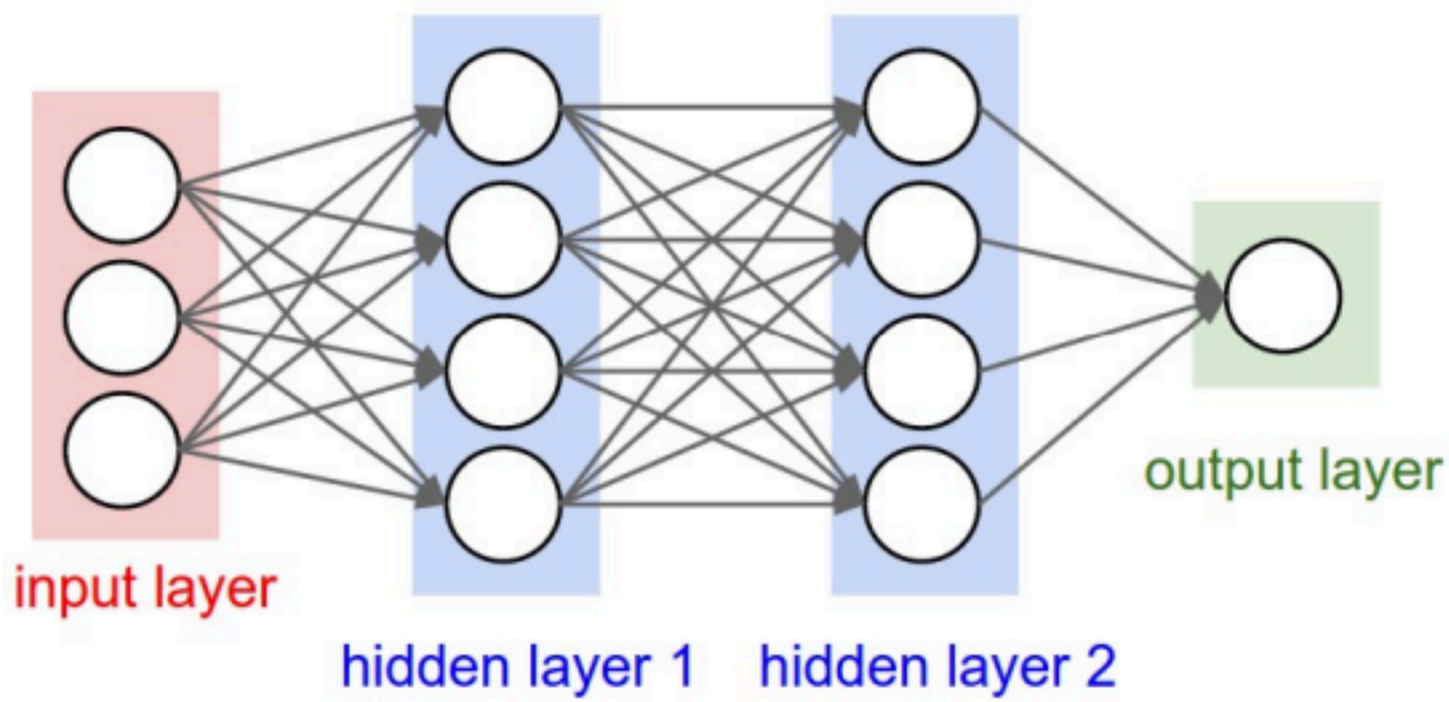
$$\phi(s,a_1) = \begin{bmatrix} \psi_1(s,a_1) \\ \psi_2(s,a_1) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \phi(s,a_2) = \begin{bmatrix} 0 \\ 0 \\ \psi_1(s,a_2) \\ \psi_2(s,a_2) \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \phi(s,a_3) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \psi_1(s,a_3) \\ \psi_2(s,a_3) \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \phi(s,a_4) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \psi_1(s,a_4) \\ \psi_2(s,a_4) \\ 1 \end{bmatrix}$$

Gradient Descent: Intuition from Linear Regression



Neural Network Function Approximation

- Learn parameter θ
- $Q_\theta(s,a) = f_\theta(\phi(s,a))$, where θ is a parameter of neural network
- Can generalize to unseen states better



Experience Replay

- In traditional Q-learning, each experience is used once at a time and discarded
- Inefficient use of Data, takes very long time to train neural networks
- Use **experience replay** instead:
  - Store atomic experiences *(s_i, a_i, r_{i+1}, s_{i+1})* in replay memory
  - During training, sample experiences from replay memory

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies