RLearning:
Short guides to reinforcement learning

Unit 4-1: Neural Networks

Davud Rostam-Afschar (Uni Mannheim)

How to deal with very large state-action spaces?

Tabular Value Iteration and Q-Learning

- Markov Decision Processes: value iteration

$$V(s) \leftarrow \max_a R(s) + \gamma \sum_{s'} \mathbb{P}\left(s' \mid s, a\right) V\left(s'\right)$$

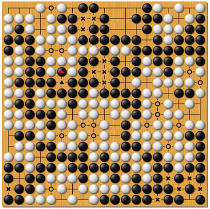
- Reinforcement Learning: Q-Learning

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_a Q\left(s', a'\right) - Q(s, a)\right]$$

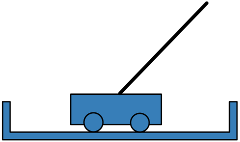
- Complexity depends on number of states and actions

Large State Spaces

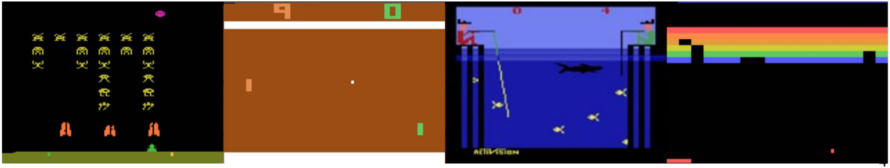
- Computer Go: 3^{361} states



- Inverted pendulum: (x, x', θ, θ')
  - 4-dimensional
  - continuous state space



- Atari: $210 \times 160 \times 3$ dimensions (pixel values)



Functions to be Approximated

- Policy: $\pi(s) \to a$
- Value function: $V(s) \in \mathbb{R}$
- Q-function: $Q(s, a) \in \mathbb{R}$

Q-function Approximation

- Let $s = (x_1, x_2, \ldots, x_n)$
  $\rightarrow$ states are defined by a vector of features x.
- Linear

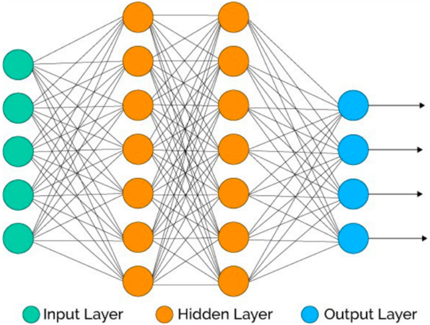
$$Q(s, a) \approx \sum_i w_{ai} x_i$$

- Non-linear (e.g., neural network)

$$Q(s, a) \approx g(\boldsymbol{x}; \boldsymbol{w})$$

Traditional Neural Network

- Network of units (computational neurons) linked by weighted edges

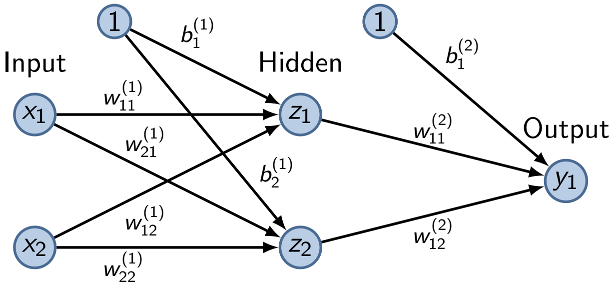


- Each unit computes: $z = h(\boldsymbol{w}'\boldsymbol{x} + b)$
  - Inputs: $\boldsymbol{x}$
  - Output: z
  - Weights (parameters): $\boldsymbol{w}$
  - Bias: b
  - Activation function (usually non-linear): h

Readings: Deep Neural Networks Goodfellow, Bengio, and Courville (2016, chapters 6, 7, 8)

One hidden Layer Architecture

- Feed-forward neural network



- Hidden units: $z_j = h_1\left(\boldsymbol{w}_j^{'(1)}\boldsymbol{x} + b_j^{(1)}\right)$
- Output units: $y_k = h_2\left(\boldsymbol{w}_k^{'(2)}\boldsymbol{z} + b_k^{(2)}\right)$
- Overall: $y_k = h_2\left(\sum_j w_{kj}^{(2)} h_1\left(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right)$

Common Activation Functions

Common activation functions h

- Threshold:
  $$h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

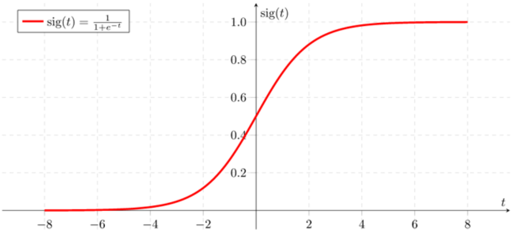
Common activation functions h

- Threshold:
  $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$
- Sigmoid:
  $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$

Common activation functions h

- Threshold:
  $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$

- Sigmoid:
  $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$



Common activation functions h

- Threshold:
  $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$

- Sigmoid:
  $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$

- Gaussian:
  $h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$

Common activation functions h

- Threshold:
  $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$
- Sigmoid:
  $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$
- Gaussian:
  $h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$
- Tanh : $h(a) =$
  $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$

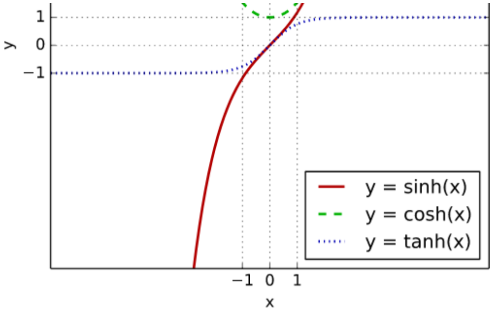
Common activation functions h

- Threshold:
$$h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

- Sigmoid:
$$h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$$

- Gaussian:
$$h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$$

- Tanh : $h(a) =$
$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

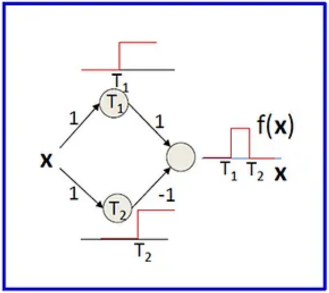


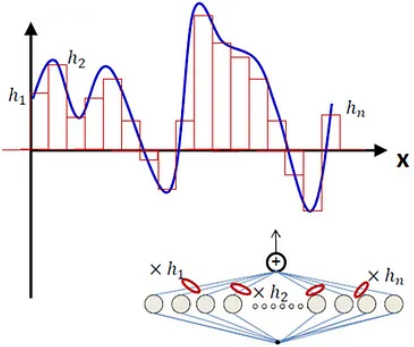
Common activation functions h

- Threshold:
  $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$
- Sigmoid:
  $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$
- Gaussian:
  $h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$
- Tanh : $h(a) =$
  $\tanh(a) = \frac{e^a-e^{-a}}{e^a+e^{-a}}$
- Identity: $h(a) = a$

Universal Function Approximation

- **Theorem:** Neural networks with at least one hidden layer of sufficiently many sigmoid/tanh/Gaussian units can approximate any function arbitrarily closely.





Minimize least squared error

- Minimize error function (Euclidian norm is commonly used for distance)

$$J(\boldsymbol{W}) = \frac{1}{2}\sum_n J_n(\boldsymbol{W})^2 = \frac{1}{2}\sum_n \|f\left(\boldsymbol{x}_n, \boldsymbol{W}\right) - y_n\|_2^2$$

  where J is the error function, f is the function encoded by the neural net and n is the number data points.

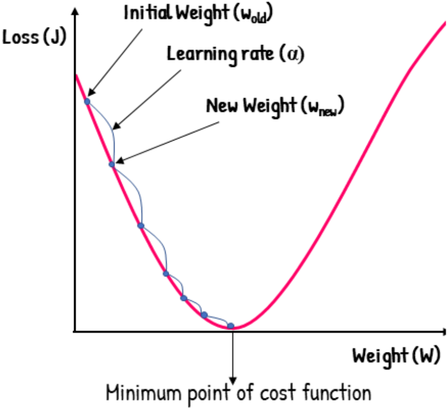
- Train by gradient descent (a.k.a. backpropagation)
  - For each example $(\boldsymbol{x}_n, y_n)$, adjust the weights as follows:

$$w_{ji} \leftarrow w_{ji} - \alpha \frac{\partial J_n}{\partial w_{ji}}$$

α is the stepsize.

Minimize least squared error

- Train by gradient descent (a.k.a. backpropagation)
  - For each example $(\boldsymbol{x}_n, y_n)$, adjust the weights as follows:



$$w_{ji} \leftarrow w_{ji} - \alpha \frac{\partial J_n}{\partial w_{ji}}$$

References I

Goodfellow, I., Y. Bengio, and A. Courville (2016): *Deep learning*, vol. 196. MIT press, Available at http://deeplearningbook.org/.

Takeaways

Neural Nets to Approximate Policies, Value or Quality Functions

- Tabular methods fail in large or continuous state-action spaces
- Neural networks approximate
  - policies,
  - value functions, and
  - Q-functions
- A basic network has
  - weighted inputs,
  - nonlinear activations, and
  - outputs
- Neural networks can approximate any continuous function (universal approximation)