

CS885 Reinforcement Learning

Lecture 4a: May 11, 2018

Deep Neural Networks
[GBC] Chap. 6, 7, 8

Quick recap

- Markov Decision Processes: value iteration

$$V(s) \leftarrow \max_a R(s) + \gamma \sum_{s'} \Pr(s'|s, a) V(s')$$

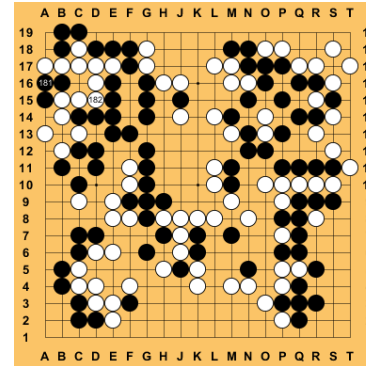
- Reinforcement Learning: Q-Learning

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

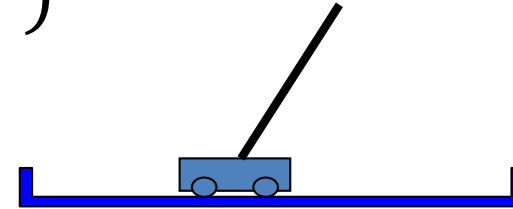
- Complexity depends on number of states and actions

Large State Spaces

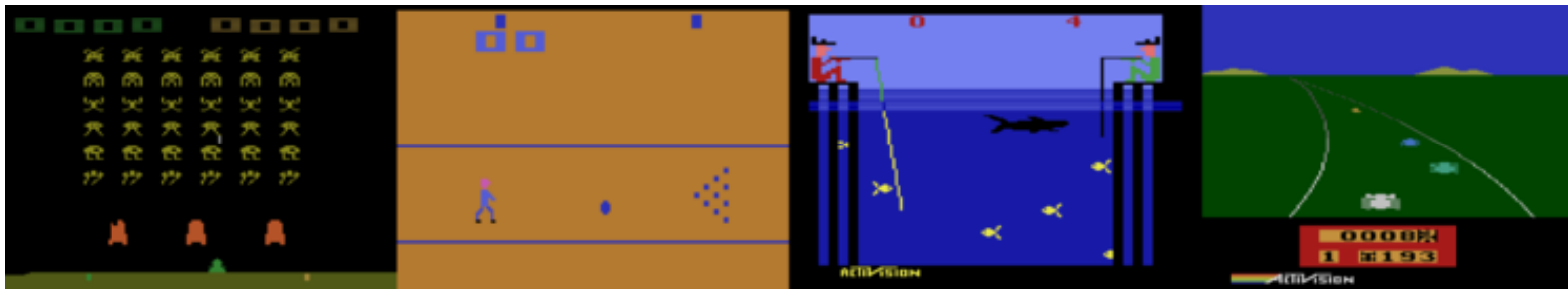
- Computer Go: 3^{361} states



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



- Atari: 210x160x3 dimensions (pixel values)



Functions to be Approximated

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

Q-function Approximation

- Let $s = (x_1, x_2, \dots, x_n)^T$

- Linear

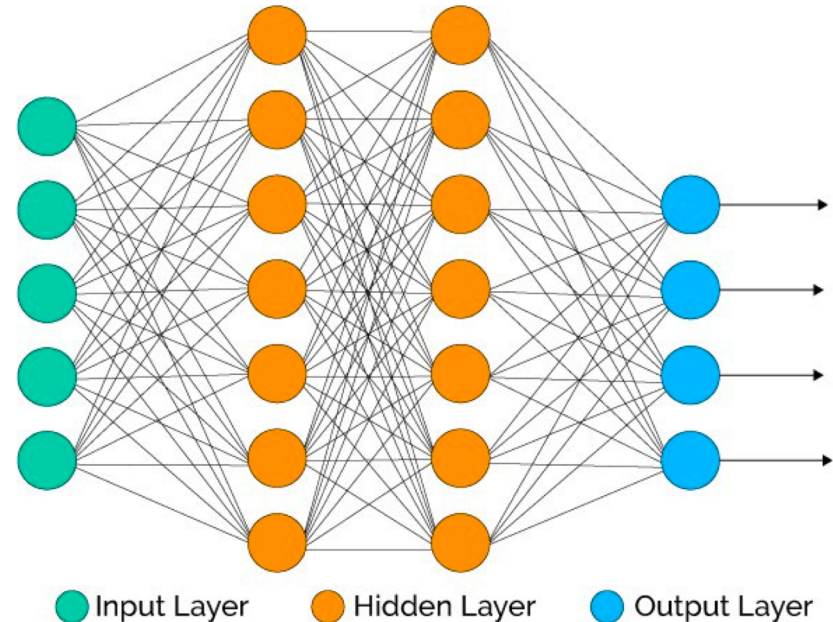
$$Q(s, a) \approx \sum_i w_{ai} x_i$$

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

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

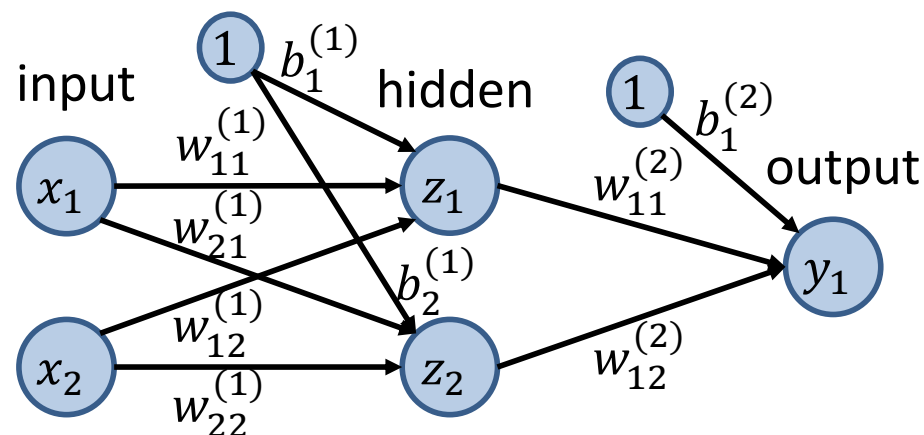
Traditional Neural Network

- Network of units (computational neurons) linked by weighted edges
- Each unit computes:
$$z = h(\mathbf{w}^T \mathbf{x} + b)$$
 - Inputs: \mathbf{x}
 - Output: z
 - Weights (parameters): \mathbf{w}
 - Bias: b
 - Activation function (usually non-linear): h



One hidden Layer Architecture

- Feed-forward neural network



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

Traditional 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.
- Picture:

Minimize least squared error

- Minimize error function

$$E(\mathbf{W}) = \frac{1}{2} \sum_n E_n(\mathbf{W})^2 = \frac{1}{2} \sum_n \|f(\mathbf{x}_n, \mathbf{W}) - y_n\|_2^2$$

where f is the function encoded by the neural net

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

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_n}{\partial w_{ji}}$$

Deep Neural Networks

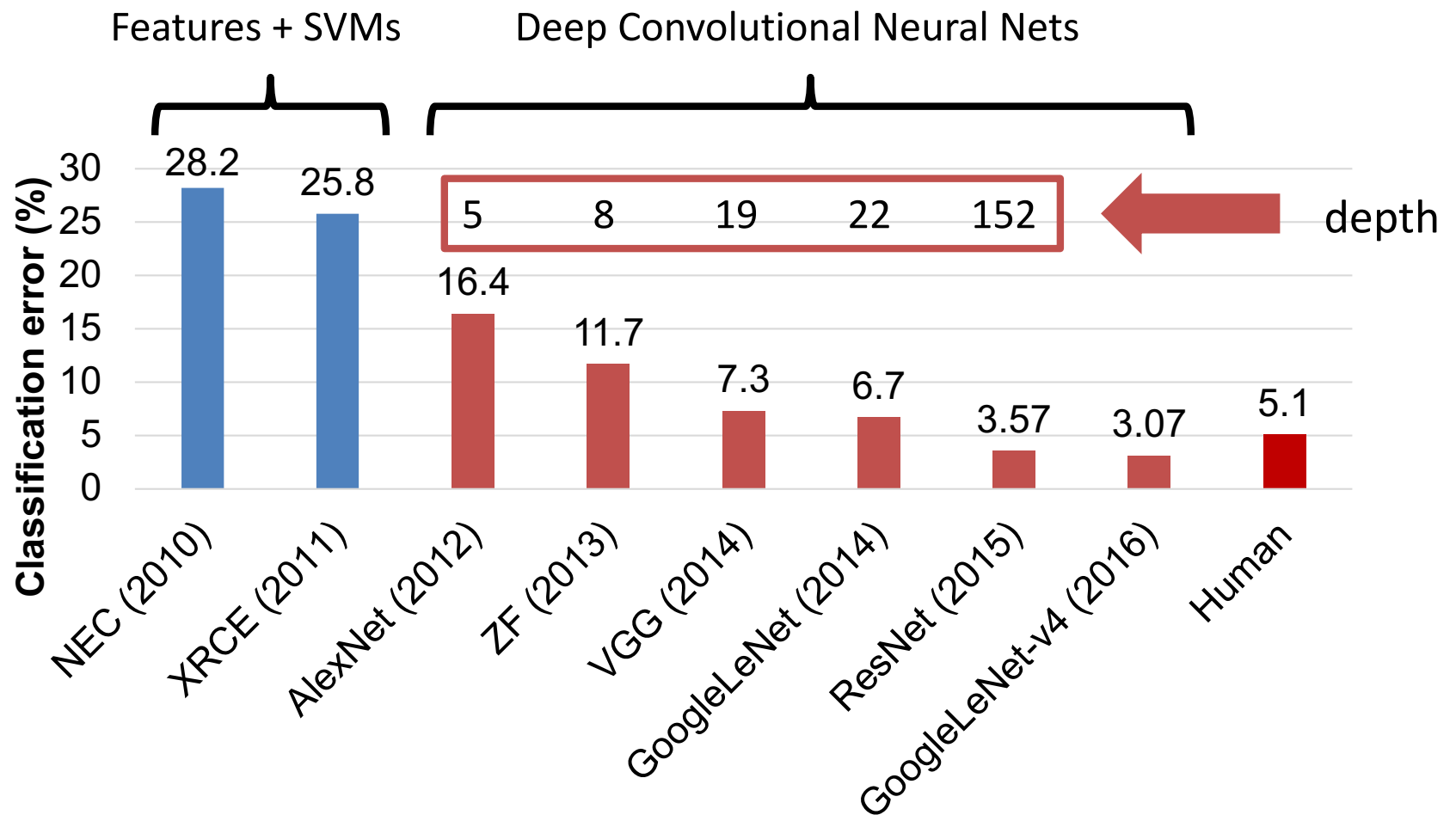
- Definition: neural network with **many hidden layers**
- Advantage: **high expressivity**
- Challenges:
 - How should we train a deep neural network?
 - How can we avoid overfitting?

Mixture of Gaussians

- Deep neural network (hierarchical mixture)
- Shallow neural network (flat mixture)

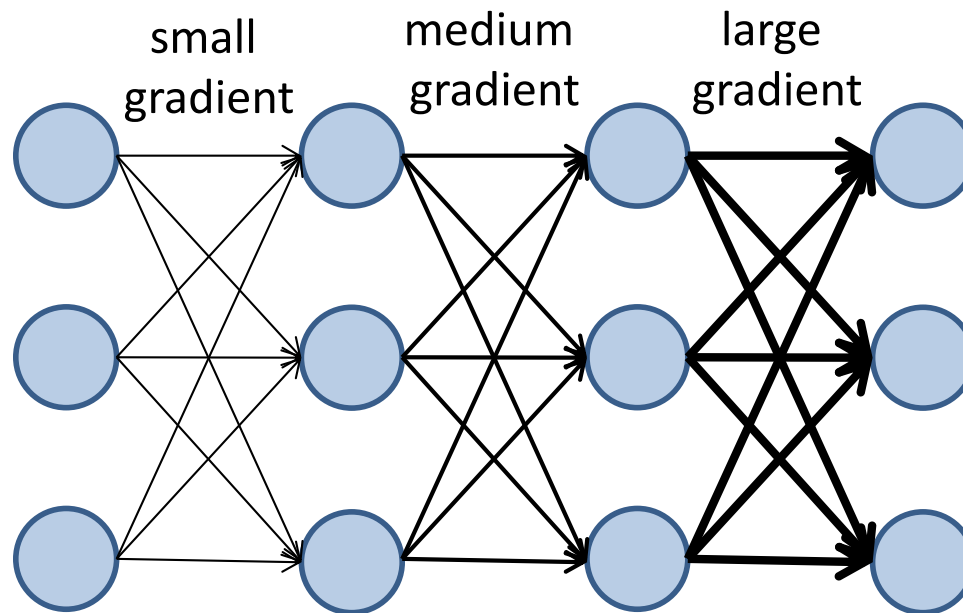
Image Classification

- ImageNet Large Scale Visual Recognition Challenge



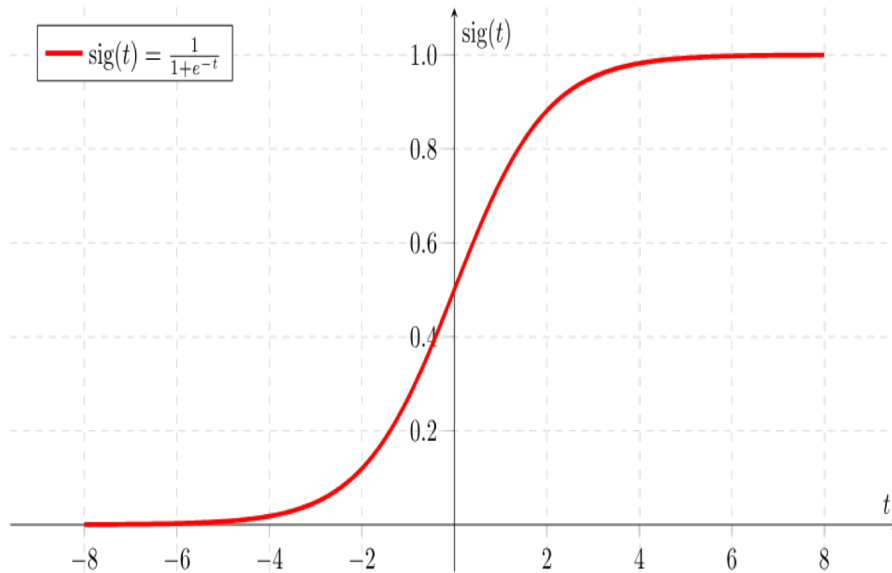
Vanishing Gradients

- Deep neural networks of sigmoid and hyperbolic units often suffer from **vanishing gradients**

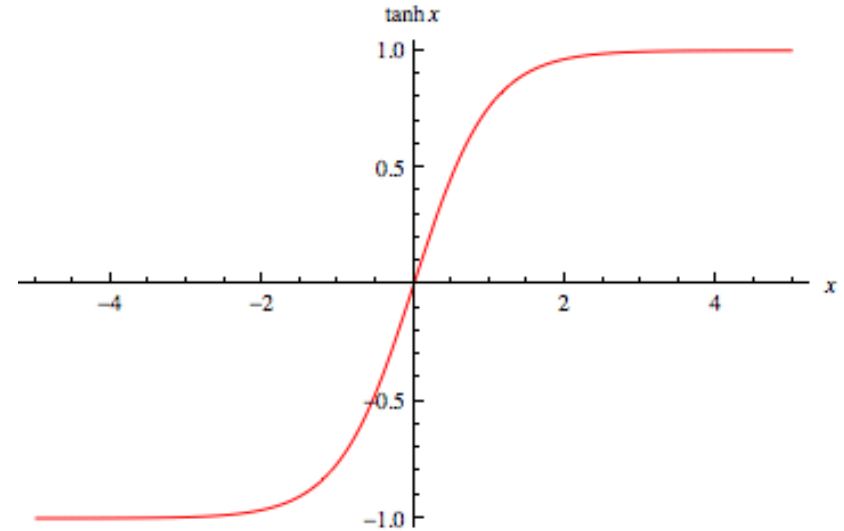


Sigmoid and hyperbolic units

- Derivative is always less than 1



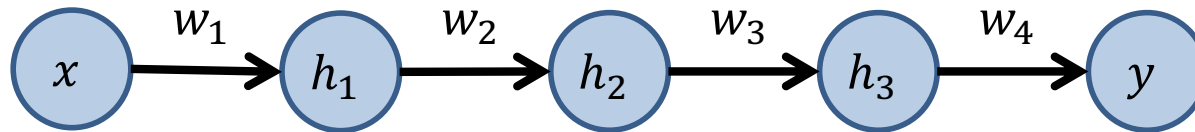
sigmoid



hyperbolic

Simple Example

- $y = \sigma \left(w_4 \sigma \left(w_3 \sigma \left(w_2 \sigma \left(w_1 x \right) \right) \right) \right)$



- Common weight initialization in $(-1,1)$
- Sigmoid function and its derivative always less than 1
- This leads to vanishing gradients:

$$\frac{\partial y}{\partial w_4} = \sigma'(a_4)\sigma(a_3)$$

$$\frac{\partial y}{\partial w_3} = \sigma'(a_4)w_4\sigma'(a_3)\sigma(a_2) \leq \frac{\partial y}{\partial w_4}$$

$$\frac{\partial y}{\partial w_2} = \sigma'(a_4)w_4\sigma'(a_3)w_3\sigma'(a_2)\sigma(a_1) \leq \frac{\partial y}{\partial w_3}$$

$$\frac{\partial y}{\partial w_1} = \sigma'(a_4)w_4\sigma'(a_3)w_3\sigma'(a_2)w_2\sigma'(a_1)x \leq \frac{\partial y}{\partial w_2}$$

Mitigating Vanishing Gradients

- Some popular solutions:
 - Pre-training
 - **Rectified linear units**
 - Batch normalization
 - Skip connections

Rectified Linear Units

- Rectified linear: $h(a) = \max(0, a)$

- Gradient is 0 or 1
- Sparse computation

- Soft version
("Softplus") :

$$h(a) = \log(1 + e^a)$$

- **Warning: softplus does not prevent gradient vanishing (gradient < 1)**

