Learning and Adaptation

(Automated) learning useful when
- constructing a model explicitly (by hand) too difficult
- the model changes over time (adaptation)

This lecture:
- **Reinforcement Learning**: interleaved decision-making and learning when parts of model (initially) unknown
- **Neural Networks**: supervised learning from samples of data

Reinforcement Learning

What if system model (as with MDPs) is incomplete?
- reward function $R(s, a, s')$ is unknown
- transition probabilities $P(s, a, s')$ unknown

Find near-optimal policies by **Reinforcement Learning**:
- Learning and execution are interleaved
- With every new reward and state, update model
Reinforcement Learning

- Applications:
  - robotics
  - control of distributed systems: power, telecom, ...
  - game playing
- Lots of different algorithms and approaches
- Issues:
  - Size of the state space
  - Slow learning when lots of states
  - Particularly difficult with partial observability
- This lecture: brief intro to Q-learning

Q-Learning

Inputs to Q-Learning

- action set $A$
- state set $S$
- discount factor $\gamma$ (as with MDPs)
- learning rate $\lambda \in ]0,1[$ (higher $\rightarrow$ faster learning)

Output: $Q$-values $Q(s,a) : S \times A \rightarrow \mathbb{R}$

- An estimate for the value of taking action $a$ in state $s$
- Summarizes both
  - the transition probabilities from $s$ with $a$, and
  - the values of successors of $s$ with $a$

Q-Learning Algorithm

1. Let $Q(s,a) = 0$ for all $s \in S$ and $a \in A$
2. $s :=$ starting state
3. Execute some action $a$, with new state $s'$ and reward $r$
4. $Q(s,a) := (1 - \lambda)Q(s,a) + \lambda(r + \gamma \cdot \max_{a' \in A} Q(s',a'))$
5. Set $s := s'$ and go to 3

Choice of $a$ at step 3 tries to balance between

- exploration: Improving accuracy of $Q(s,a)$
- exploitation: Taking action $a$ with highest $Q(s,a)$
Exploration vs. Exploitation

Choice of action in $s$ based on $Q(s, a_1), \ldots, Q(s, a_n)$:

- Prefer actions $a$ with high $Q(s, a)$ (exploitation)
- Try also other actions (exploration) to gain more information
- Best to base this on a measure on confidence
  - How much confidence in current $Q(s, a)$?
  - How many times has $a$ been tried before in $s$?
- First more exploration
- Later more exploitation
- Lots of different ways of doing this! (See Multi-armed bandits for solutions that apply to both RL and to MCTS.)

Classification Problems

1. Construct a classifier manually
   - Example: Eligibility for student status (Boolean classifier)
   - Example: Eligibility for a specific type of immigration visa (Boolean classifier)
2. Construct a classifier by supervised learning
   - Manual construction difficult or impossible, when classification problem is too complex or unclearly defined
   - Procedure:
     1. Assign classes to training examples (labeling)
     2. Run a supervised learning algorithm on all training instances
     3. Apply the resulting classifier to new instances
   - Sometimes works really well

Classification Problems

- If $x_1 + 0.625 \cdot x_2 \geq 0.8$ then classify as blue, otherwise red.
  (Many other classifiers $w_1x_1 + w_2x_2 \geq c$ possible.)

Classification Error

- What is the Best Classifier?
  - Often: no perfect classifier, some instances always classified wrong
  - Even when training data fully fits, quality of classifiers vary
  - $\text{Error} = \text{Difference between actual value } y \text{ and desired value } t$
  - Squared error $\sum_i(t_i - y_i)^2$ is often used
Classification Problems

- No (good) linear classifier exists

Classification Generally

- Dividing the $n$-dimensional space to 2 parts (or $m$ parts)
- Linear classification: division by a line, a plane, a hyperplane
- Neural networks one way of doing non-linear classification
- Classify $1000 \times 1000$ pixel images: $10^6$-dimensional space

Supervised Learning

- Decision-tree learning
- Support vector machines (SVM)
- Naive Bayes
- Linear regression
- Neural networks
Neural Networks

- Speech recognition, signal processing
- Natural language processing
- Image classification
- Bioinformatics

Neural Networks

+ “Model” automatically constructed based on training data
+ Learn complex non-linear models effectively (in some applications)
+ Networks can represent arbitrary functions (over a bounded interval)
  (suitable neuron type, high accuracy on a finite interval)
  (function determined by the weights of connections between neurons)
- Require lots of training data (which has to be labelled)
- Result of learning implicit in the neural network (the weights!)

Neural Networks

1943 McCulloch & Pitts model
1958 Rosenblatt’s single-layer perceptron model
1969 Minsky & Papert demonstrate limitations of perceptrons
1986 Hinton’s Backpropagation algorithm for multi-layer networks
2012 Image classification (ImageNet) success for AlexNet
2013- lots of interest in neural networks, also outside academia...
2015 AlphaGo enhances tree search with NNs, beats human champions

Neural Networks

- Feed-forward neural networks (acyclic)
  - Most commonly used
  - Good training algorithms exist
- Recurrent networks (cyclic)
  - More brain-like model
  - Fewer applications (currently)
  - Poorly understood, difficult to train
Neural Networks

Single-layer neural network without hidden nodes

Multi-layer network ("deep learning")

Nodes ("neurons") in Neural Networks

Weighted sum of inputs:

\[ z = w_0 + \sum_{i=1}^{k} x_i w_i \]

Output by activation function \( g(z) \) (next slide)

Types of Neurons

Activation (output) of a neuron:

- Linear: \( y = c \cdot z \)
- Binary threshold: \( y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases} \)
- Rectified linear: \( y = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases} \)
- Sigmoid: \( y = \frac{1}{1+e^{-cz}} \) (\( \lim_{c \to \infty} y \) approaches Binary threshold)

Why Not Just Linear Neurons (Perceptrons)?

Linear inseparability: Even with multiple layers, cannot represent \( x = y \) for Booleans, nor \( x \neq y \) (XOR).

- Led to declined interest in NNs! (Minsky & Papert 1969)
- Solution: Use non-linear activation functions!
Linear

$$y = z$$

Binary Threshold

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Rectified Linear

$$y = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Sigmoid

$$y = \frac{1}{1 + e^{-z}}$$
Steepness of the Logistic Function

The steepness of the jump from \( y = 0 \) to \( y = 1 \) can be controlled by the constant \( c \). This example has \( c = 10 \).

\[
y = \frac{1}{1 + e^{-cz}}
\]

For \( \sigma(z) = \frac{1}{1 + e^{-cz}} \), the derivative \( \frac{d\sigma(z)}{dz} = c\sigma(z)(1 - \sigma(z)) \).

The Logistic Function

sum of two logistic functions

\[
y = \frac{1}{1 + e^{-30z}} - \frac{1}{1 + e^{-(30z-20)}}
\]

The Backpropagation Algorithm

For every training instance do the following.

1. Compute the activation for every node
2. Compute the error, i.e. the difference between the output activation and the desired activation.
3. Propagate the error backwards to all nodes
4. Adjust the weights to reduce the error

The algorithm is the same for all neuron types. We use \( g(x) = \frac{1}{1 + e^{-x}} \) and \( g'(x) = g(x)(1 - g(x)) \) (the logistic function i.e. sigmoid neurons).

Reference: Russell & Norvig, *Introduction to A.I. – A Modern Approach*
The Backpropagation Algorithm: Phase 1

\( L \)  
\( N \)  
\( x_0, \ldots, x_n \)  
\( a_i \) for all \( i \in \{0, \ldots, N\} \)

number of layers  
number of nodes (input nodes + neurons)  
the inputs to the neural network  
the activation level of node \( i \)

Calculate the activation levels of all nodes (forwards):

1. Let \( a_i = x_i \) for all \( i \in \{0, \ldots, n\} \) 

2. For layers \( l \in \{2, \ldots, L\} \), and for each node \( j \) in layer \( l \)
   - \( \text{in}_j := \sum_{i \in \text{pred}(j)} w_{i,j} a_i \)
   - \( a_j := g(\text{in}_j) \)

Here \( \text{pred}(j) \) is the set of predecessors of \( j \) (and \( \text{succ}(j) \) is the set of successors).

The Backpropagation Algorithm: Phase 2

Propagate error to all nodes (backwards); adjust weights

1. For output nodes \( j \) (level \( L \)) do
   - \( \Delta(j) := g'(\text{in}_j) \cdot (y_j - a_j) \) where \( y_j \) is the desired output for \( j \)

2. For each level \( l = L - 1, L - 2, \ldots, 3, 2 \), and each node \( i \) on level \( l \)
   - \( \Delta(i) := g'(\text{in}_i) \sum_{j \in \text{succ}(i)} w_{i,j} \Delta(j) \)
   - This propagates the error backwards through the whole network.

3. Update every weight by \( w_{i,j} := w_{i,j} + \lambda a_i \Delta(j) \) 
   (\( \lambda \) is the learning rate.)

Application of the Backpropagation Algorithm

- Initial weights random (but range should be sensible)
- Weight updates
  - Separately for each training instance (on-line)
  - Gradient for small set of training instances (mini batch) + weight update
  - Gradient for all training instances (full batch) + weight update
  - Possibly: multiple runs with the same training data
- Convergence often slow: millions of training instances
- Fastest implementations with CUDA and GPUs
- Specialized hardware being developed

Properties of the Backpropagation Algorithm

- Runtime is polynomial, in size of the network and the training set
- Backpropagation cannot guarantee finding a global minimum: Different local minima can be reached, depending on initial weights, ordering of training data
- Finding weights that minimize error is NP-hard
  - Finding optimum not practical because of very large size of networks
AlexaNet

- Breakthrough in ImageNet classification competition 2012
  - 1000 image categories
  - 10,000,000 images (labelled), 256 × 256 RGB pixels
- AlexaNet’s network structure:
  - 5 convolutional layers +
  - 3 fully connected layers (4096, 4096 & 1000 nodes, respectively)
  - with 4096·4096 = 16,777,216 and 4096 × 1000 = 4,096,000 connections
  - Total number of connections (weights) is about 60 million
  - The 1000-node layer is the outputs for the 1000 classes
  - Rectified linear nodes (as good as but faster than sigmoid)
- Later winners had more layers: 19 layers in 2014, 152 layers in 2015
- AlexaNet error rate 15.3 per cent, ResNet (2015) 3.57 per cent

Reliability of Image Classification

- Images misclassified as real objects, with high confidence!
- 50 pixels randomly perturbed can lead to misclassification


Nguyen, Yosinski, Clune, Deep Neural Network are Easily Fooled: High Confidence Predictions for Unrecognizable Images, Computer Vision and Pattern Recognition (CVPR’15), IEEE, 2015