Recall: The Agent Function

We can think of the entire agent, or some portion of it as implementing a function.

• inputs: the agent’s internal state and what it perceives
• outputs: the agent’s actions

We have been thinking of this as a function in the programming sense.

Let’s now think of it instead as a function in the mathematical sense.

\[ f \text{ (percept, state)} = \text{command} \]
Agent Function Examples

• state space search (example: traffic jam)
  • input = complete model of the state space
  • output = complete plan of action

• game playing, online planning (example: hex)
  • input = current state
  • output = current action

• Offline planning/learning (example: Pacman)
  • input = current state, history
  • output = current action
Machine Learning Approach

Rather than program a function directly, generalize from data.

- Gather example inputs & outputs.
- Find a function that maps between inputs and outputs effectively.
- Test how well that function generalizes to new examples.
Some examples we’ve already seen:

Q-learning
• Data consists of state/action/next state/reward.
• Learn a mapping from state/action to value.

Approximate Q-learning
• Data consists of state/action/next state/reward.
• Transform state/action into feature vector.
• Learn a linear mapping from feature vector to reward.
Why learning?

Can’t we just program the solution?

• We can’t anticipate all of the possible situations an agent may face.
• We want the agent to adapt to changes in the environment over time.
• We may not know how to solve the problem.
• We may want to model how humans learn.
What function should be learned?

In q-learning, we learn the full agent function.
  • Q-learning updates generate a value function.
  • The value function implies an optimal policy.
  • Once learning is done, the agent function is trivial: for the current state, look up the best action.

AlphaGo learned multiple helper functions:
  • An accurate move-probability distribution for use in the tree policy.
  • A fast-to-evaluate move-probability distribution for use in the default policy.
  • A board-evaluation heuristic.
Smaller units that we could learn.

Instead of learning the whole agent function, we could learn...

• State space representation
  • What features of the world are important for the task?
• Utility function
  • What outcomes are better for the agent?
• State evaluation heuristics
  • What direction seems more promising?

• Other ideas?
What does the data set look like?

• Discrete or continuous?
  • We mostly care about whether the output is continuous.

• Do we know the right answer?
  • supervised
  • semi-supervised
  • unsupervised

• Do we have all the data in advance?
  • online learning

• How noisy is the data?
Supervised Learning: Regression

- Input: x values, continuous y values
- Output: simple function from x to y
Supervised Learning: Classification

• Input: x values, discrete labels
• Output: function to label new points
Unsupervised Learning: Clustering

- Input: unlabeled x values
- Output: breakdown into clusters
Unsupervised Learning: Dimensionality Reduction

- Input: unlabeled x values
- Output: lower-dimensional representation of the data
Semi-Supervised Learning: Reinforcement Learning

- Input: states, occasional utilities
- Output: values/policy
Online Learning

Offline learning: we have all of the data in advance.

Online learning: the data arrives incrementally, and we need to make decisions before we have it all.

- Model must be easy to update with new data.
- We may want to take actions just to gather better data.

Similar (but not identical) to the online/offline planning distinction.
Evaluating Hypotheses

• To measure the accuracy of a learned function, we use a **test set** of examples that are distinct from the **training set**.

• A hypothesis generalizes well if it correctly predicts the output for the novel examples in the test set.

• We prefer hypotheses that generalize well over ones that perform optimally on the training set.
Which function models the data better?

There is often a tradeoff between complex hypotheses that fit the data better and simpler hypotheses that generalize better.
Simple Learning Algorithm: Perceptrons

Inspired by biological neurons.

How a neuron works (extreme basics):
• Connected to other neurons through dendrites.
• Sense the activity of neighboring neurons.
• If neighbors reach some threshold, activate.
• On activation, send an electrical pulse down the axon.
Mathematical Model of a Neuron

• Neuron represented by a node.
• Connections to other neurons represented by edges.
• Each edge has a weight.
• Sum up weighted activation of neighbors.
• Activate if sum is above threshold.

\[
\text{output} = \begin{cases} 
0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\
1 & \text{if } \sum_j w_j x_j > \text{threshold}
\end{cases}
\]
Boolean Functions with Neurons

Perceptrons can represent many boolean functions.

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Exercise: choose weights and threshold to represent AND