Passive Learning
(Ch. 21.1-21.2)

People generally remember...
(learning activities)

- 10% of what they read
- 20% of what they hear
- 30% of what they see
- 50% of what they see and hear
- 70% of what they say and write
- 90% of what they do.

People are able to...
(learning outcomes)

- Read
- Hear
- View Images
- Analyze

Define
List
Describe
Explain
Demonstrate
Apply
Practice
Analyze
Define
Create
Evaluate

Passive Learning

Active Learning

Participate in Hands-On-Workshops
Design/Perform a Presentation - "Do the Real Thing"
Design/Lessons
Simulate, Model, or Experience a Lesson
Watch a Demonstration
Analyzing and Evaluating
Collecting and Organizing
Designing and Preparing
Questioning the Assumptions
Discovering"
Step 1. EM Algorithm

For an example, let’s go back to the original data but convert for hidden:

\[ P(\text{mood}) = 0.5 \]
\[ P(\text{HW} = \text{easy} \mid \text{mood}) = 0.8 \]
\[ P(\text{HW} = \text{easy} \mid \neg \text{mood}) = 0.25 \]

... saw 3 HW: easy, easy, hard

Step 1. is done (initialize parameter guess as the above probabilities)
Step 2. EM Algorithm

Step 2: estimate unknown

In other words we need to find \( P(\text{mood} | \text{data}) \)

In the case where all variables were visible, this would just have been: \[
\frac{\text{[number of positive mood]}}{\text{total}}
\]

However, since we can’t see which ones, we have to estimate using parameters
Step 2. EM Algorithm

If “N” is our total, then we let “\( \hat{N} \)” be our estimate count, where: (Bayes rule)

\[
\hat{N}(\text{mood}) = \sum_i P(\text{mood}|\text{HW}_i) = \sum_i \frac{P(\text{HW}_i|\text{mood}) \cdot P(\text{mood})}{\sum_j P(\text{HW}_i|\text{mood}=j) \cdot P(\text{mood}=j)}
\]

So in our 2 easy, 1 difficult example:

\[
2 \cdot \frac{0.8 \cdot 0.5}{0.8 \cdot 0.5 + 0.25 \cdot 0.5} + 1 \cdot \frac{0.2 \cdot 0.5}{0.2 \cdot 0.5 + 0.75 \cdot 0.5} = 1.7343358396
\]

So our new estimate is 1/N that or:

\[
P(\text{mood}) = \frac{1.7343358396}{3} = 0.57811194653
\]

just Bayes rule:
\[
P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(A,B) + P(\sim A,B)}
\]

... A=mood, B=HW

more easy HW estimate I’m in a good mood
Step 3. EM Algorithm

Step 3: find best parameters

Now that we have $P(\text{mood})$ estimate, we use it to compute table for $P(\text{HW?} \mid \text{mood})$

Again, we have to approximate the number of homework that came from good/bad mood:

$$\hat{N}(\text{HW?} = \text{easy}, \text{mood}) = \sum_{i \in \text{HW?} = \text{easy}} P(\text{mood} \mid HW_i)$$

(same as before, but don’t include “hards”)
So before we used this to calculate the total number of stuff caused by a good “mood”:

$$2 \cdot \frac{0.8 \cdot 0.5}{0.8 \cdot 0.5 + 0.25 \cdot 0.5} + 1 \cdot \frac{0.2 \cdot 0.5}{0.2 \cdot 0.5 + 0.75 \cdot 0.5} = 1.7343358396$$

Now if we want to find a new estimate for number of easy homeworks caused by mood, ignore the hard part.
Step 3. EM Algorithm

So before we used this to calculate the total number of stuff caused by a good “mood”:

$$2 \cdot \frac{0.8 \cdot 0.5}{0.8 \cdot 0.5 + 0.25 \cdot 0.5} + 1 \cdot \frac{0.2 \cdot 0.5}{0.2 \cdot 0.5 + 0.75 \cdot 0.5} = 1.52380952381$$

Now if we want to find a new estimate for number of easy homeworks caused by mood, ignore the hard part
Step 3. EM Algorithm

This means we estimate 1.523 of the “easy” HW came from a good mood

We just estimated that $P(\text{mood}) = 0.5781$, so with 3 examples “mood” happens 1.734 (same number as original sum)

Thus: $\frac{\hat{N}(\text{mood}, HW=easy)}{\hat{N}(\text{mood})} = \frac{1.523}{1.734} = 0.8783$ an increase from our original 0.8 $P(\text{hw=easy|mood})$

like $P(\text{easy|mood}) = P(\text{easy,mood})/P(\text{mood})$
Step 4. EM Algorithm

Then we go off and do a similar equation to get a new estimate for $P(HW=\text{easy} \mid \neg \text{mood})$.

After that, we just iterate the process, so with new value recompute $P(\text{mood})$.

Recompute: $P(HW=\text{easy} \mid \text{mood})$ and $P(HW=\text{easy} \mid \neg \text{mood})$ using new $P(\text{mood})$.

Re-recompute: $P(\text{mood})$...
EM Algorithm

You can also use the EM algorithm on HMMs, but you have to group together all transitions (since they use the same probability)

\[
P(X_{t+1} = j | X_t = i) \sum_t \hat{N}(X_{t+1} = j, X_t = i) / \sum_t \hat{N}(x_t = i)
\]

The EM algorithm is also not limited to just all things Bayesian, and can be generalized:

\[
\theta^{(i+1)} = \arg \max_{\theta} \sum_x P(Z = z | x, \theta^{(i)}) \cdot L(x, Z = z | \theta^{(i)})
\]

step 2. assume parameters, \(\theta\)  
step 3. maximize outcomes
The EM algorithm is a form of gradient descent (or hill-climbing, but no $\alpha$)

Real distribution

Some samples

EM algorithm reverse-eng.
So far we have had labeled outputs for our data (i.e. we knew the homework was easy)

We will move from this (supervised learning) to where we don’t know the correct answer, just if it was good/bad (reinforcement)

This is much more useful in practice as for hard problems we often don’t know the correct answer (else why’d we ask the computer?)
Reinforcement Learning

We will start by looking at **passive learning**, where we will not be taking actions, but just observing outcomes (because easier)

Next time we will move into **active learning**, where we can choose how we want to act to find the best outcomes/learn quickly

For now we want something we can observe, but see outcomes (i.e. rewards) for actions
Reinforcement Learning

To do this, we will go back to our friend MDP.

However since this is passive learning, we will only use the actions/arrows shown.

(T’s are terminal states, so no actions)
Reinforcement Learning

How is this different than before?

1. Rewards of states not known
2. Transition function not known
   (i.e. no 80%, 10%, 10%)

Instead we will see examples of the MDP being run and learn the utilities

<table>
<thead>
<tr>
<th></th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>←</td>
<td></td>
</tr>
<tr>
<td>←</td>
<td></td>
</tr>
<tr>
<td>←</td>
<td></td>
</tr>
<tr>
<td>←</td>
<td></td>
</tr>
<tr>
<td>←</td>
<td></td>
</tr>
<tr>
<td>←</td>
<td></td>
</tr>
</tbody>
</table>
Reinforcement Learning

Suppose we start in bottom row, left-most column and take the path shown

This will be recorded as (state)_{\text{reward}}:

\((4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (4,2)_{-1}\)
\((3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}\)

... then repeat this for more examples to better learn
The first (of three) ways to do passive learning is called direct utility estimation using reward:

\[ U(s_0, s_1, s_2, \ldots) = E \left[ \sum_{i=0}^{\infty} \gamma^i \cdot R(s_i) \right] \]

Given this sequence, we can calculate the rewards at each step (starting from end):

(1,2) has reward 50 - 1 - 1 - 1 - 1 - 1 = 45

Then (2,2) is one more, so 45 + 1 = 46... so on
Direct Utility Estimation

This gives us:

\((4,2)_1 \uparrow (3,2)_1 \rightarrow (4,2)_1 \uparrow (3,2)_1 \rightarrow (2,2)_1 \uparrow (1,2)_{50}\)

\[
\begin{align*}
40 & \quad 41 & \quad 42 & \quad 43 & \quad 44 & \quad 45 \\
\end{align*}
\]

Then we just find the average reward

(4,2) visited twice (40,42)… average = 41

… and so on

(1,2) visited once… average reward = 45

Then update averages with future examples
Direct Utility Estimation

So let’s say you go straight to goal:

\[(4,2) \rightarrow (3,2) \rightarrow (2,2) \rightarrow (1,2)\]

\[
\begin{array}{c}
44 \\
45 \\
46 \\
47 \\
\end{array}
\]

Then we update old averages with new data (only need store counts):

(4,2) visited once (44)... new average = 44

\[
\text{avg}_{\text{total}} = \frac{\text{count}_{\text{old}} \cdot \text{avg}_{\text{old}} + \text{count}_{\text{new}} \cdot \text{avg}_{\text{new}}}{\text{count}_{\text{old}} + \text{count}_{\text{new}}} = \frac{2 \cdot 41 + 44}{3} = 42
\]

(1,2) visited once... new average = 47,

so running total average now \((45+47)/2=46\)
Direct Utility Estimation

Given that we are sampling the actions, this should lead to the correct expected rewards just by simple average

(This also has changed problem back to supervised, as we “see” outcomes of actions)

But we can speed this up (i.e. learn much faster) by using some information What info have we not used?
Adaptive Dynamic Prog.

We didn’t include our bud Bellman!

\[ U(s) = R(s) + \gamma \cdot \sum_{s'} P(s'|a, s) \cdot U(s') \]

Thus, if we can learn the rewards and transitions, we can use our normal ways of solving MDPs (value/policy iteration).

This is useful as we can combine information across different states for faster learning.
Adaptive Dynamic Prog.

So given the same first example:

\((4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}\)

We’d estimate the following transitions:

\((4,2) + \uparrow = 100\% \uparrow (2 \text{ of } 2)\)
\((3,2) + \rightarrow = 50\% \uparrow, 50\% \downarrow\)
\((2,2) + \uparrow = 100\% \uparrow\)

... and we can easily see the rewards from sequence, so policy/value iteration time!

better as actions fixed no iteration
Adaptive Dynamic Prog.

This method is called adaptive dynamic programming.

Using the relationship between utilities (i.e. neighbors cannot change too much) allows us to learn quicker.

This can be sped up even more if we assume all actions have the same outcome (i.e. going “up” has same probability for any state).
The third (last) way of doing passive learning is **temporal-difference** learning.

This is a combination of the first two methods, we will keep a “running average” of each state’s utility, but also use Bellman equation.

Instead of directly averaging rewards to find utility, we will incrementally adjust them using the Bellman equation.
Temporal-Difference

Suppose we saw this example (bit different):

\[(4,2) \uparrow (3,2) \rightarrow (2,2) \uparrow (3,2) \rightarrow (2,2) \uparrow (1,2)\]

Using the direct averaging we would get:

\[U(4,2) = 40, \quad U(3,2) = 42\]

However the sample(s) so far: \((4,2) \uparrow\) is always \((3,2)\), so we’d expect (from Bellman):

\[U(4,2) = -1 + U(3,2)\]
Temporal-Difference

This would indicate our guess of $U(4,2)=40$ is a bit low (or $U(3,2)$ is a bit high)

So instead of direct average, we will do incremental adjustments using Bellman:

$$U(s) \leftarrow U(s) + \alpha \cdot (R(s) + \gamma \cdot U(s') - U(s))$$

learning rate/constant

So whenever you take an action, you update the utility of the state before the action (final terminal state does not need updating)
Let’s continue our example:
\[(4,2) \uparrow (3,2) \uparrow (2,2) \uparrow (3,2) \uparrow (2,2) \uparrow (1,2)\]

So from first example: \(U(4,2) = 40, U(3,2) = 42\)
If second example starts as:
\[(4,2) \uparrow (3,2) \uparrow \cdots\]

We’d update \((4,2)\) as: (assume \(\alpha = 0.5\))
\[
U(4,2) \leftarrow U(4,2) + \alpha \cdot (R(4,2) + \gamma \cdot U(3,2) - U(4,2))
\]
\[
\leftarrow 40 + \alpha(-1 + 1 \cdot 42 - 40)
\]
\[
\leftarrow 40.5
\]
Recap: Passive Learning

What are pros/cons between the last two methods? (adapt. dyn. prog. vs temporal-diff.)

Which do you think is faster at learning in general?
Recap: Passive Learning

What are pros/cons between the last two methods? (adapt. dyn. prog. vs temporal-diff.)
- Temporal-difference only changes a single value for each action seen
- ADP would re-solve a system of linear equations (policy “iteration”) for each action

Which do you think is faster at learning in general?

As ADP uses Bellman equations/constraints in full it learns better (but more computation)
Recap: Passive Learning

From the book’s example:

- ADP
- TD