Final Exam  Mon, May 6, 9:30–10:45

Covers all lectures after the first exam.
Same format as the first exam.

Location (if you’re in Prof. Hockenmaier’s sections)
Materials Science and Engineering Building, Room 100 (http://ada.fs.illinois.edu/0034.html)

Conflict exam: Wed, May 8, 9:30–10:45
Location: Siebel 3403.

If you need to take your exam at DRES, make sure to notify DRES in advance
CS 440/ECE448 Lecture 19: Bayes Net Inference

Including slides by Svetlana Lazebnik, 11/2016
Parameter learning

- **Inference problem**: given values of evidence variables $E = e$, answer questions about query variables $X$ using the posterior $P(X \mid E = e)$

- **Learning problem**: estimate the parameters of the probabilistic model $P(X \mid E)$ given a *training sample* $\{(x_1,e_1), \ldots, (x_n,e_n)\}$

- **Learning from complete observations**: relative frequency estimates

- **Learning from data with missing observations**: EM algorithm
Missing data: the EM algorithm

• The EM algorithm starts ("Expectation Maximization") starts with an initial guess for each parameter value.

• We try to improve the initial guess, using the algorithm on the next two slides:
  • E-step
  • M-step

<table>
<thead>
<tr>
<th>Training set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>…</td>
</tr>
</tbody>
</table>
Missing data: the EM algorithm

- **E-Step (Expectation):** Given the model parameters, replace each of the missing numbers with a probability (a number between 0 and 1) using

\[
P(C = 1|S, R, W) = \frac{P(C = 1, S, R, W)}{P(C = 1, S, R, W) + P(C = 0, S, R, W)}
\]

**Example:**

Training set

<table>
<thead>
<tr>
<th>Sample</th>
<th>C</th>
<th>S</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>0.5?</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>0.5?</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Missing data: the EM algorithm

• M-Step (Maximization): Given the missing data estimates, replace each of the missing model parameters using

\[
P(\text{Variable} = T|\text{Parents} = \text{value}) = \frac{E[\# \text{ times Variable} = T, \text{Parents} = \text{value}]}{E[\# \text{times Parents} = \text{value}]}\]

<table>
<thead>
<tr>
<th>Sample</th>
<th>C</th>
<th>S</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>0.5?</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>0.5?</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
CS440/ECE448 Lecture 20: Hidden Markov Models

Hidden Markov Model = [ i, j, k, ... ]

State Sequence Q = [ i, i, i, j, j, k, k, k, ... ]

Observations O = [ o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9 ]
Hidden Markov Models

• At each time slice $t$, the state of the world is described by an unobservable (hidden) variable $X_t$ and an observable evidence variable $E_t$

• **Transition model:** The current state is conditionally independent of all the other states given the state in the previous time step
  
  **Markov assumption:** $P(X_t \mid X_0, \ldots, X_{t-1}) := P(X_t \mid X_{t-1})$

• **Observation model:** The evidence at time $t$ depends only on the state at time $t$
  
  **Markov assumption:** $P(E_t \mid X_{0:t}, E_{1:t-1}) = P(E_t \mid X_t)$
Example

Transition model

<table>
<thead>
<tr>
<th>$R_{t-1}$</th>
<th>$P(R_t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0.7</td>
</tr>
<tr>
<td>$f$</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Observation model

<table>
<thead>
<tr>
<th>$R_t$</th>
<th>$P(U_t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0.9</td>
</tr>
<tr>
<td>$f$</td>
<td>0.2</td>
</tr>
</tbody>
</table>

state

$Rain_{t-1} ightarrow Rain_t ightarrow Rain_{t+1}$

evidence

$Umbrella_{t-1} ightarrow Umbrella_t ightarrow Umbrella_{t+1}$
An alternative visualization

Transition probabilities

<table>
<thead>
<tr>
<th>( R_{t-1} )</th>
<th>( R_t = T )</th>
<th>( R_t = F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_{t-1} = T )</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>( R_{t-1} = F )</td>
<td>0.3</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Observation (emission) probabilities

<table>
<thead>
<tr>
<th>( R_t )</th>
<th>( U_t = T )</th>
<th>( U_t = F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_t = T )</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>( R_t = F )</td>
<td>0.2</td>
<td>0.8</td>
</tr>
</tbody>
</table>

U=T: 0.9, U=F: 0.1
U=T: 0.2, U=F: 0.8
HMM Learning and Inference

• Inference tasks
  • **Filtering**: what is the distribution over the current state $X_t$ given all the evidence so far, $e_{1:t}$
  • **Smoothing**: what is the distribution of some state $X_k$ given the entire observation sequence $e_{1:t}$?
  • **Evaluation**: compute the probability of a given observation sequence $e_{1:t}$
  • **Decoding**: what is the most likely state sequence $X_{0:t}$ given the observation sequence $e_{1:t}$?

• Learning
  • Given a training sample of sequences, learn the model parameters (transition and emission probabilities)
    • EM algorithm
Markov Decision Processes (MDPs)

• Components that define the MDP. Depending on the problem statement, you either know these, or you learn them from data:
  • **States** $s$, beginning with initial state $s_0$
  • **Actions** $a$
    • Each state $s$ has actions $A(s)$ available from it
  • **Transition model** $P(s' | s, a)$
    • *Markov assumption*: the probability of going to $s'$ from $s$ depends only on $s$ and $a$ and not on any other past actions or states
  • **Reward function** $R(s)$
  • **Policy** – the “solution” to the MDP:
    • $\pi(s) \in A(s)$: the action that an agent takes in any given state
Maximizing expected utility

• The **optimal policy** $\pi(s)$ should **maximize the expected utility** over all possible state sequences produced by following that policy:

$$\sum_{\text{state sequences} \text{ starting from } s_0} P(\text{sequence}|s_0 = \pi(s_0))U(\text{sequence})$$

• How to define the **utility of a state sequence**?
  • **Sum of rewards** of individual states
  • **Problem**: infinite state sequences
  • **Solution**: discount individual state rewards by a factor $\gamma$ between 0 and 1:

$$U([s_0, s_1, s_2, ...]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...$$

$$= \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{\max}}{1 - \gamma} \quad (0 < \gamma < 1)$$
Utilities of states

• Expected utility obtained by policy $\pi$ starting in state $s$:

$$U^\pi(s) = \sum_{\text{state sequences starting from } s} P(\text{sequence}|s, a = \pi(s))U(\text{sequence})$$

• The “true” utility of a state, denoted $U(s)$, is the best possible expected sum of discounted rewards
  • if the agent executes the best possible policy starting in state $s$

• Reminiscent of minimax values of states...
Finding the utilities of states

- If state $s'$ has utility $U(s')$, then what is the expected utility of taking action $a$ in state $s$?
  \[ \sum_{s'} P(s'| s, a) U(s') \]

- How do we choose the optimal action?
  \[ \pi^*(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'| s, a) U(s') \]

- What is the recursive expression for $U(s)$ in terms of the utilities of its successor states?
  \[ U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'| s, a) U(s') \]
The Bellman equation

• Recursive relationship between the utilities of successive states:

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

• For \( N \) states, we get \( N \) equations in \( N \) unknowns
  • Solving them solves the MDP
  • Nonlinear equations -> no closed-form solution, need to use an iterative solution method (is there a globally optimum solution?)
  • We could try to solve them through expectiminimax search, but that would run into trouble with infinite sequences
  • Instead, we solve them algebraically
  • Two methods: **value iteration** and **policy iteration**
Method 1: Value iteration

• Start out with every $U(s) = 0$

• Iterate until convergence
  • During the $i$th iteration, update the utility of each state according to this rule:
    $$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U_i(s')$$

• In the limit of infinitely many iterations, this is guaranteed to find the correct utility values
  • Error decreases exponentially, so in practice, don’t need an infinite number of iterations...
Method 2: Policy iteration

• Start with some initial policy $\pi_0$ and alternate between the following steps:
  • Policy evaluation: calculate $U^{\pi_i}(s)$ for every state $s$
  • Policy improvement: calculate a new policy $\pi_{i+1}$ based on the updated utilities

• Notice it’s kind of like hill-climbing in the N-queens problem.
  • Policy evaluation: Find ways in which the current policy is suboptimal
  • Policy improvement: Fix those problems

• Unlike Value Iteration, this is guaranteed to converge in a finite number of steps, as long as the state space and action set are both finite.
Method 2, Step 1: Policy evaluation

• Given a fixed policy $\pi$, calculate $U^\pi(s)$ for every state $s$

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^\pi(s')$$

• $\pi(s)$ is fixed, therefore $P(s'|s, \pi(s))$ is an $s' \times s$ matrix, therefore we can solve a linear equation to get $U^\pi(s)$!

• Why is this “Policy Evaluation” formula so much easier to solve than the original Bellman equation?

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$$
CS 440/ECE448 Lecture 22: Reinforcement Learning

Slides by Svetlana Lazebnik, 11/2016
Modified by Mark Hasegawa-Johnson, 4/2019

By Nicolas P. Rougier - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=29327040
Reinforcement learning strategies

• **Model-based**
  • Learn the **model** of the MDP (transition probabilities and rewards) and try to **solve the MDP** concurrently

• **Model-free**
  • **Learn how to act** *without* explicitly learning the transition probabilities $P(s' | s, a)$
  • **Q-learning**: learn an **action-utility function** $Q(s,a)$ that tells us the value of doing action $a$ in state $s$
Model-based reinforcement learning

- **Basic idea:**
  Try to **learn the model** of the MDP (transition probabilities and rewards) and **learn how to act** (solve the MDP) simultaneously

- **Learning the model:**
  - Keep track of how many times **state s’ follows state s when you take action a**
  - **Update the transition probability** \( P(s' | s, a) \)
    according to these relative frequencies
  - Keep track of the rewards \( R(s) \)

- **Learning how to act:**
  - **Estimate the utilities** \( U(s) \) using Bellman’s equations
  - Choose the **action that maximizes expected future utility:**
    \[
    \pi^* (s) = \arg \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')
    \]
Exploration vs. exploitation

• **Exploration**: take a **new action** with **unknown consequences**
  - Pros:
    - Get a more accurate model of the environment
    - Discover higher-reward states than the ones found so far
  - Cons:
    - When you’re exploring, you’re not maximizing your utility
    - Something bad might happen

• **Exploitation**: go with the **best strategy found so far**
  - Pros:
    - Maximize reward as reflected in the current utility estimates
    - Avoid bad stuff
  - Cons:
    - Might also prevent you from discovering the true optimal strategy
Incorporating exploration

- **Idea:** explore more in the beginning, become more and more greedy over time

- **Standard ("greedy") selection of optimal action:**
  \[ a = \arg \max_{a' \in A(s)} \sum_{s'} P(s' | s, a') U(s') \]

- **Modified strategy** with exploration function \( f(u,n) \)
  \( f(u,n) \) trades off **greed** [preference for high utility \( u \)] against **curiosity** [preference for low observed frequencies \( n \)]

\[
f(u,n) = \begin{cases} 
R^+ & \text{if } n < N_e \\
u & \text{otherwise}
\end{cases}
\]

- Set utility of \( a' \) to \( R^+ \) [= optimistic reward estimate] if \( a' \) in state \( s \) explored less than \( N_e \) [a constant] times
- Set utility to actual observed utility

\[
a = \arg \max_{a' \in A(s)} f \left( \sum_{s'} P(s' | s, a') U(s'), N(s, a') \right)
\]

- Exploration function
- Number of times we’ve taken action \( a' \) in state \( s \)
Model-free reinforcement learning

• **Idea:** learn how to act *without* explicitly learning the transition probabilities $P(s' | s, a)$

• **Q-learning:** learn an *action-utility function* $Q(s,a)$ that tells us the value of doing action $a$ in state $s$

• Relationship between Q-values and utilities:

$$U(s) = \max_a Q(s,a)$$

• Selecting an action: $\pi^*(s) = \arg\max_a Q(s,a)$

• Compare with: $\pi^*(s) = \arg\max_a \sum_{s'} P(s'| s, a) U(s')$

  • With Q-values, don’t need to know the transition model to select the next action
Temporal difference (TD) learning

- **Equilibrium constraint** on Q values:
  \[ Q(s, a) = R(s) + \gamma \sum P(s'| s, a) \max_{a'} Q(s', a') \]

- **Temporal difference (TD) update:**
  - Pretend that the currently observed transition \((s,a,s')\) is the *only* possible possible outcome.
  - Call this “local quality” as \(Q_{local}(s, a)\);
  - it is computed using \(Q(s, a)\).
  \[ Q_{local}(s, a) = R(s) + \gamma \max_{a'} Q(s', a') \]
  - Then interpolate between \(Q(s, a)\) and \(Q_{local}(s, a)\) to compute \(Q_{new}(s, a)\).
  \[ Q_{new}(s, a) = (1 - \alpha)Q(s, a) + \alpha Q_{local}(s, a) \]
Function approximation

• So far, we’ve assumed a lookup table representation for utility function $U(s)$ or action-utility function $Q(s,a)$

• But what if the state space is really large or continuous?

• Alternative idea: approximate the utility function, e.g., as a weighted linear combination of features:

\[
U(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots w_n f_n(s)
\]

• RL algorithms can be modified to estimate these weights
• More generally, functions can be nonlinear (e.g., neural networks)

• Recall: features for designing evaluation functions in games

• Benefits:
  • Can handle very large state spaces (games), continuous state spaces (robot control)
  • Can generalize to previously unseen states
Notation

Usually we have two databases:

• A training database consists of \( N \) different training tokens (one token = one image, or sentence, or speech files, or whatever). We write them as vectors, \( \hat{X}_i = [X_{i1}, \ldots, X_{iM}] \), for \( 1 \leq i \leq N \). Each one has an associated reference (ground truth) label \( Y_i \).

• A testing database contains only the test tokens \( \hat{X}_i \), for \( N + 1 \leq i \)
For both training and testing, we have to present the token $\vec{X}_i$ to the input of the neural net, and then the neural net computes some output $\vec{F}_i$. 

Notation
A deep neural net has thousands of neurons (nodes).

Each neuron (node) has two key variables:

- The "affine", $Z_{ij}$, models the synapse of a biological neuron, collecting information from a lot of other neurons:
  \[ Z_{ij} = \sum_k A_{ik} W_{kj} \]

- The "activation," $A_{ij}$, models the axon of a biological neuron i.e., it’s zero when the input is negative, and nonzero when the input is positive:
  \[ A_{ij} = g(Z_{ij}) \]
Notation for a Neural Net without Layers

- $A_{ij}$ is the $j^{th}$ activation for the $i^{th}$ token:
  - Some of the activations are provided by the input, i.e., $A_{ij} = X_{ij}$ for some of the $j$’s.
  - Some of the activations are outputs, i.e., $F_{ij} = A_{ij}$ for some of the $j$’s.
  - Some of the activations are neither inputs nor outputs. Those are called “hidden nodes.”
  - Which ones are inputs, hidden, and outputs? Well, it depends on the particular neural network design, there’s no way to know, in general.

- $Z_{ij}$ is the $j^{th}$ affine for the $i^{th}$ token

- $W_{kj}$ is the $(k, j)^{th}$ weight.
Notation for a Neural Net with Layers

• $A^{(l)}_{ij}$ is the $j^{th}$ activation in the $l^{th}$ layer for the $i^{th}$ token:
  • The $0^{th}$ layer is the input, i.e., $A^{(0)}_{ij} = X_{ij}$.
  • The $L^{th}$ layer is the output, i.e., $F_{ij} = A^{(L)}_{ij}$.
  • All other layers are “hidden layers.”

• $Z^{(l)}_{ij}$ is the $j^{th}$ affine in the $l^{th}$ layer for the $i^{th}$ token

• $W^{(l)}_{kj}$ is the $(k, j)^{th}$ weight in the $l^{th}$ layer.

$$Z^{(l)}_{ij} = \sum_k A^{(l-1)}_{ik} W^{(l)}_{kj}$$
Forward Propagation (Using the Neural Net)

• We use a neural net by presenting a token $\vec{X}_i$, and computing the output $\vec{F}_i$.

• This is done by setting:
  • $A_{ij}^{(0)} = X_{ij}$
  • For $1 \leq l \leq L$:
    • $Z_{ij}^{(l)} = \sum_k A_{ik}^{(l-1)} W_{kj}^{(l)}$
    • $A_{ij}^{(l)} = g(Z_{ij}^{(l)})$
    • $F_{ij} = A_{ij}^{(L)}$

• This algorithm is called "forward propagation," because information propagates forward through the network, from the $0^{th}$ layer to the $L^{th}$ layer.
How well did it do?

• We test a neural net by computing $\hat{F}_i$ from $\hat{X}_i$, for each of the tokens $1 \leq i \leq N$, and then comparing the network output to the reference (ground truth) answer, $Y_i$.
  • During training: we measure error using training data, and try to train the network in order to reduce the error rate.
  • During ”development test:” we compare different networks on the development test data.
  • During “evaluation test:” our customer tests our network with data it’s never seen before.

• But... How do we compare $\hat{F}_i$ to $Y_i$? i.e., how we define “error” or “loss”? 
Regression problems: Sum-squared error

- For example, suppose that the network output is an image.
- An image is a vector, $\vec{F}_i = [F_{i1}, \ldots, F_{iM}]$
- The “right answer” is the image we were trying to reconstruct, $\vec{Y}_i = [Y_{i1}, \ldots, Y_{iM}]$.
- Then a reasonable loss function is sum-squared error (SSE):

$$L_{SSE} = \sum_{i=1}^{N} \sum_{j=1}^{M} (Y_{ij} - F_{ij})^2$$
Classifier problems: Cross-entropy

- On the other hand, for this course, we usually want $Y_i$ to be some category label, for example, $Y_i = \text{“chickens”}$.
- In that case, we can use a special kind of nonlinearity at the output of our neural network, called a softmax, that gives a probabilistic interpretation to the network outputs:
  $$F_{ij} = P(Y_i = j^{th} \text{ type of category})$$
- Then a reasonable loss function is the log probability of the correct class:
  $$L_{CE} = -\sum_{i=1}^{N} \ln F_{i,Y_i}$$
- This error criterion is called “cross entropy” for reasons that are fascinating but way beyond the scope of this course.
Classifier output: Softmax

• We want $Y_i$ to be some category label, for example, $Y_i = “\text{boots}”$.

• In that case, we want $F_{ij}$ to meet the criteria for a probability, i.e., we need $F_{ij} \geq 0$ and $\sum_j F_{ij} = 1$.

• In order to do that, we use a special kind of nonlinearity in the last layer of the neural net, called a softmax:

$$F_{ij} = \frac{e^{Z_{ij}^{(L)}}}{\sum_k e^{Z_{ik}^{(L)}}}$$
Training the Neural Net

A neural net is trained according to gradient descent:

\[ W_{jk}^{(l)} = W_{jk}^{(l)} - \eta \frac{\partial L}{\partial W_{jk}^{(l)}} \]

So that the loss function, L, gradually approaches a local minimum.
Training the Neural Net: Notation

• Let’s use the following shorthand:

\[ \delta(\text{Variable}) = \frac{\partial L}{\partial (\text{Variable})} \]

For example:

\[ \delta W_{kj}^{(l)} = \frac{\partial L}{\partial W_{kj}^{(l)}} \]
Training the Neural Net: Last Layer

The cross entropy loss is:

\[ L_{CE} = - \sum_{i=1}^{N} \ln F_{i,Y_i} \]

Its derivative is:

\[ \delta Z_{ij}^{(L)} = \begin{cases} F_{ij} - 1 & j = Y_i \\ F_{ij} & j \neq Y_i \end{cases} \]

Here’s how to remember that:

- If \( j \) is the right answer, then error is minimized \( (\delta Z_{ij}^{(L)} = 0) \) when \( F_{ij} = 1 \).

- If \( j \) is the wrong answer, then error is minimized \( (\delta Z_{ij}^{(L)} = 0) \) when \( F_{ij} = 0 \).
Convolution versus Matrix Multiplication

A regular neural net uses a matrix multiplication in each layer:

\[
Z_{ij}^{(l)} = \sum_k A_{ik}^{(l-1)} W_{kj}^{(l)}
\]

A convolutional neural net uses a convolution at each layer:

\[
\vec{Z}_i^{(l)} = W^{(l)} \cdot \vec{A}_i^{(l-1)}
\]

\[
\vec{Z}_i^{(l)} = W^{(l)} * \vec{A}_i^{(l-1)}
\]
Convolution with Many Channels

Usually, we want the convolutional network to compute many different channels, \( c \):

\[
Z_{i,j,c}^{(l)} = \sum_k A_{i,k}^{(l-1)} W_{j-k,c}^{(l)}
\]

Each of the channels is computing a different type of feature (average, edge, etc.).
Each pixel, in each output channel, tells the degree to which that channel exists at that location in the image.
Deep Reinforcement Learning

CS440/ECE448 Lecture 24

Slides by Svetlana Lazebnik, 11/2017
Modified by Mark Hasegawa-Johnson, 4/2019

Image: Megajuice, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741
Deep Q learning

- Regular TD update: “nudge” Q(s,a) towards the target

\[ Q(s, a) \leftarrow Q(s, a) + \alpha \left( R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \]

- Deep Q learning: encourage estimate to match the target by minimizing squared error:

\[ L(w) = \left( R(s) + \gamma \max_{a'} Q(s', a'; w) - Q(s, a; w) \right)^2 \]

- Compare to supervised learning:

\[ L(w) = (y - f(x; w))^2 \]

- **Key difference**: the target in Q learning is not fixed – (s’,a’) is just one step ahead of (s,a)!
Online Q learning algorithm

• In state $s$, perform action $a$. Environment sends you to state $s'$; choose the action $a'$ that you’ll perform there.
• Observe: $Q^{local}(s, a) = R(s) + \gamma \max_{a'} Q(s', a'; W)$
• Update weights to reduce the error
  $$L(W) = (Q^{local} - Q(s, a; W))^2$$
• Gradient:
  $$\nabla_W L = (Q(s, a; W) - Q^{local}) \nabla_W Q$$
• Weight update:
  $$W \leftarrow W - \eta \nabla_W L$$
• This is called stochastic gradient descent (SGD)
• “Stochastic” because the training sample $(s,a,s',a')$ was chosen at random by our exploration function
Does Q-learning Converge?

• No!

• Because:

\[ a = \text{argmax } Q(s, a) \]

• If we always choose the action that is best, according to our current estimate of the Q-function, then we can never learn anything about any of the other actions!
Incorporating exploration (slide from last week)

• **Idea:** explore more in the beginning, become more and more greedy over time

• Standard ("greedy") selection of optimal action:

\[
a = \arg \max_{a' \in A(s)} \sum_{s'} P(s' | s, a') U(s')
\]

• Modified strategy:

\[
a = \arg \max_{a' \in A(s)} f \left( \sum_{s'} P(s' | s, a') U(s'), N(s, a') \right)
\]

- **exploration function**
- **Number of times we’ve taken action** \(a'\) **in state**

\[
f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}
\]

(optimistic reward estimate)
...but that doesn’t work either:

\[ f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases} \]

- ... which means that we get at least \( N_e \) samples of each action
- We can estimate \( Q(s,a) \) based on \( N_e \) samples
- But \( N_e \) is a constant, so it never \( \rightarrow \infty \)
- So Error never \( \rightarrow 0 \)
Policy gradient methods

• Learning the policy directly can be much simpler than learning Q values
• We can train a neural network to output stochastic policies, or probabilities of taking each action in a given state
• Softmax policy:

\[
\pi(s, a; u) = \frac{\exp(f(s, a; u))}{\sum_{a'} \exp(f(s, a'; u))}
\]
Policy gradient methods

• Learning the policy directly can be much simpler than learning Q values
• We can train a neural network to output *stochastic policies*, or probabilities of taking each action in a given state
• *Softmax* policy:

\[
\pi(s, a; u) = \frac{\exp(f(s, a; u))}{\sum_{a'} \exp(f(s, a'; u))}
\]
Policy gradient: the softmax function

- Notice that the softmax is normalized so that

\[ \pi(s, a; u) \geq 0, \text{ and } \sum_a \pi(s, a; u) = 1 \]

- So we can interpret \( \pi(s, a; w) \) as some kind of probability. Something like “the probability that \( a \) is the best action to take from state \( s \).”

- In reality, there is no such probability. There is just one correct action. But the agent doesn’t know what it is! So \( \pi(s, a; u) \) is kind of like the agent’s “degree of belief” that \( a \) is the best action (determined by parameters \( u \)).
Actor-critic algorithm

- Remember the relationship between the utility of a state, and the quality of an action:
  \[ U(s) = \max_a Q(s, a) \]

- If we don’t know which action is best, then we could say that
  \[ U(s) \approx \sum_a \pi(s, a; u)Q(s, a; w) \]

- \( \pi(s, a; u) \) is the “actor:” a neural net that tells the agent how to act.
- \( Q(s, a; w) \) is the “critic:” a neural net that tells the agent how good or bad that action was.
Actor-critic algorithm

- Define objective function as total discounted reward:

\[ J(u) = E \left[ R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots \right] \]

- The gradient for a stochastic policy is given by

\[ \nabla_u J = E \left[ \nabla_u \log \pi(s, a; u) Q^\pi(s, a; w) \right] \]

- Actor network update:

\[ u \leftarrow u + \alpha \nabla_u J \]

- Critic network update: use Q-learning (following actor’s policy)
Neural Language Models

A neural LM defines a distribution over the V words in the vocabulary, conditioned on the preceding words.

- **Output layer**: V units (one per word in the vocabulary) with softmax to get a distribution
- **Input**: Represent each preceding word by its d-dimensional embedding.
  - **Fixed-length history** (n-gram): use preceding n−1 words
  - **Variable-length history**: use a recurrent neural net
Recurrent neural networks (RNNs)

**Basic RNN:** Modify the standard feedforward architecture (which predicts a string $w_0...w_n$ one word at a time) such that the output of the current step ($w_i$) is given as additional input to the next time step (when predicting the output for $w_{i+1}$).

- “Output” — typically (the last) hidden layer.

![Diagram of Feedforward and Recurrent Nets](image.png)
Basic RNNs

Each time step corresponds to a feedforward net where the hidden layer gets its input not just from the layer below but also from the activations of the hidden layer at the previous time step.
CS440/ECE448 Lecture 26: Speech

Mark Hasegawa-Johnson, 4/17/2019, CC-By 3.0
A Sequence Model you Know: HMM
You’ve seen this slide before, in lecture 20, on HMMs...

• **Markov assumption for state transitions**
  • The current state is conditionally independent of all the other states given the state in the previous time step
  \[ P(Q_t \mid Q_{0:t-1}) = P(Q_t \mid Q_{t-1}) \]

• **Markov assumption for observations**
  • The evidence at time $t$ depends only on the state at time $t$
  \[ P(E_t \mid Q_{0:t}, E_{1:t-1}) = P(E_t \mid Q_t) \]
The Problem of Continuous Observations

• But what about the likelihood? How can we model $P(E_t | Q_t)$?

• The big problem: $E_t$ is continuous, not discrete, so we can’t model $P(E_t | Q_t)$ using a lookup table!
Solutions to the Problem of Continuous Observations

Most systems model $P(E|Q)$ using one of these three standard methods:

1. Use a parameterized probability density, such as a Gaussian. In this case you learn senone-dependent parameters ($\mu_Q$ and $\sigma_Q^2$).

2. Quantize E (using vector quantization) to one of K different code vectors. Then you can learn the lookup table $P_W(E = k|Q)$ for $1 \leq k \leq K$.

3. Use a neural net with a softmax output to compute $P(Q|E)$, then use Bayes’ rule to get $P(E|Q)$ from $P(Q|E)$.
Classifier output: Softmax

You’ve seen this slide before, in lecture 24, on Deep Learning....

• We want $Q_t$ to be a senone, for example, $Q_t = “the jth type of phoneme αi”.  
• In that case, we can force the neural net to learn want the neural net to compute a probability,

$$F_j = P(Q = j | E)$$

...if we just force $F_j$ to meet the criteria for a probability, i.e., we need

$$F_j \geq 0, \quad \sum_j F_j = 1$$

• In order to do that, we use a special kind of nonlinearity in the last layer of the neural net, called a softmax:

$$F_j = \frac{e^{Z_j}}{\sum_k e^{Z_k}}$$
Hybrid DNN-HMM: the problem

• The softmax computes $P(Q|E)$
• The HMM needs to know $P(E|Q)$
• How can we get $P(E|Q)$ from $P(Q|E)$?
• Answer: Bayes’ rule!
Estimating $p(E|Q)$ from $p(Q|E)$

Bayes rule:

$$P(E|Q) = \frac{P(Q|E)P(E)}{P(Q)}$$

... but notice, if our goal is to find the best possible state sequence $Q_1, ..., Q_T$, then we don’t care about the $P(E)$ factor:

$$\arg\max_Q P(E|Q) = \arg\max_Q \frac{P(Q|E)}{P(Q)}$$
Hybrid DNN-HMM: the solution

\[ P(E_1, E_2, Q_1, Q_2, \ldots | W) = P_W(Q_1 | Q_0)P(E_1 | Q_1)P_W(Q_2 | Q_1)P(E_2 | Q_2) \ldots \]

From the neural net

\[ \propto P_W(Q_1 | Q_0) \left( \frac{P(Q_1 | E_1)}{P(Q_1)} \right) P_W(Q_2 | Q_1) \left( \frac{P(Q_2 | E_2)}{P(Q_2)} \right) \ldots \]

HMM Parameters
Hybrid DNN-HMM: intuitive explanation

- Prior probability, p(Q), tells how frequently HMM state Q is, in normal conversations, *if we don’t hear the speech*
- DNN computes a posterior probability, p(Q|E), saying how probable Q is *given the available evidence*
- If p(Q|E) > p(Q), that means that the evidence favors Q more than usual, so we should consider the possibility that this rare word has been spoken.
- If p(Q|E) is still a small number, that doesn’t really matter; what really matters is whether p(Q|E) > p(Q)
CS440/ECE448 Lecture 27: Societal Impacts of AI

Image source: https://www.britac.ac.uk/audio/machines-morality-and-future-medical-care

Slides by Svetlana Lazebnik, 12/2017
Modified by Mark Hasegawa-Johnson, 4/2019
AI and privacy

• Concerns
  • Personal data being inadvertently revealed or falling into the wrong hands
  • Personal data being misused by the parties who collected it
  • Personal data enabling individuals to be manipulated without their knowledge

• Potential solutions
  • Technological: encryption, differential confidentiality, anonymizing tools
  • Regulation: require the use of a technology; forbid disclosure
AI, bias, and fairness

• Concerns
  • AI will inadvertently absorb biases from data
  • Making important decisions based on biased data will exacerbate bias: especially for law enforcement, employment, loans, health insurance, etc.
  • Even well-intentioned applications can create negative side effects: filter bubbles, targeted advertising
  • Outcomes cannot be appealed because AI systems are opaque and proprietary

• Potential solutions
  • Regulation and transparency: e.g., right to explanation
  • More inclusivity among AI technologists: AI4ALL
AI ethics

• We should be aware of all these issues when developing AI technologies!
  • Privacy violations
  • Potential for deception, misuse and manipulation
  • Exacerbating bias and unfair outcomes
  • Lack of transparency and due process
  • Threats to human rights and dignity
  • Weaponization
  • Unintended consequences