Projects

- Project descriptions due today!
Last class

- Sequence to sequence models
- Attention
- Pointer networks
Today

• Weak supervision

• Hashlamot semantic parsing
Weak supervision
Weak supervision

• We have assumed that we have as input pairs of natural language and logical form

• In practice those are hard to collect and we usually have (language, denotation) pairs

<table>
<thead>
<tr>
<th>Heavy supervision</th>
<th>Light supervision</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>How tall is Lebron James?</em></td>
<td><em>How tall is Lebron James?</em></td>
</tr>
<tr>
<td><em>HeightOf.Lebron.James</em></td>
<td>203cm</td>
</tr>
<tr>
<td><em>What is Steph Curry's daughter called?</em></td>
<td><em>What is Steph Curry's daughter called?</em></td>
</tr>
<tr>
<td><em>ChildrenOf.Steph.Curry ⊓ Gender.Female</em></td>
<td><em>Riley Curry</em></td>
</tr>
<tr>
<td><em>Youngest player of the Cavaliers</em></td>
<td><em>Youngest player of the Cavaliers</em></td>
</tr>
<tr>
<td><em>arg min(PlayerOf.Cavaliers, BirthDateOf)</em></td>
<td><em>Kyrie Irving</em></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The problem

- Before we trained with cross entropy over tokens, but we don’t have tokens here.

*largest city in US*

NYC

\( \text{t} \) \( \rightarrow \) softmax \( \rightarrow \) Type (0.7) Profession (0.2) \( \rightarrow \) argmax \( \rightarrow \) Type

\( \text{t+1} \)
This looks familiar

Search with CKY

Can we do something similar with a seq2seq model?
Markov Decision Process

- Sequence of states, actions and rewards
  - $s_0, s_1, s_2, \ldots, s_T$ from a set $S$
  - $a_0, a_1, a_2, \ldots, a_T$ from a set $A$
    - Let’s assume a deterministic transition function $f:S \times A \rightarrow S$
  - $r_0, r_1, r_2, \ldots, r_T$ given by a reward function $r(s,a)$
- We want a policy $\pi(a \mid s)$ providing a distribution over actions that will maximize expected future reward
Seq2seq as MDP

- $s_t$: $h_t$
- $a_t$ is in $A(s_t)$
  - Either all symbols in the target vocabulary
  - All **valid** symbols if we check grammaticality
- $r_t$ is zero in all steps except the last. Then, it is 1 if execution results in a correct answer and 0 otherwise.

Liang et al, 2017, Guu et al., 2017
Seq2seq as MDP: policy

\[
p(z \mid x) = \prod_t p(z_t \mid x, z_0, \ldots, z_{t-1})
\]
\[
= \prod_t p(a_t \mid x, a_0, \ldots, a_{t-1})
\]
\[
= \prod_t \pi(a_t \mid s_t)
\]
\[
\pi(a_t \mid s_t) = \text{softmax}(W^{(s)} h_t)
\]
Seq2seq as MDP: policy

\[ p(z \mid x) = \prod_t p(z_t \mid x, z_0, \ldots, z_{t-1}) = \prod_t p(a_t \mid x, a_0, \ldots, a_{t-1}) = \prod_t \pi(a_t \mid s_t) \]

\[ \pi(a_t \mid s_t) = \text{softmax}(W^{(s)}h_t) \]

How do we learn?
Option 1: Maximum marginal likelihood

- Our data is language-dentation pairs \((x,y)\)
- We obtain \(y\) by constructing a logical form \(z\)
- We can use maximum marginal likelihood like before
- Interleave search and learning
  - Apply search to get candidate logical forms (with fixed model)
  - Update parameters based on candidates
- Difference from before:
  - Search was done with CKY and learning was a globally-normalized model
  - Search can be done with beam search and we have a locally-normalized model
Explanation

• No exact search —> Need a search algorithm

• In linear models: use the globally normalized scoring function to score partial trees

• Should this work?

Scoring function: A distribution over full derivations $p_\theta(d | x)$:

But search over partial derivations:
Explanation

• For sequence to sequence models:

\[ \sum \log p(z) R(z) \]
Explanation

• For sequence to sequence models:

\[
\sum \log p(z) R(z)
\]

\[
t=1 \quad t=2 \quad t=3 \quad t=4 \quad t=5 \quad t=6
\]
Maximum marginal likelihood

- $y$ is independent of $x$ conditioned on $z$

\[
p_{\theta}(y \mid x) = \sum_z p_{\theta}(z \mid x) \cdot p(y \mid z)
\]

\[
= \sum_z p_{\theta}(z \mid x) R(z) = E_{p_{\theta}(z \mid x)}[R(z)]
\]

\[
\mathcal{L}_{\text{MML}}(\theta) = \log \prod_{(x, y)} p_{\theta}(y \mid x) = \log \prod_{(x, y)} E_{p_{\theta}(z \mid x)}[R(z)]
\]

\[
= \sum_{(x, y)} \log \sum_z p_{\theta}(z \mid x) \cdot R(z)
\]
Gradient of MML

- Gradient has similar form to what we have seen in the past, except that we are not in a log-linear model. Let's assume a binary reward:

\[
\nabla_{\theta} \log \sum_z p_\theta(z \mid x) \cdot R(z) = \sum_z \frac{p_\theta(z) R(z) \nabla \log p_\theta(z \mid x)}{\sum_{z'} p_\theta(z' \mid x) \cdot R(z')} \\
= \sum_z p(z \mid x, R(z) = 1) \nabla \log p_\theta(z \mid x)
\]

- Compute the gradient of the log probability for every logical form, and weight the gradient using the reward.
Computing the gradient

• We can not enumerate all of the logical forms

• Instead we perform beam search as usual and get a beam $Z$ containing $K$ logical forms.

• We imagine that this beam is the entire set of possible logical forms

$$\sum_{z \in Z} p(z \mid x, R(z) = 1) \nabla \log p_\theta(z \mid x)$$

• For every $z$ we can compute the gradient of $\log p(z \mid x)$ since this is now the usual seq2seq setup.
Option 2: policy gradient

- We would like to simply maximize our expected reward

\[ E_{p_\theta(z|x)}[R(z)] = \sum_z p_\theta(z \mid x) R(z) \]

\[ \mathcal{L}_{RL}(\theta) = \sum_{(x,y)} \sum_z p_\theta(z \mid x) R(z) = \sum_{(x,y)} E_{p_\theta(z|x)}[R(z)] \]

\[ \nabla \mathcal{L}_{RL}(\theta) = \sum_{(x,y)} \sum_z p_\theta(z \mid x) R(z) \nabla \log p_\theta(z \mid x) \]

\[ = \sum_{(x,y)} E_{p_\theta(z|x)}[R(z) \nabla \log p_\theta(z \mid x)] \]

- Weight the gradient by the product of the reward and the model probability
Computing the gradient

• Again, we can not sum over all logical forms

• But the gradient for every example is an expectation over a distribution we can sample from!

• So we can sample many logical forms, compute the gradient and sum them weighted by the product of the model probability and reward

• Again, for every sample this is regular seq2seq and we can compute an approximate gradient
Some differences

- Using MML with beam search is a biased estimator and has less exploration - we only observe the approximate top-K logical forms.

- Using RL could be harder to train. If we have a correct logical form $z^*$ that has low probability at the beginning of training, then the contribution to the gradient would be very small and it would be hard to bootstrap.
Intermediate summary

• Training with a seq2seq model with weak supervision is problematic because the loss function is not a differentiable function of the input

• We saw both MML and RL approaches for getting around that

• In both we find a set of logical forms, compute the gradient for them like in supervised learning, and weight them in some way to form a final gradient

• This let’s us train with SGD

• Often this is still hard to train - more ahead
Variant 1: $\epsilon$-greedy beam search

- Do MML, but instead of beam search, at each time step $T$
  - Compute all possible continuations
  - for $i=1\ldots K$
    - with probability $\epsilon$
      - randomly sample a continuation without replacement
    - O/W take top continuation
- Motivation: improve exploration in beam search
Variant 2: Replay Buffer/ Memory

- Concept from DQN, a value-based RL method
  - Agents performs actions and observe rewards
  - Those are stored in a huge memory buffer
  - Training is done by sampling from the buffer
- Motivation: make RL look like supervised learning by decor relating samples
Variant 2: Replay Buffer/Memory

- We can do something similar with different motivation:
  - For every example keep a buffer with all programs with positive reward
  - Replace beam search by sampling a negative example + marginalizing or sampling a negative example
- It can be shown this reduces the gradient variance
- Requires smaller beams —> faster training
Variant 3: MML/RL hybrids

- There is a relation between MML and RL gradients
  
  - \( \text{grad}_{rl} = p_+(Z) \text{grad}_{mml} \)
  
  - \( p_+(Z) \): sum of probabilities for all positive reward programs
  
- Problem: at the beginning of training \( p_+(Z) \) is really small
  
- Replace \( p_+(Z) \) with \( \alpha (=0.1) \) if \( p_+(Z) < \alpha \)
  
- Effect: more exploitation at the beginning of training without too much bias.
Variant 4: continuous relaxations

- Using softmax directly
- Temperature
- Straight-through estimator

Softmax

- Replace argmax with softmax at training time
  - Pass as input the average of embeddings
  - Use argmax at test time
- Could approximate argmax (skewed distributions)
- Train-test mismatch (you don’t train on logical forms)

softmax

Embed and avg.
Adding temperature

\[ p(a) = \frac{\exp\left( \frac{s(a)}{t} \right)}{\sum_{a'} \exp\left( \frac{s(a')}{t} \right)} \]

- Start with high temperature:
  - when \( t = 1 \) this is softmax
  - when \( t \) is high - uniform
- Anneal temperature towards 0: get close to argmax
Straight-through estimator

- Use argmax at forward pass
- Pretend that you had softmax in the backward pass
Global loss functions

- The first is “better” than the second from a maximum likelihood point of view.
Exposure bias

• Training time:
  
  • Model observes correct tokens only
    
    • *I saw the big …*

  • At test time we observer predicted tokens
    
    • *I saw the the…*

• After an error our hidden state might be different from anything we have seen and errors can accumulate - train-test mismatch
Solution - RL?

- We can just define a reward on the entire token sequence! Then we have the same setup as semantic parsing

- But training with REINFORCE is much harder than training with ML…

- Imitation learning
Imitation learning
Imitation learning

• In sequence to sequence the expert is simply the correct sequence. So this is exactly maximum likelihood

• Imitation learning algorithms provide a way to avoid exposure bias
Dagger

- For an example \((x, y)\) and an expert \(\pi^*\)

- Define a policy \(\pi = \beta \pi^* + (1-\beta)\pi_t\)

- Sample from \(\pi\), and use \((x,y)\) to define loss

- Train

- Reduce \(\beta\) exponentially
Dagger

- This gets rid of exposure bias

- Problems: in some cases after you sample something wrong, it is hard to define an expert/oracle at all!
Summary

• Using sequence to sequence models with delayed reward does not have a consensus solution yet

• Solutions range from

  • Continuous relaxations
  
  • REINFORCE like algorithm

  • Mixing MML with RL in various ways

  • Many flavors of curriculum learning/annealing
Adding structure
Structure

• We have treated semantic parsing as a sequence to sequence problem ignoring the syntax of the output language.

• We can reduce the search space by taking that into account.
CFG formal language

• If the formal language is a CFG (highly likely) we can use existing algorithms

• One can efficiently compute for any prefix if it can be generated by a CFG with an Earley parser (top-down, left-to-right parser)

• Try all possible continuations and allow only valid ones at both training (pre-process) and test (online) time
Design the language so it’s trivial

Largest city in the US

(v0 USA)
(v1 Hop v0 CityIn)
(v2 Argmax v1 Population)

(v2 Argmax v1 Population)

(v1 Hop v0 CityIn)  Population

(v0 USA)  CityIn

LF Describes bottom up derivation
Design the language so it’s trivial

\(x:\) One tower has a yellow base.

\(z:\) EqualInt 1 Count Filter ALL_ITEMS \(\lambda x\) And IsYellow \(x\) IsBottom \(x\)

\(s:\) Int Set Bool Item
  Bool Int Int Set BoolFunc BoolFunc Bool Bool Bool Bool Bool Item

- This is a top-down parser
Arbitrary shift-reduce parsers

• For any structure for a logical form - if we can define a shift-reduce parser that outputs that structure we can map the input language to a sequence of discrete actions and use sequence to sequence
Graph-based parsing

• Unaware of graph-based neural approaches for semantic parsing

• There is a bit of work on that on syntactic parsing (Lee et al., EMNLP 2016 best paper)