Project 2 grades will be up tomorrow morning

Final project guidelines posted on the website (proposals due Nov 9, presentations Dec 5+7, project due Dec 15)

- Includes some pointers to datasets, etc.
- Be thinking about what you want to do!
Recall: RNNs

- Cell that takes some input $\mathbf{x}$, has some hidden state $\mathbf{h}$, and updates that hidden state and produces output $\mathbf{y}$ (all vector-valued)
Recall: LSTMs

- Forget gate $f$ controls how cell state changes, $i/o$ control input/output
- $g$ reflects the main computation of the cell

Goldberg lecture notes
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Recall: Alignments in NLI

- Two statements often have a natural alignment between them.
- Process the hypothesis with knowledge of the premise.
- Seeing the alignment lets you make entailment judgments as you’re reading the sentence.

Bowman et al. (2015)
Attention Mechanism

- *Learned* notion of alignment to some input

Compare hidden state to encoded input vectors to compute alignment, use that to compute an input to further processing

- Attention models: 85-86% on SNLI, SOTA = 88%
This Lecture

- Encoder-decoder models for machine translation
- Attention
- Handling rare words in machine translation
- Other applications
Encoder-Decoder Models
encoder-decoder

- Encode a sequence into a fixed-sized vector
- Now use that vector to produce a *sentence* as output from a separate LSTM *decoder*
It’s not an ACL tutorial on vector representations of meaning if there’s at least one Ray Mooney quote.

▶ Is this true? Sort of...we’ll come back to this later
Inference

- Generate next word conditioned on previous word as well as hidden state

- W size is $|\text{vocab}| \times |\text{hidden state}|$, softmax over entire vocabulary

$$P(w_i | x, w_{i-1}) = \text{softmax}(W \bar{h})$$

Decoder has separate parameters, so this can learn to be a language model (produce a plausible next word given current one)
Inference

- Generate next word conditioned on previous word as well as hidden state

  - During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state

  - Need to actually evaluate computation graph up to this point to form input for the next state

  - Decoder is advanced one state at a time until [STOP] is reached

```
the movie was great
```
Objective: maximize $\log P(w_i^*|x, w_{i-1}^*)$

One loss term for each target-sentence word, feed the correct word regardless of model’s prediction

Length of gold sequence is known, can run the whole encoder-decoder in one computation graph and compute losses
Scheduled Sampling

- Model needs to do the right thing even with its own predictions

- Scheduled sampling: with probability $p$, take the gold as input, else take the model’s prediction

- Starting with $p = 1$ and decaying it works best

Bengio et al. (2015)
Implementation Details

- Sentence lengths vary for both encoder and decoder:
  - Dynamic computation graphs framework (PyTorch, DyNet) build graphs of the correct length for a batch on-the-fly
  - Otherwise, pad everything to the right length and use a mask or indexing to access a subset of terms

- Beam search: when decoding, can use beam search rather than taking the one-best word each time

- Ensembling: these models are nonconvex, almost always works better to train several and ensemble their predictions
Machine Translation Results

WMT English-French: 12M sentence pairs, 80,000 word target vocab

Classic phrase-based system: ~33 BLEU, uses additional target-language data

Rerank with LSTMs: 36.5 BLEU (long line of work here; Devlin+ 2014)

Sutskever+ (2014) seq2seq single: 30.6 BLEU

Sutskever+ (2014) seq2seq ensemble: 34.8 BLEU

- But English-French is a really easy language pair and there’s tons of data for it! Does this approach work for anything harder?
Machine Translation Results

WMT English-German: 4.5M sentence pairs, 50,000 word target vocab

Classic phrase-based system: 20.7 BLEU

Luong+ (2014) seq2seq: 14 BLEU

- Not nearly as good...
Attention
Problems with Neural MT Models

- Encoder-decoder models like to repeat themselves:

  Un garçon joue dans la neige → A boy plays in the snow boy plays boy plays

- Often a byproduct of training these models poorly

- Solution: include coverage in the model so we don’t repeat stuff: Haitao Mi et al. (2016) for MT, See and Manning (2017) for summarization
Problems with Neural MT Models

- Unknown words:

  *en*: The *écotax* portico in *Pont-de-Buis*, ... [truncated] ..., was taken down on Thursday morning

  *fr*: Le *portique écotaxe* de *Pont-de-Buis*, ... [truncated] ..., a été *démonté* jeudi matin

  *nn*: Le *unk* de *unk* à *unk*, ... [truncated] ..., a été pris le jeudi matin

- We restricted the target vocabulary to 80,000 — that throws out a lot!

- Fixed vocabulary is too restrictive, especially around named entities
Problems with Neural MT Models

- Bad at long sentences: 1) a fixed-size representation doesn’t scale; 2) LSTMs still have a hard time remembering for really long periods of time

RNNsearch: introduces attention mechanism to give “variable-sized” representation

Bahdanau et al. (2014)
Suppose we knew the source and target would be purely monotonic.

Can look at the corresponding input word when translating — this could scale!

Much less burden on the hidden state.
For each decoder state, compute a weighted sum of input states reflecting what’s most important right now.

\[ e_{ij} = f(\tilde{h}_i, h_j) \]

- Unnormalized scalar weight

\[ \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \]

- Normalized scalar weight

\[ c_i = \sum_j \alpha_{ij} h_j \]

- Weighted sum of input hidden states (vector)

The movie was great
Attention

- Can also use attention weights from previous timestep as input to current attention computation; captures monotonicity
  
  Luong et al. (2015)

\[ e_{ij} = f(\tilde{h}_i, h_j) \]

\[ \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \]

\[ c_i = \sum_j \alpha_{ij} h_j \]

\[ f(\tilde{h}_i, h_j) = \tanh(W[\tilde{h}_i, h_j]) \]

- Bahdanau+ (2014): additive

\[ f(\tilde{h}_i, h_j) = \tilde{h}_i \cdot h_j \]

- Luong+ (2015): dot product

\[ f(\tilde{h}_i, h_j) = \tilde{h}_i^\top W h_j \]

- Luong+ (2015): bilinear
Encoder hidden states capture contextual source word identity

Decoder hidden states are now mostly responsible for selecting what to attend to

Doesn’t take a complex hidden state to walk monotonically through a sentence and spit out word-by-word translations
Machine Translation Results

WMT English-French: 12M sentence pairs, 80,000 word target vocab

Classic phrase-based system: ~33 BLEU, uses additional target-language data

Rerank with LSTMs: 36.5 BLEU (long line of work here; Devlin+ 2014)

Sutskever+ (2014) seq2seq single: 30.6 BLEU

Sutskever+ (2014) seq2seq ensemble: 34.8 BLEU

Bahdanau+ (2014) seq2seq with attention: 28.5 BLEU

- But English-French is a really easy language pair!

Results from Luong et al. (ACL 2015)
Machine Translation Results

WMT English-German: 4.5M sentence pairs, 50,000 word target vocab

Classic phrase-based system: 20.7 BLEU

Basic seq2seq: 14 BLEU

seq2seq with attention: 16.8 BLEU

seq2seq with attention aware of previous attention: 18.1 BLEU

^ ensemble + rare word handling: 23.0 BLEU

- Attention more critical for the harder English-German task

Results from Luong et al. (EMNLP 2015)
Dealing with Rare Words
1) Named entities: copy (and maybe transliterate)
2) Rare concepts: may be able to get from transliteration, generally hard

Neural MT models have to generate from a fixed vocabulary, but we at least want to be able to copy named entities

Jean et al. (2015), Luong et al. (2015)
Copying

- Predict an unk token with a pointer to a source word to copy

- Input: en: The $\text{unk}_1$ portico in $\text{unk}_2$ ...

- Output: fr: Le $\text{unk}_0 \text{unk}_1$ de $\text{unk}_2$ ...

- Easy to do and helps a lot! (+ a few BLEU points, typically)

- Similar to pointer networks, which we’ll see later

Jean et al. (2015), Luong et al. (2015)
Rare Words: Character Models

- If we predict an unk token, generate the results from a character LSTM
- Can potentially transliterate new concepts, but architecture is more complicated and slower to train
- Models like this in part contributed to dynamic computation graph frameworks becoming popular

Luong et al. (2016)
Rare Words: Word Piece Models

- Use Huffman encoding on a corpus, keep most common $k$ (~10,000) character sequences for source and target

  Input: _the _eco tax _port i co _in _Po nt - de - Bu is;...
  Output: _le _port ique _éco taxe _de _Pont - de - Buis;

- Captures common words and parts of rare words
- Subword structure may make it easier to translate
- Model balances translating and transliterating without explicit switching

Wu et al. (2016)
Google’s NMT System

- 8-layer LSTM encoder-decoder with attention, word piece vocabulary of 8k-32k

Wu et al. (2016)
Google’s NMT System

English-French:
Google’s phrase-based system: 37.0 BLEU
Luong+ (2015) seq2seq ensemble with rare word handling: 37.5 BLEU
Google’s 32k word pieces: 38.95 BLEU

English-German:
Google’s phrase-based system: 20.7 BLEU
Luong+ (2015) seq2seq ensemble with rare word handling: 23.0 BLEU
Google’s 32k word pieces: 24.2 BLEU

Wu et al. (2016)
Human Evaluation (En-Es)

- Similar to human-level performance on *English-Spanish*

Wu et al. (2016)
Other Applications
Other Applications

- Parsing: input is a sentence, output is a bracketed sentence

- Attention is essential: <70 F1 without it, 88.3 F1 / 90.5 F1 (ensemble) with it

- The best parsers still use some structure — we’ll come back to these

Vinyals et al. (2014)
Other Applications

- Summarization/compression
  - Input: article/sentence, output: compressed article/sentence

On Friday, the U.S. intends to announce...

- Long articles, hard to deal with even with attention

- Speech recognition/text-to-speech: neural nets are good at dealing with continuous speech signals!
RNNs are effective at machine translation, but lots of tricks to get them to work right

Attention is a critical way to get a better representation of the input

Handling rare words is important, lots of techniques here

Encoder-decoder models can be successfully applied to most tasks where you generate language as output