Recall: Training Tips

- Parameter initialization is critical to get good gradients, some useful heuristics (e.g., Glorot initializer)
- Dropout is an effective regularizer
- Think about your optimizer: Adam or tuned SGD work well

Recall: Word Vectors

- The president said that the downturn was over
- [Finch and Chater 92, Shuetze 93, many others]
Recall: Continuous Bag-of-Words

- Predict word from context

\[ \text{the} \text{dog} \text{bit} \text{the} \text{man} \]

Mikolov et al. (2013)

- Matrix factorization approaches useful for learning vectors from really large data

\[ \sum, \text{size } d \]

\[ \text{Multiply by } W \rightarrow \text{softmax} \rightarrow P(w|w_{-1}, w_{+1}) \]

Using Word Embeddings

- Approach 1: learn embeddings directly from data in your neural model, no pretraining
  - Often works pretty well
- Approach 2: pretrain using GloVe, keep fixed
  - Faster because no need to update these parameters
  - Need to make sure GloVe vocabulary contains all the words you need
- Approach 3: initialize using GloVe, fine-tune
  - Not as commonly used anymore

Compositional Semantics

- What if we want embedding representations for whole sentences?

- Skip-thought vectors (Kiros et al., 2015), similar to skip-gram generalized to a sentence level (more later)

- Is there a way we can compose vectors to make sentence representations? Summing? RNNs?

This Lecture

- Recurrent neural networks
- Vanishing gradient problem
- LSTMs / GRUs
- Applications / visualizations
RNN Basics

RNN Motivation

- Feedforward NNs can’t handle variable length input: each position in the feature vector has fixed semantics
- Instead, we need to:
  1) Process each word in a uniform way
  2) …while still exploiting the context that that token occurs in
- These don’t look related (great is in two different orthogonal subspaces)

RNN Abstraction

- Cell that takes some input $x$, has some hidden state $h$, and updates that hidden state and produces output $y$ (all vector-valued)

RNN Uses

- Transducer: make some prediction for each element in a sequence
  
  
  output $y =$ score for each tag, then softmax

- Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose
  
  predict sentiment (matmul + softmax)
  translate
  paraphrase/compress
Elman Networks

- Computes output from hidden state $y_t = \tanh(Uh_t + b_y)$
- Updates hidden state based on input and current hidden state $h_t = \tanh(Wx_t + Vh_{t-1} + b_h)$
- Long history! (invented in the late 1980s)

Elman (1990)

Training Elman Networks

- "Backpropagation through time": build the network as one big computation graph, some parameters are shared
- RNN potentially needs to learn how to "remember" information for a long time!

it was my favorite movie of 2016, though it wasn't without problems

- "Correct" parameter update is to do a better job of remembering the sentiment of favorite

Vanishing Gradient

- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0

LSTMs/GRUs

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Gated Connections

- Designed to fix “vanishing gradient” problem using gates
  \[ h_t = h_{t-1} \odot f + \text{func}(x_t) \]
  
  \[ h_t = \tanh(Wx_t + Vh_{t-1} + b_h) \]
  
  Elman

- Vector-valued “forget gate” \( f \) computed based on input and previous hidden state
  \[ f = \sigma(W^xf x_t + W^hf h_{t-1}) \]

- Sigmoid: elements of \( f \) are in \([0, 1]\)
  \[ f = \sigma(W^xf x_t + W^hf h_{t-1}) \]

- If \( f = 1 \), we simply sum up a function of all inputs — gradient doesn’t vanish!

LSTM

- “Cell” \( c \) in addition to hidden state \( h \)
  \[ c_t = c_{t-1} \odot f + \text{func}(x_t, h_{t-1}) \]

- Vector-valued forget gate \( f \) depends on the \( h \) hidden state
  \[ f = \sigma(W^xf x_t + W^hf h_{t-1}) \]

- Basic communication flow: \( x \rightarrow c \rightarrow h \), each step of this process is gated in addition to gates from previous timesteps

LSTM Diagrams

- \( f, i, o \) are gates that control information flow
- \( g \) reflects the main computation of the cell

- Can we ignore \( c \) in our current computation?
- Can an LSTM sum up its inputs \( x \)?
- Can we ignore a particular input \( x \)?
- Can we output something without changing \( c \)?
LSTMs

- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

Gradient still diminishes, but in a controlled way and generally by less — usually initialize forget gate = 1 to remember everything to start

Understanding LSTM Parameters

- Initialize hidden layer randomly
- Need to learn how the gates work: what do we forget/remember?
- g uses an arbitrary nonlinearity, this is the “layer” of the cell

LSTM: more complex and slower, may work a bit better

GRUs

- GRU: faster, a bit simpler
- Two gates: z (forget, mixes s and h) and r (mixes h and x)
What do RNNs produce?

- **Encoding of the sentence** — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- **Encoding of each word** — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

Multilayer Bidirectional RNN

- Sentence classification based on concatenation of both final outputs
- Token classification based on concatenation of both directions’ token representations

Training RNNs

- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network
- Example: sentiment analysis

Training RNNs

- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network
- Example: language modeling (predict next word given context)
What can LSTMs model?

- Sentiment
- Encode one sentence, predict
- Language models
- Move left-to-right, per-token prediction
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (later in the course)

Visualizing LSTMs

- Train character LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells (components of c) to understand them
- Counter: know when to generate \n
The sole importance of the crossing of the Berezina lies in the fact that it plainly and indisputably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action—the one Kutuzov and the general staff of the army demanded—namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all---carried on by vis inertia---pressed forward into boats and into the ice-covered water and did not surrender.

Karpathy et al. (2015)
**Visualizing LSTMs**

- Train character LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation

```
#define CONFIG_AUDIT_SSCALL
static int sclass_read_class(int class, int *mask)
{
    int i;
    for (i = 0; i < READ_CLASS_SIZE; i++)
        mask[i] = 0;
    return i;
}
```

Karpathy et al. (2015)

**Natural Language Inference**

<table>
<thead>
<tr>
<th>Premise</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A boy plays in the snow</td>
<td>entails</td>
</tr>
<tr>
<td>A man inspects the uniform of a figure</td>
<td>contradicts</td>
</tr>
<tr>
<td>An older and younger man smiling</td>
<td>neutral</td>
</tr>
</tbody>
</table>

- Long history of this task: “Recognizing Textual Entailment” challenge in 2006 (Dagan, Glickman, Magnini)
- Early datasets: small (hundreds of pairs), very ambitious (lots of world knowledge, temporal reasoning, etc.)

**What can LSTMs model?**

- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (next lecture)
- Textual entailment
  - Encode two sentences, predict

**Uninterpretable:** probably doing double-duty, or only makes sense in the context of another activation

Visualize activations of specific cells to see what they track

Train character LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

What can LSTMs model?
**SNLI Dataset**

- Show people captions for (unseen) images and solicit entailed / neural / contradictory statements
- >500,000 sentence pairs
- Encode each sentence and process
- 100D LSTM: 78% accuracy (Bowman et al., 2016)
- 300D LSTM: 80% accuracy (Liu et al., 2016)
- 300D BiLSTM: 83% accuracy (Liu et al., 2016)
- Later: better models for this

**Takeaways**

- RNNs can transduce inputs (produce one output for each input) or compress the whole input into a vector
- Useful for a range of tasks with sequential input: sentiment analysis, language modeling, natural language inference, machine translation
- Next time: CNNs and neural CRFs

Diagram:
- 3-way softmax classifier
- 200d tanh layer
- 100d premise
- 100d hypothesis
- Sentence model with premise input
- Sentence model with hypothesis input

Bowman et al. (2015)