CS 4803 / 7643: Deep Learning

Topics:
- LSTMs
- Language Modeling
- Recursive Neural Networks

Zsolt Kira
Georgia Tech
Administrivia

• Project comments have been given back (second iteration)

• **NOTE:** Updated PS3
  [https://piazza.com/class/jos18hnsf6t2cf?cid=267](https://piazza.com/class/jos18hnsf6t2cf?cid=267)
Long Short Term Memory (LSTM)

**Vanilla RNN**

\[ h_t = \tanh \left( W \left( h_{t-1}, x_t \right) \right) \]

**LSTM**

\[
\begin{pmatrix}
    i \\
    f \\
    o \\
    g
\end{pmatrix} =
\begin{pmatrix}
    \sigma \\
    \sigma \\
    \sigma \\
    \tanh
\end{pmatrix} W \begin{pmatrix}
    h_{t-1} \\
    x_t
\end{pmatrix}
\]

\[ c_t = f \odot c_{t-1} + i \odot g \]

\[ h_t = o \odot \tanh(c_t) \]

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Meet LSTMs

(C) Dhruv Batra  
Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
LSTMs Intuition: Memory

- Cell State / Memory
LSTMs Intuition: Forget Gate

• Should we continue to remember this “bit” of information or not?

\[ f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f) \]
LSTMs Intuition: Input Gate

- Should we update this “bit” of information or not?
  - If so, with what?

\[
i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)
\]
LSTMs Intuition: Memory Update

- Forget that + memorize this

\[ C_t = f_t \ast C_{t-1} + i_t \ast \tilde{C}_t \]
LSTMs Intuition: Output Gate

• Should we output this “bit” of information to “deeper” layers?

\[
o_t = \sigma (W_o [h_{t-1}, x_t] + b_o) \\
h_t = o_t \times \tanh(C_t)
\]
LSTMs Intuition: Additive Updates

Backpropagation from $c_t$ to $c_{t-1}$ only elementwise multiplication by $f$, no matrix multiply by $W$
LSTMs Intuition: Additive Updates

Uninterrupted gradient flow!
LSTMs Intuition: Additive Updates

Uninterrupted gradient flow!

Similar to ResNet!
LSTMs

- A pretty sophisticated cell
LSTM Variants #1: Peephole Connections

- Let gates see the cell state / memory

\[
\begin{align*}
    f_t &= \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\
    i_t &= \sigma (W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\
    o_t &= \sigma (W_o \cdot [C_t, h_{t-1}, x_t] + b_o)
\end{align*}
\]
LSTM Variants #2: Coupled Gates

- Only memorize new if forgetting old

\[ C_t = f_t \cdot C_{t-1} + (1 - f_t) \cdot \tilde{C}_t \]
LSTM Variants #3: Gated Recurrent Units

- Changes:
  - No explicit memory; memory = hidden output
  - $Z = \text{memorize new and forget old}$

\[
egin{align*}
  z_t &= \sigma (W_z \cdot [h_{t-1}, x_t]) \\
  r_t &= \sigma (W_r \cdot [h_{t-1}, x_t]) \\
  \tilde{h}_t &= \tanh (W \cdot [r_t \ast h_{t-1}, x_t]) \\
  h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
Other RNN Variants

[An Empirical Exploration of Recurrent Network Architectures, Jozefowicz et al., 2015]

\[
\begin{align*}
\text{MUT1:} \\
z &= \text{sigm}(W_{xz}x_t + b_z) \\
r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
h_{t+1} &= \text{tanh}(W_{hh}(r \odot h_t) + \text{tanh}(x_t) + b_h) \odot z \\
&\quad + h_t \odot (1 - z) \\
\text{MUT2:} \\
z &= \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\
r &= \text{sigm}(x_t + W_{hr}h_t + b_r) \\
h_{t+1} &= \text{tanh}(W_{hh}(r \odot h_t) + W_{zh}x_t + b_h) \odot z \\
&\quad + h_t \odot (1 - z) \\
\text{MUT3:} \\
z &= \text{sigm}(W_{xz}x_t + W_{hz} \text{tanh}(h_t) + b_z) \\
r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
h_{t+1} &= \text{tanh}(W_{hh}(r \odot h_t) + W_{zh}x_t + b_h) \odot z \\
&\quad + h_t \odot (1 - z)
\end{align*}
\]
Hidden Markov Models (computer scientists love them!)

• Hidden Markov Models have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.
  – We cannot be sure which state produced a given output. So the state is “hidden”.
  – It is easy to represent a probability distribution across N states with N numbers.

• To predict the next output we need to infer the probability distribution over hidden states.
  – HMMs have efficient algorithms for inference and learning.
A fundamental limitation of HMMs

• What happens when an HMM generates data?
  – At each time step it must select one of its hidden states.
  – With $N$ hidden states it can only remember $\log(N)$ bits about what it generated so far.

• Consider the information that the first half of an utterance contains about the second half:
  – The syntax needs to fit (e.g. number and tense agreement).
  – The semantics needs to fit. The intonation needs to fit.
  – The accent, rate, volume, and vocal tract characteristics must all fit.

• All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half. $2^{100}$ is big!
Recurrent neural networks

• RNNs are very powerful, because they combine two properties:
  – Distributed hidden state that allows them to store a lot of information about the past efficiently.
  – Non-linear dynamics that allows them to update their hidden state in complicated ways.

• With enough neurons and time, RNNs can compute anything that can be computed by your computer.
  – Universal approximation machine (Turing machine)
Language Modeling
Importance of text processing

• Text processing core to several companies (Google, Facebook, Yahoo, etc.)

• Lots of applications
  – Spam detection
  – Ad, movie, music recommendations
  – Email categorization
  – Machine Translation
  – Speech recognition
  – Image annotation
  – Generation of news articles
NLP Tasks

- Sentiment analysis
- Morphology
- Parsing
- Semantic parsing
- Paraphrase
- Analogies
- Language modeling
- Named entity recognition
- Part of speech tagging
- ...
Previous Approaches

• Previous approaches dominated by:
  – N-grams
  – Word classes
  – Bag-of-words representations

• Standard application of ML techniques
  – Logistic regression
  – SVMs

• Sound familiar?
  – Deep learning is prime candidate for text: Rich hierarchical structure that can be represented
  – Not as simple as you think
N-Grams

• Standard approach to language modeling

• Task: compute probability of a sentence $W$

$$P(W) = \prod_i P(w_i | w_1 \ldots w_{i-1})$$

• Often simplified to trigrams:

$$P(W) = \prod_i P(w_i | w_{i-2}, w_{i-1})$$
N-gram example

\[ P(\text{"this is a sentence"}) = P(\text{this}) \times P(\text{is}|\text{this}) \times P(\text{a}|\text{this, is}) \times P(\text{sentence}|\text{is, a}) \]

- The probabilities are estimated from counts:

\[ P(\text{a}|\text{this, is}) = \frac{C(\text{this is a})}{C(\text{this is})} \]

- Smoothing is used to redistribute probability to unseen events (this avoids zero probabilities)

*A Bit of Progress in Language Modeling* (Goodman, 2001)
One-hot representations

- Simple way how to encode discrete concepts, such as words

Example:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday    = [1 0 0 0 0]
Tuesday   = [0 1 0 0 0]
is        = [0 0 1 0 0]
a         = [0 0 0 1 0]
today     = [0 0 0 0 1]
```

Also known as 1-of-N (where in our case, N would be the size of the vocabulary)
Bag of Words representations

- Sum of one-hot codes
- Ignores order of words

Example:

```plaintext
vocabulary = (Monday, Tuesday, is, a, today)
Monday Monday       = [2 0 0 0 0]
today is a Monday   = [1 0 1 1 1]
today is a Tuesday  = [0 1 1 1 1]
is a Monday today   = [1 0 1 1 1]
```

Can be extended to bag-of-N-grams to capture local ordering of words
Representations using Distributional Similarity

You can get a lot of value by representing a word by means of its neighbors

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in saying that Europe needs unified banking regulation to replace the hodgepodge

疚 These words will represent banking 猝

You can vary whether you use local or large context to get a more syntactic or semantic clustering
Vector representations

• Instead of a sparse one-hot vector, represent words as a dense vector

- Bigram neural language model
- Previous word is used to predict the current word by going through hidden layer (classifier with as many outputs as there are words in the vocabulary)
• The input is encoded as one-hot
• The model will learn compressed, continuous representations of words (usually the matrix of weights between the input and hidden layer)
• We call the vectors in the matrix between the input and hidden layer *word vectors* (also known as *word embeddings*)

• Each word is associated with a real valued vector in N-dimensional space (usually $N = 50 - 1000$)

• The word vectors have some similar properties to word classes; however, many degrees of similarity are captured
Word vectors training and use

- These word vectors can be subsequently used as features in many NLP tasks (Collobert et al, 2011)

- As word vectors can be trained on huge text datasets, they provide generalization for systems trained with limited amount of supervised data

- More complex model architectures can be used for obtaining the word vectors (neural net language model with multi-task learning (Collobert & Weston, 2008))
Linguistic regularities in word vectors

- Recently, it was shown that word vectors capture many linguistic properties (gender, tense, plurality, even semantic concepts like “capital city of”)
- We can do nearest neighbor search around result of vector operation “King – man + woman” and obtain “Queen” ([Linguistic regularities in continuous space word representations](Mikolov et al, 2013))
Various architectures for word vectors

- Neural net based word vectors were traditionally trained as part of neural network language model (Bengio et al, 2003)
- This models consists of input layer, projection layer, hidden layer and output layer.

- Four-gram neural net language model architecture (Bengio 2001)
- The training is done using stochastic gradient descent and backpropagation
Word representations using RNNs

- Input layer $w$ and output layer $y$ have the same dimensionality as the vocabulary
- Hidden layer $s$ is orders of magnitude smaller
- $U$ is the matrix of weights between input and hidden layer, $V$ is the matrix of weights between hidden and output layer
BPTT for different values

From wikipedia:

In information theory, **perplexity** is a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models.

- Importance of BPTT training on Penn Corpus. BPTT=1 corresponds to standard backpropagation.
Linguistic regularities in word vectors

- Recently, it was shown that word vectors capture many linguistic properties (gender, tense, plurality, even semantic concepts like "capital city of")
- We can do nearest neighbor search around result of vector operation "King – man + woman" and obtain "Queen" (Linguistic regularities in continuous space word representations (Mikolov et al, 2013))
More examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Nearest token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris - France + Italy</td>
<td>Rome</td>
</tr>
<tr>
<td>bigger - big + cold</td>
<td>colder</td>
</tr>
<tr>
<td>sushi - Japan + Germany</td>
<td>bratwurst</td>
</tr>
<tr>
<td>Cu - copper + gold</td>
<td>Au</td>
</tr>
<tr>
<td>Windows - Microsoft + Google</td>
<td>Android</td>
</tr>
<tr>
<td>Montreal Canadiens - Montreal + Toronto</td>
<td>Toronto Maple Leafs</td>
</tr>
</tbody>
</table>
Building on Word Vector Space Models

the country of my birth
the place where I was born

But how can we represent the meaning of longer phrases?
By mapping them into the same vector space!
Mapping Phrases to Vector Spaces

Use principle of compositionality

The meaning (vector) of a sentence is determined by
(1) the meanings of its words and
(2) the rules that combine them.

Models in this section can jointly learn parse trees and compositional vector representations.
Semantic Vector Spaces

Vectors representing Phrases and Sentences that do not ignore word order and capture semantics for NLP tasks

Single Word Vectors
- Distributional Techniques
- Useful as features inside models, e.g. CRFs for NER, etc.
- Cannot capture longer phrases

Documents Vectors
- Bag of words models
- LSA, LDA
- Great for IR, document exploration, etc.
- Ignore word order, no detailed understanding
Sentence Parsing: Desired Output

The cat sat on the mat.
Learn structure and representation

The cat sat on the mat.
Structure Prediction

Inputs: two candidate children’s representations

Outputs:
1. The semantic representation if the two nodes are merged.
2. Score of how plausible the new node would be.
Recursive Neural Network Definition

\[ \text{score} = \begin{pmatrix} 1.3 \\ 8 \\ 5 \\ 3 \\ 3 \end{pmatrix} = \text{parent} \]

\[ \text{score} = U^T p \]

\[ p = \tanh(W \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} b), \]

Same \( W \) parameters at all nodes of the tree
Parsing a sentence with an ReNN

Start with word vectors
The cat sat on the mat.
The cat sat on the mat.
Can be used for vision too!
Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.