Logistics

- A3 grades are out.
- Project Presentation next Thursday! (04/14)
- Project final report due on 04/18.

Project Presentation

- 9 groups
- Each group presents for 12 minutes, and 3 minutes for QA
- We will send out the order before next Thursday

Also applies for the final report

Problem Description
- What is the task?
- System input and output
- Examples will be helpful

Reference/Related work
- Put your work in context: what has been done before?
- What's new in your work?

Methodology
- What you have done
  - Preprocessing of the data
  - What are your data samples? Features?
  - What methods do you experiment with? And why do you think they’re reasonable and suitable for the task?
- Experiments
  - Datasets
  - Evaluation metrics
  - Baselines
  - Results, tables, figures, etc

Word representation
- Recurrent Neural Networks (RNNs) and language modeling
- Bidirectional and deep RNNs
- Extensions on RNNs
  - Gated Recurrent Units (GRU)
  - Long short—term memories (LSTMs)

[Some slides are borrowed from Marek Rei, Ekaterina Kochmar, and Richard Socher's lectures]
Representing words as vectors

Let's represent words (or any objects) as vectors. We want to construct them, so that similar words have similar vectors.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>I live in Cambridge</td>
<td>19</td>
</tr>
<tr>
<td>I live in Paris</td>
<td>68</td>
</tr>
<tr>
<td>I live in Tallinn</td>
<td>0</td>
</tr>
<tr>
<td>I live in yellow</td>
<td>0</td>
</tr>
</tbody>
</table>

* Tallinn
  * Cambridge
  * London
  * Paris

* yellow
  * red
  * blue
  * green

1-hot vectors

How can we represent words as vectors?

Option 1: each element represents a different word.

Also known as “1-hot” or “1-of-N” representation.

<table>
<thead>
<tr>
<th></th>
<th>bear</th>
<th>cat</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>bear</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cat</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>frog</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

bear = [1.0, 0.0, 0.0]  cat = [0.0, 1.0, 0.0]

1-hot vectors

When using 1-hot vectors, we can’t fit many and they tell us very little. Need a separate dimension for every word we want to represent.

Distributed vectors

Option 2: each element represents a property, and they are shared between the words.

Also known as “distributed” representation.

<table>
<thead>
<tr>
<th></th>
<th>furry</th>
<th>dangerous</th>
<th>mammal</th>
</tr>
</thead>
<tbody>
<tr>
<td>bear</td>
<td>0.9</td>
<td>0.85</td>
<td>1</td>
</tr>
<tr>
<td>cat</td>
<td>0.85</td>
<td>0.15</td>
<td>1</td>
</tr>
<tr>
<td>frog</td>
<td>0.65</td>
<td>0.1</td>
<td>0</td>
</tr>
</tbody>
</table>

bear = [0.9, 0.85, 1.0]  cat = [0.85, 0.15, 1.0]

Distributed vectors

Can use cosine to calculate similarity between two words

\[
\cos(a, b) = \frac{\sum a_i \cdot b_i}{\sqrt{\sum a_i^2} \cdot \sqrt{\sum b_i^2}}
\]

\[
\cos(\text{lion, bear}) = 0.998 \\
\cos(\text{lion, dog}) = 0.809 \\
\cos(\text{cobra, dog}) = 0.727
\]
**Distributed vectors**

We can infer some information, based only on the vector of the word.
We don’t even need to know the labels on the vector elements.

**Distributional hypothesis**

Words which are similar in meaning occur in similar contexts.
(Harris, 1954)

You shall know a word by the company it keeps
(Firth, 1957)

He is reading a magazine. I was reading a newspaper.
This magazine published my story. The newspaper published an article.
She buys a magazine every month. He buys this newspaper every day.

**Count-based vectors**

One way of creating a vector for a word:
Let’s count how often a word occurs together with specific other words.

<table>
<thead>
<tr>
<th></th>
<th>reading a magazine</th>
<th>I was reading a newspaper</th>
</tr>
</thead>
<tbody>
<tr>
<td>This magazine</td>
<td>This magazine</td>
<td>published my story</td>
</tr>
<tr>
<td>Published</td>
<td>Published</td>
<td>an article</td>
</tr>
<tr>
<td>He buys a</td>
<td>She buys a</td>
<td>magazine every month</td>
</tr>
<tr>
<td>Magazine every</td>
<td>Magazine every</td>
<td>month</td>
</tr>
<tr>
<td>Magazine every</td>
<td>He buys this</td>
<td>this newspaper every day</td>
</tr>
<tr>
<td>Magazine every</td>
<td>Magazine every</td>
<td>month</td>
</tr>
<tr>
<td>Magazine every</td>
<td>Magazine every</td>
<td>day</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reading</th>
<th>a</th>
<th>this</th>
<th>published</th>
<th>my</th>
<th>buys</th>
<th>the</th>
<th>art</th>
<th>every</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>magazine</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>newspaper</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Count-based vectors**

- More frequent words dominate the vectors.
  Can use a weighting scheme like PMI or TF-IDF.

\[ \text{PMI}(x, z) = \log \frac{p(x, z)}{p(x)p(z)} \]

- tf-idf \( (w, z) = \text{freq}_{w,z} \cdot \log \frac{V}{n_w} \)

- Large number of sparse features
  Can use matrix decomposition like Singular Value Decomposition (SVD) or Latent Dirichlet Allocation (LDA).

**Neural word embeddings**

Neural networks will automatically try to discover useful features in the data, given
a specific task.

Idea: Let’s allocate a number of parameters for each word and allow the neural
network to automatically learn what the useful values should be.

Often referred to as “word embeddings”, as we are embedding the words into a
real-valued low-dimensional space.

**Embeddings through language modelling**

Predict the next word in a sequence based on the previous words.

\[ \text{i-th output} = \text{P}(w_i | \text{context}) \]

Use this to guide the training for word embeddings.


I read at my desk
I study at my desk
Embeddings through error detection

Take a grammatically correct sentence and create a corrupted counterpart.
Train the neural network to assign a higher score to the correct version of each sentence.


Word similarity

<table>
<thead>
<tr>
<th>Germany</th>
<th>China</th>
<th>Russia</th>
<th>Slovakia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>Beijing</td>
<td>Moscow</td>
<td>Bratislava</td>
</tr>
<tr>
<td>Czech</td>
<td>Hungarian</td>
<td>Slovak</td>
<td>Russian</td>
</tr>
</tbody>
</table>


Analogy recovery

The task of analogy recovery. Questions in the form:

\[ a \text{ is to } b \text{ as } c \text{ is to } d \]

The system is given words a, b, c, and it needs to find d. For example:

- ‘apple’ is to ‘apples’ as ‘car’ is to ?
- ‘man’ is to ‘woman’ as ‘king’ is to ?

Word embeddings in practice

Word2vec is often used for pretraining.

- It will help your models start from an informed position
- Requires only plain text - which we have a lot
- It very fast and easy to use
- Already pretrained vectors also available (trained on 100B words)
Today’s Outline

- Word representation
- Recurrent Neural Networks (RNNs) and language modeling
- Bidirectional and deep RNNs
- Extensions on RNNs
  - Gated Recurrent Units (GRU)
  - Long short-term memories (LSTMs)

[Some slides are borrowed from Marek Rei, Ekaterina Kochmar, and Richard Socher’s lectures]

Language Models

A language model computes a probability for a sequence of words: \( P(w_1, \ldots, w_T) \)

- Useful for machine translation and speech
  - Word choice: \( p(\text{walking home after school}) > p(\text{walking house after school}) \)

- Use incorrect but necessary Markov assumptions

\[
P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_1, \ldots, w_{t-1}) \approx \prod_{t=1}^{T} P(w_t | w_{t-1})
\]

Recurrent Neural Networks!

- RNNs tie the weights at each time step
- Condition the neural network on all previous words
- RAM requirement only scales with number of words

Recurrent Neural Network language model

We use the same set of \( W \) weights at all time steps!
Everything else is the same:

\[
h_t = \sigma \left( W^{(hh)} h_{t-1} + W^{(hx)} x_t \right)
\]

\[
\tilde{y}_t = \text{softmax} \left( W^{(S)} h_t \right)
\]

\[
P(x_{t+1} = v_j | x_1, \ldots, x_t) = \tilde{y}_{t,j}
\]

\( h_0 \in \mathbb{R}^{D_h} \) is some initialization vector for the hidden layer at time step 0

\( x[t] \) is the column vector of \( L \) at index \([t]\) at time \( t \)

\( W^{(hh)} \in \mathbb{R}^{D_h \times D_h} \quad W^{(hx)} \in \mathbb{R}^{D_h \times d} \quad W^{(S)} \in \mathbb{R}^{V \times D_h} \)

Recurrent Neural Network language model

Given list of word vectors: \( x_1, \ldots, x_T \)
At a single time step:

\[
h_t = \sigma \left( W^{(hh)} h_{t-1} + W^{(hx)} x_t \right)
\]

\[
\tilde{y}_t = \text{softmax} \left( W^{(S)} h_t \right)
\]

\[
P(x_{t+1} = v_j | x_1, \ldots, x_t) = \tilde{y}_{t,j}
\]

\( h_0 \in \mathbb{R}^{D_h} \) is some initialization vector for the hidden layer at time step 0

\( x[t] \) is the column vector of \( L \) at index \([t]\) at time \( t \)

\( W^{(hh)} \in \mathbb{R}^{D_h \times D_h} \quad W^{(hx)} \in \mathbb{R}^{D_h \times d} \quad W^{(S)} \in \mathbb{R}^{V \times D_h} \)

Objective function for language models

\( \hat{y}_t \in \mathbb{R}^{V} \) is a probability distribution over the vocabulary

Same cross entropy loss function but predicting words instead of classes

\[
J^{(1)}(\theta) = - \sum_{j=1}^{V} y_{t,j} \log \hat{y}_{t,j}
\]
**Training RNNs is hard**
- Multiply the same matrix at each time step during forward prop
- Ideally inputs from many time steps ago can modify output $y$

**The vanishing/exploding gradient problem**
- Multiply the same matrix at each time step during backprop
- Detailed derivations in the appendix of these slides!

**Why is the vanishing gradient a problem?**
- The error at a time step ideally can tell a previous time step from many steps away to change during backprop

**The vanishing gradient problem for language models**
- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word
- Example:
  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____

**Trick for exploding gradient: clipping trick**
- The solution first introduced by Mikolov is to clip gradients to a maximum value.

```
Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode
\[ \dot{g} = \frac{\dot{g}}{||\dot{g}||} \]
If $||\dot{g}|| \geq$ threshold then
\[ \dot{g} \leftarrow \text{threshold} \]
end if
```
- Makes a big difference in RNNs.

**For vanishing gradients: Initialization + ReLUs!**
- Initialize $W^{\text{ir}}$s to identity matrix $I$
- Set $f(z) = \text{rect}(z) = \max(z, 0)$
- Huge difference!
Today's Outline

- Word representation
- Recurrent Neural Networks (RNNs) and language modeling
- Bidirectional and deep RNNs
- Extensions on RNNs
  - Gated Recurrent Units (GRU)
  - Long short—term memories (LSTMs)

Sequence modeling for other tasks

- Classify each word into:
  - NER
  - Entity level sentiment in context
  - opinionated expressions

- Example application and slides from paper
  *Opinion Mining with Deep Recurrent Nets*
  by Irsoy and Cardie 2014

Opinion Mining with Deep Recurrent Nets

Goal: Classify each word as
direct subjective expressions (DSEs) and expressive subjective expressions (ESEs).

DSE: Explicit mentions of private states or speech events expressing private states

ESE: Expressions that indicate sentiment, emotion, etc. without explicitly conveying them.

Example Annotation

In BIO notation (tags either begin-of-entity (B_X) or continuation-of-entity (I_X)):
The committee, [as usual]_{LSE} has refused to make any statements}_{LSE}

\[
\begin{array}{cccccc}
\text{The} & \text{committee} & , & \text{as} & \text{usual} & \text{has} \\
\text{O} & \text{O} & \text{O} & \text{B}_{\text{ESE}} & \text{L}_{\text{ESE}} & \text{O} & \text{B}_{\text{DSE}} \\
\text{I}_{\text{DSE}} & \text{I}_{\text{DSE}} & \text{I}_{\text{DSE}} & \text{I}_{\text{DSE}} & \text{I}_{\text{DSE}} & \text{I}_{\text{DSE}} & \text{O} \\
\end{array}
\]

Approach: Recurrent Neural Network

- Notation from paper (so you get used to different ones)

\[
\begin{align*}
h_t &= f(W_x h_{t-1} + b) \\
y_t &= g(U h_t + c)
\end{align*}
\]

- \(x\) represents a token (word) as a vector.
- \(y\) represents the output label (B, I or O) — \(g = \text{softmax} \ 1\)
- \(h\) is the memory, computed from the past memory and current word. It summarizes the sentence up to that time.

Bidirectional RNNs

Problem: For classification you want to incorporate information from words both preceding and following

\[
\begin{align*}
h_t &= f(W_x h_{t-1} + b) \\
y_t &= g(U h_t + c)
\end{align*}
\]

\(h_t = \overline{h_t} \overline{h_t}\) now represents (summarizes) the past and future around a single token.
Deep Bidirectional RNNs

Each memory layer passes an intermediate sequential representation to the next.

Machine Translation (MT)

- Traditional MT:
  - A lot of human feature engineering
  - Very complex systems
  - Many different, independent machine learning problems

Deep learning to the rescue! ...

Maybe, we could translate directly with an RNN?

RNN Translation Model Extensions

1. Train different RNN weights for encoding and decoding

MT with RNNs – Simplest Model

Encoder: 
\[ h_t = \phi(h_{t-1}, x_t) = f(W^{(h)} h_{t-1} + W^{(x)} x_t) \]

Decoder: 
\[ y_t = \text{softmax}(W^{(y)} h_t) \]

Minimize cross entropy error for all target words conditioned on source words
\[ \max \frac{1}{N} \sum_{n=1}^{N} \log p(y^{(n)} | x^{(n)}) \]

It’s not quite that simple ;)

RNN Translation Model Extensions

Notation: Each input of \( \phi \) has its own linear transformation matrix. Simple: 
\[ h_t = \phi(h_{t-1}) = f(W^{(h)} h_{t-1}) \]

2. Compute every hidden state in decoder from
   - Previous hidden state (standard)
   - Last hidden vector of encoder \( c \)
   - Previous predicted output word \( y_{t-1} \)

\[ h_{D,t} = \phi_D(h_{t-1}, c, y_{t-1}) \]

Cho et al. 2014
Today’s Outline

• Word representation
• Recurrent Neural Networks (RNNs) and language modeling
• Bidirectional and deep RNNs
• Extensions on RNNs
  — Gated Recurrent Units (GRU)
  — Long short—term memories (LSTMs)

GRUs

• Standard RNN computes hidden layer at next time step directly:
  \[ h_t = f \left( W^{\text{hh}} h_{t-1} + W^{\text{hx}} x_t \right) \]
• GRU first computes an update gate (another layer) based on current input word vector and hidden state
  \[ z_t = \sigma \left( W^{\text{zx}} x_t + U^{\text{zx}} h_{t-1} \right) \]
  \[ r_t = \sigma \left( W^{\text{rx}} x_t + U^{\text{rx}} h_{t-1} \right) \]
• Compute reset gate similarly but with different weights
  \[ r_t = \sigma \left( W^{\text{rh}} x_t + U^{\text{rh}} h_{t-1} \right) \]

GRU intuition

• If reset is close to 0, ignore previous hidden state
  Allows model to drop information that is irrelevant in the future
• Update gate \( z_t \) controls how much of past state should matter now.
  • If \( z \) close to 1, then we can copy information in that unit through many time steps! Less vanishing gradient!
  • Units with short-term dependencies often have reset gates very active
GRU intuition

- Units with long term dependencies have active update gates $z$
- Illustration:

$$z_t = \sigma(W^{(z)} x_t + U^{(z)} h_{t-1})$$
$$r_t = \sigma(W^{(r)} x_t + U^{(r)} h_{t-1})$$
$$\tilde{h}_t = \tanh(W x_t + r_t \odot U h_{t-1})$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

Long-short-term-memories (LSTMs)

- We can make the units even more complex
- Allow each time step to modify
  - Input gate (current cell matters) $i_t = \sigma(W^{(i)} x_t + U^{(i)} h_{t-1})$
  - Forget (gate 0, forget past) $f_t = \sigma(W^{(f)} x_t + U^{(f)} h_{t-1})$
  - Output (how much cell is exposed) $o_t = \sigma(W^{(o)} x_t + U^{(o)} h_{t-1})$
  - New memory cell $\tilde{c}_t = \tanh(W^{(c)} x_t + U^{(c)} h_{t-1})$
  - Final memory cell: $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
  - Final hidden state: $h_t = o_t \odot \tanh(c_t)$

- Diagram:

```
http://deeplearning.net/tutorial/lstm.html
```

```
I was given a card by her in the garden
in the garden, she gave me a card
She gave me a card in the garden
She was given a card by me in the garden
in the garden, I gave her a card
I gave her a card in the garden
```

```
Mary admires John
John admires Mary
Mary is in love with John
John is in love with Mary
Mary respects John
John respects Mary
```
What we learned today

• Word representation
• Recurrent Neural Networks (RNNs) and language modeling
• Bidirectional and deep RNNs
• Extensions on RNNs
  – Gated Recurrent Units (GRU)
  – Long short—term memories (LSTMs)