Recap: Neural Probabilistic Language Model

- **Core idea**
  - Learn a shared distributed encoding (word embedding) for the words in the vocabulary.


Recap: word2vec

- **Goal**
  - Make it possible to learn high-quality word embeddings from huge data sets (billions of words in training set).

- **Approach**
  - Define two alternative learning tasks for learning the embedding:
    - "Continuous Bag of Words" (CBOW)
    - "Skip-gram"
  - Designed to require fewer parameters.

Recap: word2vec CBOW Model

- **Continuous BOW Model**
  - Remove the non-linearity from the hidden layer
  - Share the projection layer for all words (their vectors are averaged)

  \[ \text{Bag-of-Words model} \quad \text{(order of the words does not matter anymore)} \]

Recap: word2vec Skip-Gram Model

- **Continuous Skip-Gram Model**
  - Similar structure to CBOW
  - Instead of predicting the current word, predict words within a certain range of the current word.
  - Give less weight to the more distant words
Problems with 100k-1M outputs

- Weight matrix gets huge!
  - Example: CBOW model
  - One-hot encoding for inputs
  - Input-hidden connections are just vector lookups.
  - This is not the case for the hidden-output connections!
- Softmax gets expensive!
  - Need to compute normalization over 100k-1M outputs

Solution: Hierarchical Softmax

- Idea
  - Organize words in binary search tree, words are at leaves
  - Factorize probability of word \( w \), as a product of node probabilities along the path.
  - Learn a linear decision function \( y = v^T h \) at each node to decide whether to proceed with left or right child node.
  - Decision based on output vector of hidden units directly.

Topics of This Lecture

- Recurrent Neural Networks (RNNs)
  - Motivation
  - Intuition
- Learning with RNNs
  - Formalization
  - Comparison of Feedforward and Recurrent networks
  - Backpropagation through Time (BPTT)
- Problems with RNN Training
  - Vanishing Gradients
  - exploding Gradients
  - Gradient Clipping

Recurrent Neural Networks

- Up to now
  - Simple neural network structure: 1-to-1 mapping of inputs to outputs
- This lecture: Recurrent Neural Networks
  - Generalize this to arbitrary mappings

Application: Part-of-Speech Tagging

Legend: Click on the legend words to toggle highlighting. See the top of this page.

Application: Predicting the Next Word
Application: Machine Translation

<table>
<thead>
<tr>
<th>French words</th>
<th>English words</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>C</td>
<td>Z</td>
</tr>
<tr>
<td>&lt;S&gt;</td>
<td>&lt;E&gt;</td>
</tr>
</tbody>
</table>

I. Sutskever, O. Vinyals, Q. Le, *Sequence to Sequence Learning with Neural Networks*, NIPS 2014.

RNNs: Intuition

- Example: Language modeling
  - Suppose we had the training sequence “cat sat on mat”
  - We want to train a language model
    \[ p(\text{next word} | \text{previous words}) \]
  - First assume we only have a finite, 1-word history.
  - I.e., we want those probabilities to be high:
    \[ p(\text{cat} | <S>) \]
    \[ p(\text{sat} | \text{cat}) \]
    \[ p(\text{on} | \text{sat}) \]
    \[ p(\text{mat} | \text{on}) \]
    \[ p(<E> | \text{mat}) \]

- Turning this into an RNN (wait for it...)

- Turning this into an RNN (done!)

- Training this on a lot of sentences would give us a language model.
  - I.e., a way to predict
    \[ p(\text{next word} | \text{previous words}) \]
RNNs: Intuition

• Training this on a lot of sentences would give us a language model.

• I.e., a way to predict
  \[ p(\text{next word} \mid \text{previous words}) \]

sample!
RNNs: Intuition

• Training this on a lot of sentences would give us a language model.
• I.e., a way to predict \( p(\text{next word} | \text{previous words}) \)

---

Topics of This Lecture

• Recurrent Neural Networks (RNNs)
  - Motivation
  - Intuition
• Learning with RNNs
  - Formalization
  - Comparison of Feedforward and Recurrent networks
  - Backpropagation through Time (BPTT)
• Problems with RNN Training
  - Vanishing Gradients
  - Exploding Gradients
  - Gradient Clipping

---

RNNs: Introduction

• RNNs are regular NNs whose hidden units have additional forward connections over time.
  - You can unroll them to create a network that extends over time.
  - When you do this, keep in mind that the weights for the hidden units are shared between temporal layers.

---

Feedforward Nets vs. Recurrent Nets

• Imagine a feedforward network
  - Assume there is a time delay of 1 in using each connection.
  - This is very similar to how an RNN works.
  - Only change: the layers share their weights.

  \( \Rightarrow \) The recurrent net is just a feedforward net that keeps reusing the same weights.
Backpropagation with Weight Constraints

- It is easy to modify the backprop algorithm to incorporate linear weight constraints
  - To constrain $w_1 = w_2$, we start with the same initialization and then make sure that the gradients are the same:
    $$\nabla w_1 = \nabla w_2$$
  - We compute the gradients as usual and then use
    $$\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$$
    for both $w_1$ and $w_2$.

Backpropagation Through Time (BPTT)

- Formalization
  - Inputs $x_t$
  - Outputs $y_t$
  - Hidden units $h_t$
  - Initial state $h_0$
  - Connection matrices
    - $W_{xh}$
    - $W_{h}$
    - $W_{h}$
  - Configuration $h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + \beta)$

Recap: Backpropagation Algorithm

- Efficient propagation scheme
  - $y_i$ is already known from forward pass! (Dynamic Programming)
  - Propagate back the gradient from layer $j$ and multiply with $y_i$.

Backpropagation Through Time (BPTT)

- Error function
  - Computed over all time steps:
    $$E = \sum_{1 \leq t \leq T} E_t$$

Backpropagation Through Time (BPTT)

- Backpropagated gradient
  - For weight $w_{ij}$:
    $$\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial h_i} \frac{\partial h_i}{\partial w_{ij}}$$
Backpropagation Through Time (BPTT)

- **Backpropagated gradient**
  - For weight $w_{ij}$:
  - In general:
    \[
    \frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k < T} \left( \frac{\partial E_t}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}} \frac{\partial h_{k-1}}{\partial w_{ij}} \right)
    \]

- **Analyzing the terms**
  - For weight $w_{ij}$:
    \[
    \frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k < T} \left( \frac{\partial E_t}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}} \frac{\partial h_{k-1}}{\partial w_{ij}} \right)
    \]
  - Propagation term:
    \[
    \frac{\partial h_k}{\partial h_{k-1}} = \prod_{1 \leq i \leq k} \frac{\partial h_i}{\partial h_{i-1}}
    \]

**Topics of This Lecture**

- Recurrent Neural Networks (RNNs)
  - Motivation
  - Intuition
- Learning with RNNs
  - Formalization
  - Comparison of Feedforward and Recurrent networks
  - Backpropagation through Time (BPTT)
- Problems with RNN Training
  - Vanishing Gradients
  - Exploding Gradients
  - Gradient Clipping

**Problems with RNN Training**

- **Summary**
  - Backpropagation equations
    \[
    E = \sum_{1 \leq t \leq T} E_t
    \]
    \[
    \frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k < T} \left( \frac{\partial E_t}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}} \frac{\partial h_{k-1}}{\partial w_{ij}} \right)
    \]
    \[
    \frac{\partial h_k}{\partial h_{k-1}} = \prod_{1 \leq i \leq k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{1 \leq i \leq k} \mathbf{W}_{ii} \text{diag}(\sigma'(h_{i-1}))
    \]
  - Remaining issue: how to set the initial state $h_0$?
  - Learn this together with all the other parameters.

- **Training RNNs is very hard**
  - As we backpropagate through the layers, the magnitude of the gradient may grow or shrink exponentially
  - Exploding or vanishing gradient problem!
  - In an RNN trained on long sequences (e.g., 100 time steps) the gradients can easily explode or vanish.
  - Even with good initial weights, it is very hard to detect that the current target output depends on an input from many time-steps ago.
Exploding / Vanishing Gradient Problem

- Consider the propagation equations:
  \[
  \frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq s \leq T} \left( \frac{\partial E_s}{\partial h_s} \frac{\partial h_s}{\partial h_{t-s}} \frac{\partial h_{t-s}}{\partial w_{ij}} \right) \\
  \frac{\partial h_t}{\partial h_k} = \prod_{l \geq k} \frac{\partial h_l}{\partial h_{l-1}} = \prod_{l \geq k} W_{hl}^{(l)} \sigma'(h_{l-1}) \\
  = (W_{hl}^{(l)})^t
  \]
- if \( t \) goes to infinity and \( l = t - k \).

  \Rightarrow \text{We are effectively taking the weight matrix to a high power.}
  
  - The result will depend on the eigenvalues of \( W_{hl} \).
    - Largest eigenvalue > 1 \( \Rightarrow \) Gradients may explode.
    - Largest eigenvalue < 1 \( \Rightarrow \) Gradients will vanish.
  - This is very bad...

Why Is This Bad?

- Vanishing gradients in language modeling
  - 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 ____“

    \Rightarrow The RNN will have a hard time learning such long-range dependencies.

Gradient Clipping

- Trick to handle exploding gradients
  - If the gradient is larger than a threshold, clip it to that threshold.

Algorithm 1: Pseudo-code for norm clipping the gradients whenever they explode

\[
g \leftarrow \frac{g}{\|g\|} \\
\text{if } \|g\| \geq \text{threshold} \text{ then} \\
\quad g \leftarrow \frac{g}{\text{threshold}} \\
\text{end if}
\]

- This makes a big difference in RNNs

Gradient Clipping Intuition

- Example
  - Error surface of a single RNN neuron
  - High curvature walls
  - Solid lines: standard gradient descent trajectories
  - Dashed lines: gradients rescaled to fixed size

References and Further Reading

- RNNs