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.
**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!
  - State $h$ is not one-hot, and vocabulary size is 1M.
  - $W'_{N \times V}$ has $300 \times 1M$ entries
- Softmax gets expensive!
  - Need to compute normalization over 100k-1M outputs

**Recap: Hierarchical Softmax**

- Idea
  - Organize words in binary search tree, words are at leaves
  - Factorize probability of word $w_j$ as a product of node probabilities along the path.
  - Learn a linear decision function $y = v_n(w_j) h$ 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**

**Application: Predicting the Next Word**
**Perceptual and Sensory Augmented Computing**

**Advanced Machine Learning**

**Winter’16**

---

**Application: Machine Translation**

- **French words**: A → B → C → D
- **English words**: W → X → Y → Z

![Diagram of machine translation](image)

---

**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:
    - \(p(\text{cat} | <S>)\)
    - \(p(\text{sat} | \text{cat})\)
    - \(p(\text{on} | \text{sat})\)
    - \(p(\text{mat} | \text{on})\)
    - \(p(<E> | \text{mat})\)

---

**RNNs: Intuition**

- **Vanilla 2-layer classification net**
  - 10,001D class scores (Softmax over 10k words and a special <END> token)
  - \(y_1 = W_{h0}h_0\)
  - Hidden layer (e.g., 5000 vectors)
  - \(h_1 = \max \{0, W_{xh}x_1\}\)
  - Word embedding (3000 vector for each word)

![Diagram of vanilla 2-layer classification net](image)

---

**RNNs: Intuition**

- **Turning this into an RNN (wait for it...)**
  - 10,001D class scores (Softmax over 10k words and a special <END> token)
  - \(y_1 = W_{h0}h_0\)
  - Hidden layer (e.g., 5000 vectors)
  - \(h_1 = \max \{0, W_{xh}x_1\}\)
  - Word embedding (3000 vector for each word)

![Diagram of RNN](image)

---

**RNNs: Intuition**

- **Turning this into an RNN (done!)**
  - 10,001D class scores (Softmax over 10k words and a special <END> token)
  - \(y_1 = W_{h0}h_0\)
  - Hidden layer (e.g., 5000 vectors)
  - \(h_1 = \max \{0, W_{xh}x_1 + W_{h0}h_0\}\)
  - Word embedding (3000 vector for each word)

![Diagram of trained RNN](image)

---

**RNNs: Intuition**

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

![Diagram of trained RNN](image)
RNNs: Intuition

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

• I.e., a way to predict

\[ p(\text{next word} | \text{previous words}) \]

• Sample!
RNNs: Intuition

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

I.e., a way to predict $p(\text{next word} | \text{previous words})$

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

Samples <END>? Done!

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.

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 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.

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.

⇒ 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 \( y_3 = w_3 \), we start with the same initialization and then make sure that the gradients are the same:
    \[
    \nabla w_3 = \nabla w_2
    \]
  - We compute the gradients as usual and then use
    \[
    \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad \text{for both } w_1 \text{ and } w_2.
    \]

Recap: Backpropagation Algorithm

- Efficient propagation scheme
  - \( y_i \) is already known from forward pass! (Dynamic Programming)
  \[ \Rightarrow \text{Propagate back the gradient from layer } j \text{ and multiply with } y_i. \]

Backpropagation Through Time (BPTT)

- Formalization
  - Inputs \( x_t \)
  - Outputs \( y_t \)
  - Hidden units \( h_t \)
  - Initial state \( h_0 \)
  - Connection matrices
    - \( W_{xh} \)
    - \( W_{yh} \)
    - \( W_{hh} \)
  - Configuration
    \[ h_t = \sigma (W_{xh} x_t + W_{yh} h_{t-1} + b) \]
    \[ \hat{y}_t = \text{softmax} (W_{yh} h_t) \]

- Error function
  - Computed over all time steps:
    \[ E = \sum_{t=1}^{T} E_t \]

Backpropagation Through Time (BPTT)

- Backpropagated gradient
  - For weight \( w_{ij} \):
    \[ \frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial w_{ij}} + \frac{\partial E_{t-1}}{\partial w_{ij}} \frac{\partial h_{t-1}}{\partial w_{ij}} \]
Backpropagation Through Time (BPTT)

- Backpropagated gradient
  - For weight $w_{ij}$:
    - In general: $\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k \leq T} \left( \frac{\partial E_t}{\partial h_k} \frac{\partial h_k}{\partial h_{ik}} \frac{\partial h_{ik}}{\partial w_{ij}} \right) + \ldots$
  - This is the “immediate” partial derivative (with $h_{0i}$ as constant)

- Analyzing the terms
  - For weight $w_{ij}$:
    - This is the “immediate” partial derivative (with $h_{0i}$ as constant)
  - Propagation term:

**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**

- 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:
  \[
  \frac{\partial E_l}{\partial w_{ij}} = \sum_{1 \leq s \leq l} \left( \frac{\partial E_s}{\partial h_i} \frac{\partial h_i}{\partial h_k} \frac{\partial h_k}{\partial w_{ij}} \right)
  \]
  \[
  \frac{\partial h_i}{\partial h_k} = \prod_{l > k} \frac{\partial h_i}{\partial h_{l-1}} = \prod_{l > k} W_{hi}^d \sigma'(h_{l-1})
  \]
  \[
  = (W_{hi}^d)^l
  \]
- If \( l \) goes to infinity and \( l = 1 - k \).

\[ \Rightarrow \] We are effectively taking the weight matrix to a high power.
- The result will depend on the eigenvalues of \( W_{hi} \).
  - 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.

\[ \text{Algorithm 1} \quad \text{Pseudo-code for norm clipping the gradients whenever they explode} \]

\[
\begin{align*}
g & \leftarrow \frac{g}{\|g\|} \quad \text{if} \quad \|g\| \geq \text{threshold} \\
g & \leftarrow \text{threshold} \\
\end{align*}
\]
- 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