This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Gaussian Processes
- Learning with Latent Variables
  - Prob. Distributions & Approx. Inference
  - Mixture Models
  - EM and Generalizations
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, RNNs, RBMs, etc.

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)
  - Bag-of-Words model (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
- Implementation
  - Randomly choose a number \( R \in [1, C] \).
  - Use \( R \) words from history and \( R \) words from the future of the current word as correct labels.
  - \( R+R \) word classifications for each input.
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.
    - $W_{V \times V}$ 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 $w_0$ as a product of node probabilities along the path.
  - Learn a linear decision function $y = v_h(w_0)$ 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

Application: Part-of-Speech Tagging

Application: Predicting the Next Word
Application: Machine Translation

1. French words
2. English words

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
  - \[ y_4 = W_{y_4}h_4 \]
  - Hidden layer
    - (e.g., 5000 vectors)
    - \[ h_4 = \max [0, W_{\text{h4}}x_4] \]
  - Word embedding
    - (3000 vector for each word)

RNNs: Intuition

- Turning this into an RNN (done!)
  - 10,001D class scores
    - Softmax over 10k words and a special <END> token
  - \[ y_4 = W_{y_4}h_4 \]
  - Hidden layer
    - (e.g., 5000 vectors)
    - \[ h_4 = \max [0, W_{\text{h4}}x_4 + W_{\text{h4}0}h_3] \]
  - Word embedding
    - (3000 vector for each word)

RNNs: Intuition

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

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

• I.e., a way to predict

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

Slide credit: Andrej Karpathy, Fei-Fei Li
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 $w_1 = w_2$, we start with the same initialization and then make sure that the gradients are the same:
    \[ \nabla w_1 = \nabla w_2 \n \]
  - 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 $w_2$.

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 $W_{xh}$, $W_{hy}$, $W_{hh}$
  - Configuration $h_t = \sigma (W_{xh} x_t + W_{hy} h_{t-1} + b)$
    \[ \hat{y}_t = \text{softmax} (W_{yh} h_t) \]
  - Error function
    \[ E = \sum_{t=1}^{T} E_t \]

Backpropagation Through Time (BPTT)

- Backpropagated gradient
  - For weight $w_{ij}$
    \[ \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial h_t} \frac{\partial h_t}{\partial w_{ij}} \]
• Backpropagated gradient
  - For weight $w_{ij}$:
    $$ \frac{\partial E}{\partial w_{ij}} = \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial w_{ij}} + \frac{\partial E_{t-1}}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_{ij}} + \ldots $$
  - In general:
    $$ \frac{\partial E}{\partial w_{ij}} = \sum_{1 \leq k \leq T} \left( \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-k}} \frac{\partial h_{t-k}}{\partial w_{ij}} \right) $$

• Analyzing the terms
  - For weight $w_{ij}$:
    $$ \frac{\partial E}{\partial w_{ij}} = \sum_{1 \leq k \leq T} \left( \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-k}} \frac{\partial h_{t-k}}{\partial w_{ij}} \right) $$
  - This is the “immediate” partial derivative (with $h_0$ as constant)

• Summary
  - Backpropagation equations
    $$ E = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k \leq T} \left( \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-k}} \frac{\partial h_{t-k}}{\partial w_{ij}} \right) $$
    $$ \frac{\partial h_t}{\partial h_k} = \prod_{1 \leq l \leq k} \frac{\partial h_l}{\partial h_{l-1}} = \prod_{1 \leq l \leq k} W_{kl}^T \text{diag}(\sigma'(h_{l-1})) $$

> 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:

\[
\frac{\partial E}{\partial w_{ij}} = \sum_{t \leq T} \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_{ij}}
\]

\[
\frac{\partial h_t}{\partial h_{t-1}} = \prod_{t \geq k} \frac{\partial h_t}{\partial h_{t-1}} = \prod_{t \geq k} W_{hh}^{t-t_0} \sigma'(h_{t-1})
\]

- If \( t \) goes to infinity and \( l = t - k \).
- We are effectively taking the weight matrix to a high power.
- The result will depend on the eigenvalues of \( W_{hh} \).
  - 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 ____”
  - 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

```plaintext
if \( \| g \| \geq \text{threshold} \)
then
  \( g \leftarrow \frac{\text{threshold}}{\| g \|} \cdot g \)
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