Recap: Residual Networks

- Core component
  - Skip connections bypassing each layer
  - Better propagation of gradients to the deeper layers
  - This makes it possible to train (much) deeper networks.

Recap: R-CNN for Object Detection

- Core component
  - Skip connections bypassing each layer
  - Better propagation of gradients to the deeper layers
  - This makes it possible to train (much) deeper networks.
Recap: Faster R-CNN

- One network, four losses
  - Remove dependence on external region proposal algorithm.
  - Instead, infer region proposals from same CNN.
  - Feature sharing
  - Joint training
  - Object detection in a single pass becomes possible.

Recap: Fully Convolutional Networks

- CNN
  - FCN
  - Intuition
    - Think of FCNs as performing a sliding-window classification, producing a heatmap of output scores for each class

Recap: Semantic Image Segmentation

- Encoder-Decoder Architecture
  - Problem: FCN output has low resolution
  - Solution: perform upsampling to get back to desired resolution
  - Use skip connections to preserve higher-resolution information

Topics of This Lecture

- Recap
  - ResNets
  - Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- Outlook: Recurrent Neural Networks

Neural Networks for Sequence Data

- Up to now
  - Simple structure: Input vector → Processing → Output
- In the following, we will look at sequence data
  - Interesting new challenges
  - Varying input/output length, need to memorize state, long-term dependencies, ...
- Currently a hot topic
  - Early successes of NNs for text/language processing.
  - Very good results for part-of-speech tagging, automatic translation, sentiment analysis, etc.
  - Recently very interesting developments for video understanding, image-text modeling (e.g., creating image descriptions), and even single-image understanding (attention processes).

Motivating Example

- Predicting the next word in a sequence
  - Important problem for speech recognition, text autocorrection, etc.
- Possible solution: The trigram (n-gram) method
  - Take huge amount of text and count the frequencies of all triplets (n-tuples) of words.
  - Use those frequencies to predict the relative probabilities of words given the two previous words
  \[
  p(w_3 | w_2 = b, w_1 = a) = \frac{\text{count}(abc)}{\text{count}(aba)} \\
  p(w_3 | w_2 = b, w_1 = a) = \frac{\text{count}(abf)}{\text{count}(abf)}
  \]
  - State-of-the-art until not long ago...
Problems with N-grams

- **Problem: Scalability**
  - We cannot easily scale this to large \( N \).
  - The number of possible combinations increases exponentially.
  - So does the required amount of data.

- **Problem: Partial Observability**
  - With larger \( N \), many counts would be zero.
  - The probability is not zero, just because the count is zero.
  - Need to back off to (N-1)-grams when the count for N-grams is too small.
  - Necessary to use elaborate techniques, such as Kneser-Ney smoothing, to compensate for uneven sampling frequencies.

Let’s Try Neural Networks for this Task

- Important issues
  - How should we encode the words to use them as input?
  - What internal NN structure do we need?
  - How can we perform classification (softmax) with so many possible outputs?

Neural Probabilistic Language Model

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

Word Embedding

- Idea
  - Encode each word as a vector in a \( d \)-dimensional feature space.
  - Typically, \( V \sim 1M, d \in (50, 300) \)

- Learning goal
  - Determine weight matrix \( W_{V \times d} \) that performs the embedding.
  - Shared between all input words.

- Input
  - Vocabulary index \( x \) in 1-of-\( K \) encoding.
  - For each input \( x \), only one row of \( W_{V \times d} \) is needed.
  - \( W_{V \times d} \) is effectively a look-up table.

Visualization of the Resulting Embedding

- (part of a 2.5D map of the most common 2500 words)
Popular Word Embeddings

- **Open issue**
  - What is the best setup for learning such an embedding from large amounts of data (billions of words)?

- **Several recent improvements**
  - word2vec [Mikolov 2013]
  - GloVe [Pennington 2014]
  - Pretrained embeddings available for everyone to download.

**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.

**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)

**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.
Interesting property

- Embedding often preserves linear regularities between words
  - Analogy questions can be answered through simple algebraic operations with the vector representation of words.
- Example
  - What is the word that is similar to small in the same sense as bigger is to big?
  - For this, we can simply compute $X = \text{vec}('bigger') - \text{vec}('big') + \text{vec}('small')$.
  - Then search the vector space for the word closes to $X$ using the cosine distance.
  - Result (when words are well trained): vec('smaller').
- Other example
  - E.g., $\text{vec}('King') - \text{vec}('Man') + \text{vec}('Woman') \approx \text{vec}('Queen')$.

Evaluation on Analogy Questions

<table>
<thead>
<tr>
<th>Type of relationship</th>
<th>Word Pair 1</th>
<th>Word Pair 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common capital city</td>
<td>Athens</td>
<td>Greece</td>
</tr>
<tr>
<td>All capital cities</td>
<td>Astana</td>
<td>Kazakhstan</td>
</tr>
<tr>
<td>Currency</td>
<td>Angola</td>
<td>kwazania</td>
</tr>
<tr>
<td>City-in-state</td>
<td>Chicago</td>
<td>Illinois</td>
</tr>
<tr>
<td>Man-Woman</td>
<td>brother</td>
<td>sister</td>
</tr>
<tr>
<td>Adjective to adverb</td>
<td>apparent</td>
<td>rapidly</td>
</tr>
<tr>
<td>Opposite</td>
<td>possibly</td>
<td>ethically</td>
</tr>
<tr>
<td>Comparative</td>
<td>great</td>
<td>tough</td>
</tr>
<tr>
<td>Superlative</td>
<td>easy</td>
<td>toughest</td>
</tr>
<tr>
<td>Present Participle</td>
<td>think</td>
<td>luckiest</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>Switzerland</td>
<td>reading</td>
</tr>
<tr>
<td>Post tense</td>
<td>walking</td>
<td>reading</td>
</tr>
<tr>
<td>Plural nouns</td>
<td>mouse</td>
<td>swimming</td>
</tr>
<tr>
<td>Plural verbs</td>
<td>work</td>
<td>speak</td>
</tr>
</tbody>
</table>

Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Vector Dimension</th>
<th>Training words</th>
<th>Accuracy [%]</th>
<th>Training time [Hrs x CPU (notes)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSLM</td>
<td>100</td>
<td>4k</td>
<td>34.2</td>
<td>14 x 180</td>
</tr>
<tr>
<td>CBOW</td>
<td>1000</td>
<td>6k</td>
<td>57.3</td>
<td>2 x 140</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>1000</td>
<td>6k</td>
<td>66.1</td>
<td>65.1</td>
</tr>
</tbody>
</table>

- Results
  - word2vec embedding is able to correctly answer many of those analogy questions.
  - CBOW structure better for syntactic tasks
  - Skip-gram structure better for semantic tasks

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.
    - $W_{V \times F}$ has $300 \times 1M$ entries
    - All of those need to be updated by backprop.

Solution: Hierarchical Softmax

- Idea
  - Organize words in binary search tree, words are at leaves
  - Factorize probability of word $w_i$ as a product of node probabilities along the path.
  - Learn a linear decision function $y = v_{w_i} \cdot h$ 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

- Recap: CNN Architectures
- Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- Embeddings in Vision
  - Siamese networks
  - Triplet loss networks
- Outlook: Recurrent Neural Networks

Siamese Networks

- Similar idea to word embeddings
- Learn an embedding network that preserves (semantic) similarity between inputs
  - E.g., used for patch matching

Recap: Discriminative Face Embeddings

- Learning an embedding using a Triplet Loss Network
  - Present the network with triplets of examples
    - Apply triplet loss to learn an embedding $f()$ that groups the positive example closer to the anchor than the negative one.
    - $\|f(x^+)_i - f(x^0)_i\|^2 < \|f(x^-)_i - f(x^0)_i\|^2$
  - Used with great success in Google’s FaceNet face recognition

Triplet Loss – Practical Implementation

- Triplet loss formulation
  - $\mathcal{L}_\text{triplet}(\theta) = \sum_{x_i,y_i,\lambda_i} \max(0, \lambda_i + D_{x_i,y_i} - D_{x_i,\lambda_i})$
- Practical Issue: How to select the triplets?
  - The number of possible triplets grows cubically with the dataset size.
  - Most triplets are uninformative
  - Mining hard triplets becomes crucial for learning.
  - Actually want medium-hard triplets for best training efficiency
- Popular solution: Offline hard triplet mining
  - Process the dataset to find hard triplets
  - Use those for learning
  - Iterate

Triplet Loss – Practical Implementation (2)

- Popular solution: Offline hard triplet mining
  - Process the dataset to find hard triplets
  - Use those for learning
  - Iterate

Topics of This Lecture

- Recap: CNN Architectures
- Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- Embeddings in Vision
  - Siamese networks
  - Triplet loss networks
- Outlook: Recurrent Neural Networks
Outlook: Recurrent Neural Networks

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

References and Further Reading

- Neural Probabilistic Language Model
- word2vec
- GloVe
- Hierarchical Softmax

References: Other Embeddings

- Face Embeddings