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.

Topics of This Lecture

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

Recap: Convolutional Neural Networks

- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant features
  - Classification layer at the end

Recap: AlexNet (2012)

- Similar framework as LeNet, but
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data (10^7 images instead of 10^3)
  - GPU implementation
  - Better regularization and up-to-date tricks for training (Dropout)

Recap: VGGNet (2014/15)

- Main ideas
  - Deeper network
  - Stacked convolutional layers with smaller filters (+ nonlinearity)
  - Detailed evaluation of all components

- Results
  - Improved ILSVRC top-5 error rate to 6.7%.

Slide credit: Svetlana Lazebnik


Recap: GoogLeNet (2014)

- Ideas:
  - Learn features at multiple scales
  - Modular structure

Inception module + copies
Convolution Pooling Softmax Other
Auxiliary classification outputs for training the lower layers (deprecated)

Recap: Visualizing CNNs

Relevant: GoogLeNet

Topics of This Lecture

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

The Learned Features are Generic

- Experiment: feature transfer
  - Train AlexNet-like network on ImageNet
  - Chop off last layer and train classification layer on CalTech256
  - State of the art accuracy already with only 6 training images!

Discussion

- GoogLeNet
  - 12 - fewer parameters than AlexNet
  - ~5M parameters
  - Where does the main reduction come from?
  - From throwing away the fully connected (FC) layers.

Effect

- After last pooling layer, volume is of size $[7 \times 7 \times 1024]$
- Normally you would place the first 4096-D FC layer here (Many million params).
- Instead: use Average pooling in each depth slice:
  - Reduces the output to $[1 \times 1 \times 1024]$.
  - Performance actually improves by 0.6% compared to when using FC layers (less overfitting?)

Transfer Learning with CNNs

1. Train on ImageNet
2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

I.e., swap the Softmax layer at the end
Transfer Learning with CNNs

1. Train on ImageNet
2. Use the old weights as initialization, train the full network or only some of the higher layers.
3. If you have medium sized dataset, “finetune” instead:

- Results on PASCAL VOC Detection benchmark
  - Pre-CNN state of the art: 35.1% mAP
  - R-CNN: 53.7% mAP

Other Tasks: Detection

R-CNN: Regions with CNN features

1. Input
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

- Results on PASCAL VOC Detection benchmark
  - Pre-CNN state of the art: 35.1% mAP
  - R-CNN: 53.7% mAP

Other Tasks: Semantic Segmentation

[Farabet et al. ICML 2012, PAMI 2013]

Other Tasks: Face Verification

Y. Taigman, M. Yang, M. Ranzato, L. Wolf, DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR 2014

Commercial Recognition Services

- E.g., clarifai

Try it out with your own media

Upload an image or file under http://clarifai.com or give us a direct link to a file on the web.

*By using the above, you agree to our terms of service
Commercial Recognition Services

• Be careful when taking test images from Google Search.
  • Chances are they may have been seen in the training set...

Topics of This Lecture

• Recap: CNN Architectures
• 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 \rightarrow Processing \rightarrow 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
    \[ p(w_3 = c | w_2 = b, w_1 = a) = \text{count}(abc) \]
    \[ p(w_3 = d | w_2 = b, w_1 = a) = \text{count}(abd) \]
  - 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

"softmax" units (one per possible next word)

<table>
<thead>
<tr>
<th>Internal NN structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index of word at t-2</td>
</tr>
<tr>
<td>Index of word at t-1</td>
</tr>
</tbody>
</table>

• 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

- "softmax" units (one per possible next word)
- units that learn to predict the output word from features of the input words
- learned distributed encoding of word t-2
- learned distributed encoding of word t-1
- index of word at t-2
- index of word at t-1

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

Slide adapted from Geoff Hinton

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 lookup table.

Slide adapted from Geoff Hinton

Word Embedding: Full Network

- Train on large corpus of data, learn \( W_{V \times d} \).
- Shown to outperform n-grams by [Bengio et al., 2003].

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 $R \in [1, \mathcal{C}]$.
  • 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
    $X = \text{vec}(\text{"bigger"}) - \text{vec}(\text{"big"}) + \text{vec}(\text{"small"})$
  • Then search the vector space for the word closest to $X$ using the cosine distance.
  ⇒ Result (when words are well trained): vec("smaller").

• Other example
  • E.g., vec("King") - vec("Man") + vec("Woman") ≈ 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></td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>City-in-state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Man-Woman</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adjective to adverb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opposite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Superlative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Present Participle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nationality adjective</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Past tense</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plural nouns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plural verbs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>semantic</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjective to adverb</td>
<td>apparent</td>
<td>apparently</td>
</tr>
<tr>
<td>Opposite</td>
<td>possibly</td>
<td>impossibly</td>
</tr>
<tr>
<td>Comparative</td>
<td>great</td>
<td>greater</td>
</tr>
<tr>
<td>Superlative</td>
<td>easy</td>
<td>easiest</td>
</tr>
<tr>
<td>Present Participle</td>
<td>think</td>
<td>thinking</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>Switzerland</td>
<td>Swiss</td>
</tr>
<tr>
<td>Past tense</td>
<td>walking</td>
<td>walked</td>
</tr>
<tr>
<td>Plural nouns</td>
<td>mouse</td>
<td>mice</td>
</tr>
<tr>
<td>Plural verbs</td>
<td>work</td>
<td>works</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>syntactic</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjective to adverb</td>
<td>rapid</td>
<td>rapidly</td>
</tr>
<tr>
<td>Opposite</td>
<td>unethical</td>
<td>ethically</td>
</tr>
<tr>
<td>Comparative</td>
<td>tough</td>
<td>tougher</td>
</tr>
<tr>
<td>Superlative</td>
<td>lucky</td>
<td>luckier</td>
</tr>
<tr>
<td>Present Participle</td>
<td>read</td>
<td>reading</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>Switzerland</td>
<td>Swiss</td>
</tr>
<tr>
<td>Past tense</td>
<td>swimming</td>
<td>swam</td>
</tr>
<tr>
<td>Plural nouns</td>
<td>dollar</td>
<td>dollars</td>
</tr>
<tr>
<td>Plural verbs</td>
<td>speak</td>
<td>speaks</td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Dimensionality</th>
<th>Training words</th>
<th>Accuracy (%)</th>
<th>Training time [days CPU cores]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNLM</td>
<td>100</td>
<td>68</td>
<td>34.2</td>
<td>64.5</td>
</tr>
<tr>
<td>CBOW</td>
<td>1000</td>
<td>68</td>
<td>57.3</td>
<td>68.9</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>1000</td>
<td>68</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.
    - $W_{N \times 1}$ has 300 x 1M entries
    - All of those need to be updated by backprop.

Problems with 100k-1M outputs

- 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 $w_j$ as a product of node probabilities along the path.
  - Learn a linear decision function $y = v_j^T h$ 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

- Recap: CNN Architectures
- Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- 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**