LEARNING DISTRIBUTED REPRESENTATIONS
FOR STATISTICAL LANGUAGE MODELLING
Overview

1. Discrete data and distributed representations

2. Language modelling
   - Factored RBM language model
   - Log-bilinear language model
   - Hierarchical log-bilinear language model
Discrete data

• Discrete data: datapoints with discrete-valued attributes

• When such datapoints are high-dimensional, regression / classification / density estimation is hard:
  – Amounts to estimating entries of an exponentially large table
    - Attributes correspond to table dimensions
    - Attribute values correspond to indices for the dimensions
  – Data sparsity: little or no data available for most entries
  – No a priori smoothness constraint on table entries
  – No general way to generalize to new table entries
Distributed representations

• Observation: making a model less local often improves generalization.
  – In a continuous space: average over datapoints near the point of interest.
  – In a discrete space: not clear what to average over.
    - What does “near” mean?
    - No general concept of distance / neighbourhood.

• Working with smooth functions over continuous spaces results in automatic smoothing.
  – Similar inputs produce similar outputs

• Idea: map discrete attributes to real-valued vectors and learn a smooth function that maps the vectors to the desired output values.
  – Learn the attribute mapping jointly with the function.
  – Automatic generalization!
**Statistical language modelling**

- **Goal:** Model the joint distribution of words in a sentence.

- **Such a model can be used to**
  - predict the next word given several preceding ones
  - arrange bags of words into sentences
  - assign probabilities to documents

- **Applications:** speech recognition, machine translation, information retrieval.

- **Most statistical language models are based on the Markov assumption:**
  - The distribution of the next word depends on only $n$ words that immediately precede it.
  - This assumption is clearly wrong but useful – it makes the task much more tractable.
• $n$-gram models are simply conditional probability tables for $P(w_n|w_{1:n-1})$.
  – $w_n$ is the word to be predicted (the next word)
  – words $w_{1:n-1} = w_1, \ldots, w_{n-1}$ are called the context

• $n$-gram models are estimated by counting the number of occurrences of each possible word $n$-tuple and normalizing.
  – smoothing the estimates is essential for good performance
  – many different smoothing methods exist

• $n$-gram models are the most widely used statistical language models due to their simplicity and excellent performance.

• Curse of dimensionality: the number of model parameters is exponential in $n$. 
Several neural probabilistic language models based on distributed representations have been proposed.

Common approach:
- Represent each word with a real-valued feature vector
- Represent the context by the sequence of the context word feature vectors
- Train a neural network to output the distribution for the next word from the context representation
- Learn word feature vectors jointly with other neural network parameters

Neural language models can outperform $n$-gram language models, especially when little training data is available.

Main drawback: very long training and testing times.
Conditional RBM language model

• Use a restricted Boltzmann machine to model $P(w_n|w_{1:n-1})$
  – Capture the interaction between $w_n$ and $w_{1:n-1}$ through a vector of latent variables.
  – Represent words using low-dimensional real-valued vectors.
    - $R_w$ is the feature vector for word $w$.

• Energy function:

  $$E(w_n, h; w_{1:n-1}) = -\sum_{i=1}^{n} R_{wi}W_i h$$

  – $h$ is the vector of latent variables
  – $W_i$ is the interaction matrix between the feature vector for $w_i$ and the latent variables.
  – Normalization is done only over $w_n$.

• Both inference and prediction take time linear in the number of latent variables.
Log-bilinear model

- The log-bilinear (LBL) model is perhaps the simplest neural language model.

- Given the context $w_{1:n-1}$, the LBL model first predicts the representation for the next word $w_n$ by linearly combining the representations of the context words:

$$\hat{r} = \sum_{i=1}^{n-1} C_i r_{w_i}$$

- $r_w$ is the real-valued vector representing word $w$

- Then the distribution for the next word is computed based on the similarity between the predicted representation and the representations of all words in the vocabulary:

$$P(w_n = w|w_{1:n-1}) = \frac{\exp(\hat{r}^T r_w)}{\sum_j \exp(\hat{r}^T r_j)}.$$
• Computing the probability of the given next word requires considering all $N$ words in the vocabulary.
  – Need to consider all words because the word space is unstructured.

• Idea: Organize words in the vocabulary into a binary tree and exploit its structure to speed up normalization (Morin and Bengio, 2005).
  – Construct a binary tree over words
    - words are associated with leaf nodes
    - one word per leaf
  – Replace the $N$-way decision by a sequence of $O(\log N)$ binary decisions for predicting the next word.
    - Can achieve an exponential speedup if the tree is balanced!
To define a distribution over leaf nodes:
- Specify the probability of taking the left branch at each non-leaf node.
- The probability of a leaf node is the product of probabilities of the left/right decisions that lead from the root node to the leaf node.
Constructing trees over words

• The approach of Morin and Bengio:
  – Start with the WordNet IS-A hierarchy (which is a DAG)
  – Manually select one parent node per word
  – Use clustering to make the resulting tree binary
  – Use the Neural Probabilistic Language Model for making the left/right decisions

• Drawbacks:
  – Tree construction process uses expert knowledge
  – The resulting model does not work as well as its non-hierarchical counterpart

• Our approach:
  – Construct the word tree from data alone (no experts needed)
  – Allow each word to occur more than once in the tree
  – Use the simplified log-bilinear language model for making the left/right decisions
Hierarchical log-bilinear model

- Let \( d \) be the binary code that encodes the sequence of left-right decisions in the tree that lead to word \( w \).

- Each non-leaf node in the tree is given a feature vector.
  - Used for discriminating the words in the left subtree from those in the right subtree.

- The probability of taking the left branch at \( i^{th} \) node in the sequence is
  \[
  P(d_i = 1|q_i, w_{1:n-1}) = \sigma(\hat{r}^T q_i),
  \]
  - \( \hat{r} \) is computed as in the LBL model
  - \( q_i \) is the feature vector for the node

- The probability of \( w \) being the next word is
  \[
  P(w_n = w|w_{1:n-1}) = \prod_i P(d_i|q_i, w_{1:n-1}).
  \]
We would like to cluster words based on the distribution of contexts in which they occur.

This distribution is hard to estimate and work with due to the high dimensionality of the space of contexts. — same difficulties as with estimating $n$-gram models

To avoid this problem, we represent contexts using distributed representations and cluster words based on their expected predicted representation.

Constructing a tree over words:
1. Train a model using a (balanced) random tree over words.
2. Extract the word representations from the trained model.
3. Perform hierarchical clustering on the extracted representations.
Hierarchical clustering

- Hierarchical top-down clustering of feature vectors:
  - At each level, fit a mixture of two Gaussians with spherical covariances using EM to the current group of word representations.
  - Assign words to mixture components based on the component responsibilities.

- We considered several splitting rules:
  - BALANCED: Sort the responsibilities and make the split to ensure a balanced tree.
  - ADAPTIVE: Assign the word to the component with the greater responsibility.
  - ADAPTIVE(ε): Assign the word to a component if its responsibility for the word is at least 0.5-ε.
Dataset and evaluation

- APNews dataset:
  - collection of Associated Press news stories (16 million words)

- Preprocessing (Bengio et al.):
  - convert all words to lower case
  - map all rare words and proper nouns to special symbols
  - just under 18000 words in the vocabulary

- Models were compared based on the perplexity they assigned to the test set.

- Perplexity is the geometric average of $\frac{1}{P(w_n|w_1:n-1)}$. 
Preliminary comparison:
- 10M training set, 0.5M validation set, 0.5M test set
- Feature-based models have 100D feature vectors.
- FRBM have 1000 hidden units.
- $\text{KN}_n$ is a Kneser-Ney back-off $n$-gram model.

<table>
<thead>
<tr>
<th>Model type</th>
<th>Context size</th>
<th>Model test perplexity</th>
<th>Mixture test perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRBM</td>
<td>2</td>
<td>169.4</td>
<td>110.6</td>
</tr>
<tr>
<td>Temporal FRBM</td>
<td>2</td>
<td>127.3</td>
<td>95.6</td>
</tr>
<tr>
<td>Log-bilinear</td>
<td>2</td>
<td>132.9</td>
<td>102.2</td>
</tr>
<tr>
<td>Log-bilinear</td>
<td>5</td>
<td>124.7</td>
<td>96.5</td>
</tr>
<tr>
<td>Back-off GT3</td>
<td>2</td>
<td>135.3</td>
<td>–</td>
</tr>
<tr>
<td>Back-off KN3</td>
<td>2</td>
<td>124.3</td>
<td>–</td>
</tr>
<tr>
<td>Back-off GT6</td>
<td>5</td>
<td>124.4</td>
<td>–</td>
</tr>
<tr>
<td>Back-off KN6</td>
<td>5</td>
<td>116.2</td>
<td>–</td>
</tr>
</tbody>
</table>
• Final comparison:
  – 14M training set, 1M validation set, 1M test set
  – (H)LBL used 100D feature vectors and a context size of 5.
  – KN\_n is an interpolated Kneser-Ney \_n\_gram model.

<table>
<thead>
<tr>
<th>Model type</th>
<th>Tree generating algorithm</th>
<th>Test perplex.</th>
<th>Mixture perplex.</th>
<th>Fitted mix. perplexity</th>
<th>Minutes per epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLBL RANDOM</td>
<td>151.2</td>
<td>107.2</td>
<td>106.0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>HLBL BALANCED</td>
<td>131.3</td>
<td>99.9</td>
<td>99.7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>HLBL ADAPTIVE</td>
<td>127.0</td>
<td>98.3</td>
<td>98.2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>HLBL ADAPTIVE(0.25)</td>
<td>124.4</td>
<td>97.5</td>
<td>97.4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>HLBL ADAPTIVE(0.4)</td>
<td>123.3</td>
<td>97.2</td>
<td>97.1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>HLBL ADAPTIVE(0.4) × 2</td>
<td>115.7</td>
<td>95.3</td>
<td>95.3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>HLBL ADAPTIVE(0.4) × 4</td>
<td>112.1</td>
<td>94.4</td>
<td>94.3</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>LBL –</td>
<td>117.0</td>
<td>94.0</td>
<td>94.0</td>
<td>6420</td>
<td></td>
</tr>
<tr>
<td>KN2 –</td>
<td>174.2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>KN3 –</td>
<td>125.6</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>KN6 –</td>
<td>119.2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>
The effect of the context size

- The HLBL models were based on the ADAPTIVE(0.4) × 4 tree.
- KNn is an interpolated modified Kneser-Ney n-gram model.
The End
Number of predictions \( (P(w_n|w_{1:n-1})) \) on the test set as a function of the their magnitude. Bin \( i \) (for \( i = 1, ..., 7 \)) contains predictions between \( 10^{-i} \) and \( 10^{-i+1} \). Bin 8 contains predictions smaller than \( 10^{-7} \).
Contribution to the negative log-probability of the test set as a function of the prediction magnitude. Bin $i$ (for $i = 1, \ldots, 7$) contains predictions between $10^{-i}$ and $10^{-i+1}$. Bin 8 contains predictions smaller than $10^{-7}$. 
A fragment of a t-SNE embedding of the feature vectors (learned by an LBL model) of the most frequent 1000 words.
A fragment of a t-SNE embedding of the feature vectors (learned by an LBL model) of the least frequent 1000 words.
A fragment of a t-SNE embedding of the feature vectors (learned by an HLBL model) of the most frequent 1000 words.
A fragment of a t-SNE embedding of the feature vectors (learned by an HLBL model) of the least frequent 1000 words.