11.3. Word embeddings and translation

François Fleuret
https://fleuret.org/ammi-2018/
November 2, 2018
Word embeddings and CBOW
An important application domain for machine intelligence is Natural Language Processing (NLP).

- Speech and (hand)writing recognition,
- auto-captioning,
- part-of-speech tagging,
- sentiment prediction,
- translation,
- question answering.
An important application domain for machine intelligence is Natural Language Processing (NLP).

- Speech and (hand)writing recognition,
- auto-captioning,
- part-of-speech tagging,
- sentiment prediction,
- translation,
- question answering.

While language modeling was historically addressed with formal methods, in particular generative grammars, state-of-the-art and deployed methods are now heavily based on statistical learning and deep learning.
A core difficulty of Natural Language Processing is to devise a proper density model for sequences of words.

However, since a vocabulary is usually of the order of $10^4 - 10^6$ words, empirical distributions can not be estimated for more than triplets of words.
The standard strategy to mitigate this problem is to embed words into a geometrical space to take advantage of data regularities for further [statistical] modeling.
The standard strategy to mitigate this problem is to embed words into a geometrical space to take advantage of data regularities for further [statistical] modeling.

The geometry after embedding should account for synonymy, but also for identical word classes, etc. *E.g.* we would like such an embedding to make “cat” and “tiger” close, but also “red” and “blue”, or “eat” and “work”, etc.
The standard strategy to mitigate this problem is to embed words into a geometrical space to take advantage of data regularities for further [statistical] modeling.

The geometry after embedding should account for synonymy, but also for identical word classes, etc. E.g. we would like such an embedding to make “cat” and “tiger” close, but also “red” and “blue”, or “eat” and “work”, etc.

Even though they are not “deep”, classical word embedding models are key elements of NLP with deep-learning.
Let
\[ k_t \in \{1, \ldots, W\}, \quad t = 1, \ldots, T \]
be a training sequence of \( T \) words, encoded as IDs through a vocabulary of \( W \) words.
Let 

\[ k_t \in \{1, \ldots, W\}, \ t = 1, \ldots, T \]

be a training sequence of \( T \) words, encoded as IDs through a vocabulary of \( W \) words.

Given an embedding dimension \( D \), the objective is to learn vectors

\[ E_k \in \mathbb{R}^D, \ k \in \{1, \ldots, W\} \]

so that “similar” words are embedded with “similar” vectors.
A common word embedding is the Continuous Bag of Words (CBOW) version of word2vec (Mikolov et al., 2013a).
A common word embedding is the Continuous Bag of Words (CBOW) version of word2vec (Mikolov et al., 2013a).

In this model, they embedding vectors are chosen so that a word can be predicted from [a linear function of] the sum of the embeddings of words around it.
More formally, let $C \in \mathbb{N}^*$ be a “context size”, and

$$C_t = (k_{t-C}, \ldots, k_{t-1}, k_{t+1}, \ldots, k_{t+C})$$

be the “context” around $k_t$, that is the indexes of words around it.
The embeddings vectors

\[ E_k \in \mathbb{R}^D, \quad k = 1, \ldots, W, \]

are optimized jointly with an array

\[ M \in \mathbb{R}^{W \times D} \]

so that the predicted vector of \( W \) scores

\[ \psi(t) = M \sum_{k \in \mathcal{C}_t} E_k \]

is a good predictor of the value of \( k_t \).
Ideally we would minimize the cross-entropy between the vector of scores $\psi(t) \in \mathbb{R}^W$ and the class $k_t$.

$$\sum_t - \log \left( \frac{\exp \psi(t)_{k_t}}{\sum_{k=1}^{W} \exp \psi(t)_k} \right).$$

However, given the vocabulary size, doing so is numerically unstable and computationally demanding.
The “negative sampling” approach uses a loss estimated on the prediction for the correct class $k_t$ and only $Q \ll W$ incorrect classes $\kappa_{t,1}, \ldots, \kappa_{t,Q}$ sampled at random.

In our implementation we take the later uniformly in $\{1, \ldots, W\}$ and use the same loss as Mikolov et al. (2013b):

$$
\sum_t \log \left( 1 + e^{-\psi(t)k_t} \right) + \sum_{q=1}^{Q} \log \left( 1 + e^{\psi(t)\kappa_{t,q}} \right).
$$

We want $\psi(t)k_t$ to be large and all the $\psi(t)\kappa_{t,q}$ to be small.
Although the operation

\[ x \mapsto E_x \]

could be implemented as the product between a one-hot vector and a matrix, it is far more efficient to use an actual lookup table.
The PyTorch module `nn.Embedding` does precisely that. It is parametrized with a number $N$ of words to embed, and an embedding dimension $D$.

It gets as input an integer tensor of arbitrary dimension $A_1 \times \cdots \times A_U$, containing values in $\{0, \ldots, N - 1\}$ and it returns a float tensor of dimension $A_1 \times \cdots \times A_U \times D$.

If $w$ are the embedding vectors, $x$ the input tensor, $y$ the result, we have

$$y[a_1, \ldots, a_U, d] = w[x[a_1, \ldots, a_U]][d].$$
```python
>>> e = nn.Embedding(10, 3)
>>> x = torch.tensor([[1, 1, 2, 2], [0, 1, 9, 9]], dtype=torch.int64)
>>> e(x)
tensor([[ 0.0386, -0.5513, -0.7518],
        [ 0.0386, -0.5513, -0.7518],
        [-0.4033,  0.6810,  0.1060],
        [-0.4033,  0.6810,  0.1060],
        [-0.5543, -1.6952,  1.2366],
        [ 0.0386, -0.5513, -0.7518],
        [ 0.2793, -0.9632,  1.6280],
        [ 0.2793, -0.9632,  1.6280]])
```
Our CBOW model has as parameters two embeddings

\[ E \in \mathbb{R}^{W \times D} \quad \text{and} \quad M \in \mathbb{R}^{W \times D}. \]

Its forward gets as input a pair of integer tensors corresponding to a batch of size \( B \):

- \( c \) of size \( B \times 2C \) contains the IDs of the words in a context, and
- \( d \) of size \( B \times R \) contains the IDs, for each of the \( B \) contexts, of the \( R \) words for which we want the prediction score (that will be the correct one and \( Q \) negative ones).

It returns a tensor \( y \) of size \( B \times R \) containing the dot products.

\[
y[n,j] = \frac{1}{D} M_{d[n,j]} \cdot \left( \sum_i E_{c[n,i]} \right). \]
class CBOW(nn.Module):
    def __init__(self, voc_size = 0, embed_dim = 0):
        super(CBOW, self).__init__()
        self.embed_dim = embed_dim
        self.embed_E = nn.Embedding(voc_size, embed_dim)
        self.embed_M = nn.Embedding(voc_size, embed_dim)

    def forward(self, c, d):
        sum_w_E = self.embed_E(c).sum(1).unsqueeze(1).transpose(1, 2)
        w_M = self.embed_M(d)
        return w_M.matmul(sum_w_E).squeeze(2) / self.embed_dim
Regarding the loss, we can use `nn.BCEWithLogitsLoss` which implements

\[ \sum_t y_t \log(1 + \exp(-x_t)) + (1 - y_t) \log(1 + \exp(x_t)). \]

It takes care in particular of the numerical problem that may arise for large values of \( x_t \) if implemented “naively”.
Before training a model, we need to prepare data tensors of word IDs from a text file. We will use a 100Mb text file taken from Wikipedia and

- make it lower-cap,
- remove all non-letter characters,
- replace all words that appear less than 100 times with ‘*’,
- associate to each word a unique id.

From the resulting sequence of length $T$ stored in a integer tensor, and the context size $C$, we will generate mini-batches, each of two tensors

- a 'context' integer tensor $c$ of dimension $B \times 2C$, and
- a 'word' integer tensor $w$ of dimension $B$. 
If the corpus is “The black cat plays with the black ball.”, we will get the following word IDs:

\[
\text{the: 0, black: 1, cat: 2, plays: 3, with: 4, ball: 5.}
\]

The corpus will be encoded as

\[
\begin{array}{cccccccc}
\text{the} & \text{black} & \text{cat} & \text{plays} & \text{with} & \text{the} & \text{black} & \text{ball} \\
0 & 1 & 2 & 3 & 4 & 0 & 1 & 5 \\
\end{array}
\]
If the corpus is “The black cat plays with the black ball.”, we will get the following word IDs:

the: 0, black: 1, cat: 2, plays: 3, with: 4, ball: 5.

The corpus will be encoded as

<table>
<thead>
<tr>
<th>Words</th>
<th>IDs</th>
<th>c</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>the black cat</td>
<td>0 1 2 3 4</td>
<td>0,1,3,4</td>
<td>2</td>
</tr>
<tr>
<td>black cat plays</td>
<td>1 2 3 4 0</td>
<td>1,2,4,0</td>
<td>3</td>
</tr>
<tr>
<td>cat plays with the black</td>
<td>2 3 4 0 1</td>
<td>2,3,0,1</td>
<td>4</td>
</tr>
<tr>
<td>plays with the black ball</td>
<td>3 4 0 1 5</td>
<td>3,4,1,5</td>
<td>0</td>
</tr>
</tbody>
</table>
We can train the model for an epoch with:

```python
for k in range(0, id_seq.size(0) - 2 * context_size - batch_size, batch_size):
    c, w = extract_batch(id_seq, k, batch_size, context_size)
    d = torch.empty(w.size(0), 1 + nb_neg_samples, dtype = torch.int64)
    d.random_(voc_size)
    d[:, 0] = w
    target = torch.empty(d.size())
    target.narrow(1, 0, 1).fill_(1)
    target.narrow(1, 1, nb_neg_samples).fill_(0)
    output = model(c, d)
    loss = bce_loss(output, target)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```
Some nearest-neighbors for the cosine distance between the embeddings

\[ d(w, w') = \frac{E_w \cdot E_{w'}}{\|E_w\|\|E_{w'}\|}. \]

<table>
<thead>
<tr>
<th></th>
<th>bike</th>
<th></th>
<th>cat</th>
<th></th>
<th>powerful</th>
</tr>
</thead>
<tbody>
<tr>
<td>paris</td>
<td>0.61</td>
<td>bike</td>
<td>0.61</td>
<td>cats</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>parisian</td>
<td>0.61</td>
<td>bicycle</td>
<td>0.55</td>
<td>fortresses</td>
</tr>
<tr>
<td>france</td>
<td>0.59</td>
<td>bicycles</td>
<td>0.54</td>
<td>dog</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>0.55</td>
<td>bikes</td>
<td>0.49</td>
<td>kitten</td>
<td>0.55</td>
</tr>
<tr>
<td>bordeaux</td>
<td>0.53</td>
<td>biking</td>
<td>0.44</td>
<td>feline</td>
<td>0.52</td>
</tr>
<tr>
<td>toulouse</td>
<td>0.51</td>
<td>motorcycle</td>
<td>0.42</td>
<td>pet</td>
<td>0.51</td>
</tr>
<tr>
<td>vienna</td>
<td>0.51</td>
<td>cyclists</td>
<td>0.40</td>
<td>dogs</td>
<td>0.50</td>
</tr>
<tr>
<td>strasbourg</td>
<td>0.51</td>
<td>riders</td>
<td>0.40</td>
<td>kittens</td>
<td>0.49</td>
</tr>
<tr>
<td>munich</td>
<td>0.49</td>
<td>sled</td>
<td>0.39</td>
<td>hound</td>
<td>0.49</td>
</tr>
<tr>
<td>marseille</td>
<td>0.49</td>
<td>triathlon</td>
<td>0.39</td>
<td>squirrel</td>
<td>0.48</td>
</tr>
<tr>
<td>rouen</td>
<td>0.48</td>
<td>car</td>
<td>0.38</td>
<td>mouse</td>
<td>0.48</td>
</tr>
</tbody>
</table>

François Fleuret
An alternative algorithm is the skip-gram model, which optimizes the embedding so that a word can be predicted by any individual word in its context (Mikolov et al., 2013a).

Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

R words from the future of the current word as correct labels. This will require us to do \( R \times 2 \) word classifications, with the current word as input, and each of the \( R \) words as output. In the following experiments, we use \( C = 10 \).

4 Results

To compare the quality of different versions of word vectors, previous papers typically use a table showing example words and their most similar words, and understand them intuitively. Although it is easy to show that word \textit{France} is similar to \textit{Italy} and perhaps some other countries, it is much more challenging when subjecting those vectors in a more complex similarity task, as follows. We follow previous observation that there can be many different types of similarities between words, for example, word \textit{big} is similar to \textit{bigger} in the same sense that \textit{small} is similar to \textit{smaller}.

Example of another type of relationship can be word pairs \textit{big} - \textit{biggest} and \textit{small} - \textit{smallest} [20]. We further denote two pairs of words with the same relationship as a question, as we can ask: "What is the word that is similar to \textit{small} in the same sense as \textit{biggest} is similar to \textit{big}?"

Somewhat surprisingly, these questions can be answered by performing simple algebraic operations with the vector representation of words. To find a word that is similar to \textit{small} in the same sense as \textit{biggest} is similar to \textit{big}, we can simply compute vector \( X = \text{vector("biggest")} - \text{vector("big")} + \text{vector("small")} \). Then, we search in the vector space for the word closest to \( X \) measured by cosine distance, and use it as the answer to the question (we discard the input question words during this search). When the word vectors are well trained, it is possible to find the correct answer (word \textit{smallest}) using this method.

Finally, we found that when we train high dimensional word vectors on a large amount of data, the resulting vectors can be used to answer very subtle semantic relationships between words, such as a city and the country it belongs to, e.g. France is to Paris as Germany is to Berlin. Word vectors with such semantic relationships could be used to improve many existing NLP applications, such as machine translation, information retrieval and question answering systems, and may enable other future applications yet to be invented.
Trained on large corpora, such models reflect semantic relations in the linear structure of the embedding space. *E.g.*

\[ E[paris] - E[france] + E[italy] \approx E[rome] \]
Trained on large corpora, such models reflect semantic relations in the linear structure of the embedding space. \( E.g. \)

\[
E_{[\text{paris}]} - E_{[\text{france}]} + E_{[\text{italy}]} \approx E_{[\text{rome}]}
\]

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>France - Paris</td>
<td>Italy: Rome</td>
<td>Japan: Tokyo</td>
<td>Florida: Tallahassee</td>
</tr>
<tr>
<td>big - bigger</td>
<td>small: larger</td>
<td>cold: colder</td>
<td>quick: quicker</td>
</tr>
<tr>
<td>Miami - Florida</td>
<td>Baltimore: Maryland</td>
<td>Dallas: Texas</td>
<td>Kona: Hawaii</td>
</tr>
<tr>
<td>Einstein - scientist</td>
<td>Messi: midfielder</td>
<td>Mozart: violinist</td>
<td>Picasso: painter</td>
</tr>
<tr>
<td>Sarkozy - France</td>
<td>Berlusconi: Italy</td>
<td>Merkel: Germany</td>
<td>Koizumi: Japan</td>
</tr>
<tr>
<td>copper - Cu</td>
<td>zinc: Zn</td>
<td>gold: Au</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>Putin: Medvedev</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: Linux</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>IBM: McNealy</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>France: tapas</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>

(Mikolov et al., 2013a)
The main benefit of word embeddings is that they are trained with unsupervised corpora, hence possibly extremely large.

This modeling can then be leveraged for small-corpora tasks such as

- sentiment analysis,
- question answering,
- topic classification,
- etc.
Sequence-to-sequence translation
Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

(Sutskever et al., 2014)
English to French translation.

Training:

- corpus 12M sentences, 348M French words, 30M English words,
- LSTM with 4 layers, one for encoding, one for decoding,
- 160,000 words input vocabulary, 80,000 output vocabulary,
- 1,000 dimensions word embedding, 384M parameters total,
- input sentence is reversed,
- gradient clipping.

The hidden state that contains the information to generate the translation is of dimension 8,000.

Inference is done with a “beam search”, that consists of greedily increasing the size of the predicted sequence while keeping a bag of $K$ best ones.
Comparing a produced sentence to a reference one is complex, since it is related to their semantic content.

A widely used measure is the BLEU score, that counts the fraction of groups of one, two, three and four words (aka “n-grams”) from the generated sentence that appear in the reference translations (Papineni et al., 2002).

The exact definition is complex, and the validity of this score is disputable since it poorly accounts for semantic.
<table>
<thead>
<tr>
<th>Method</th>
<th>test BLEU score (ntst14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bahdanau et al. [2]</td>
<td>28.45</td>
</tr>
<tr>
<td>Baseline System [29]</td>
<td>33.30</td>
</tr>
<tr>
<td>Single forward LSTM, beam size 12</td>
<td>26.17</td>
</tr>
<tr>
<td>Single reversed LSTM, beam size 12</td>
<td>30.59</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 1</td>
<td>33.00</td>
</tr>
<tr>
<td>Ensemble of 2 reversed LSTMs, beam size 12</td>
<td>33.27</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 2</td>
<td>34.50</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 12</td>
<td><strong>34.81</strong></td>
</tr>
</tbody>
</table>

Table 1: The performance of the LSTM on WMT’14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

(Sutskever et al., 2014)
<table>
<thead>
<tr>
<th>Type</th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our model</strong></td>
<td>Ulrich UNK, membre du conseil d’administration du constructeur automobile Audi, affirme qu’il s’agit d’une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d’administration afin qu’ils ne soient pas utilisés comme appareils d’écoute à distance.</td>
</tr>
<tr>
<td><strong>Truth</strong></td>
<td>Ulrich Hackenberg, membre du conseil d’administration du constructeur automobile Audi, déclare que la collecte des téléphones portables avant les réunions du conseil, afin qu’ils ne puissent pas être utilisés comme appareils d’écoute à distance, est une pratique courante depuis des années.</td>
</tr>
<tr>
<td><strong>Our model</strong></td>
<td>“Les téléphones cellulaires, qui sont vraiment une question, non seulement parce qu’ils pourraient potentiellement causer des interférences avec les appareils de navigation, mais nous savons, selon la FCC, qu’ils pourraient interférer avec les tours de téléphone cellulaire lorsqu’ils sont dans l’air”, dit UNK.</td>
</tr>
<tr>
<td><strong>Truth</strong></td>
<td>“Les téléphones portables sont véritablement un problème, non seulement parce qu’ils pourraient éventuellement créer des interférences avec les instruments de navigation, mais parce que nous savons, d’après la FCC, qu’ils pourraient perturber les antennes-relais de téléphonie mobile s’ils sont utilisés à bord”, a déclaré Rosenker.</td>
</tr>
<tr>
<td><strong>Our model</strong></td>
<td>Avec la crémation, il y a un “sentiment de violence contre le corps d’un être cher”, qui sera “réduit à une pile de cendres” en très peu de temps au lieu d’un processus de décomposition qui “accompagnera les étapes du deuil”.</td>
</tr>
<tr>
<td><strong>Truth</strong></td>
<td>Il y a, avec la crémation, “une violence faite au corps aimé”, qui va être “réduit à un tas de cendres” en très peu de temps, et non après un processus de décomposition, qui “accompagnerait les phases du deuil”.</td>
</tr>
</tbody>
</table>

Table 3: A few examples of long translations produced by the LSTM alongside the ground truth translations. The reader can verify that the translations are sensible using Google translate.
Ulrich Hackenberg, membre du conseil d'administration du constructeur automobile Audi, déclare que la collecte des téléphones portables avant les réunions du conseil, afin qu'ils ne puissent pas être utilisés comme appareils d'écoute à distance, est une pratique courante depuis des années.

"Les téléphones portables sont véritablement un problème, non seulement parce qu'ils pourraient éventuellement créer des interférences avec les instruments de navigation, mais parce que nous savons, d'après la FCC, qu'ils pourraient perturber les antennes-relais de téléphonie mobiles s'ils sont utilisés à bord," a déclaré Rosenker.

Figure 3: Le graphique de gauche montre la performance de notre système en fonction de la longueur des phrases, où l'axe x correspond aux phrases de test triées par leur longueur et est marqué par les longueurs réelles des séquences. Il n'y a pas de dégradation sur les phrases de moins de 35 mots, il n'y a que peu de dégradation sur les phrases les plus longues. Le graphique de droite montre la performance de l’LSTM sur les phrases avec des mots de plus en plus rares, où l’axe x correspond aux phrases de test triées par leur “rank de fréquence moyenne des mots”.

There is no degradation on sentences with less than 35 words, there is only a minor degradation on the longest sentences. The right plot shows the LSTM’s performance on sentences with progressively more rare words, where the x-axis corresponds to the test sentences sorted by their “average word frequency rank”.

François Fleuret

AMMI – Introduction to Deep Learning / 11.3. Word embeddings and translation
Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than a single LSTM with a beam of size 12.

<table>
<thead>
<tr>
<th>Method</th>
<th>test BLEU score (ntst14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bahdanau et al. [2]</td>
<td>28.45</td>
</tr>
<tr>
<td>Baseline System [29]</td>
<td>33.30</td>
</tr>
<tr>
<td>Single forward LSTM, beam size 12</td>
<td>26.17</td>
</tr>
<tr>
<td>Single reversed LSTM, beam size 12</td>
<td>30.59</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 1</td>
<td>33.00</td>
</tr>
<tr>
<td>Ensemble of 2 reversed LSTMs, beam size 12</td>
<td>33.27</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 2</td>
<td>34.50</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 12</td>
<td>34.81</td>
</tr>
</tbody>
</table>

Table 2: Methods that use neural networks together with an SMT system on the WMT'14 English to French test set (ntst14).

<table>
<thead>
<tr>
<th>Method</th>
<th>test BLEU score (ntst14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cho et al. [5]</td>
<td>34.54</td>
</tr>
<tr>
<td>Best WMT'14 result [9]</td>
<td>37.00</td>
</tr>
<tr>
<td>Rescoring the baseline 1000-best with a single forward LSTM</td>
<td>35.61</td>
</tr>
<tr>
<td>Rescoring the baseline 1000-best with a single reversed LSTM</td>
<td>35.85</td>
</tr>
<tr>
<td>Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs</td>
<td>36.50</td>
</tr>
<tr>
<td>Oracle Rescoring of the Baseline 1000-best lists</td>
<td>∼45</td>
</tr>
</tbody>
</table>

3.7 Performance on long sentences

We were surprised to discover that the LSTM did well on long sentences, which is shown quantitatively in figure 3. Table 3 presents several examples of long sentences and their translations.

3.8 Model Analysis

Figure 2: The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words model. Notice that both clusters have similar internal structure.

(Sutskever et al., 2014)
The end
References


