12.3. Word embeddings and translation

François Fleuret

https://fleuret.org/dlc/
Word embeddings and CBOW
An important application domain for machine intelligence is Natural Language Processing (NLP).

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

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.

Note that most of state-of-the-art methods are nowadays (01.12.2021) based on transformers (Vaswani et al., 2017), which incorporate word embeddings trained end-to-end with the rest of the model.
A core difficulty of Natural Language Processing is to devise a proper density model for sequences of words.

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 and exploit 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 \( k_t \in \{1, \ldots, W\}, \ t = 1, \ldots, T \) be a training sequence of \( T \) words, encoded as IDs from a \( W \) words vocabulary.

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).

In this model, the embedding vectors are chosen so that a word can be [linearly] predicted from 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.

Notes

The “context size” $C$ should be at least 1.

The vector $C_t$ has for components the $2C$ indices of the $C$ words before and after word $t$. 
The embeddings vectors
\[ E_k \in \mathbb{R}^D, \ k = 1, \ldots, W, \]
are optimized jointly with an array
\[ M \in \mathbb{R}^{W \times D} \]
so that the vector of scores
\[ \psi(t) = M \sum_{k \in \mathcal{C}_t} E_k \in \mathbb{R}^W \]
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 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 latter uniformly in $\{1, \ldots, W\}$ and use the same loss as Mikolov et al. (2013b):

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

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.

Notes

If $x$ was a one-hot vector, it would be of dimension the size of the dictionary $W$ with zeros for all components, except the one at the index of the word it encodes that would be 1. So $x$ could be made of hundred of thousands of zeros and a single one, which would be very inefficient.
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

$$y[a_1, \ldots, a_U, d] = w[x[a_1, \ldots, a_U]][d].$$
In this example, we consider an embedding that maps 10 words to a space of dimension 3. The words are referred to with their index, between 0 and 9.

To illustrate the use of this module we apply it to two sequences (1, 1, 2, 2) and (0, 1, 9, 9).

The input $x$ to the embedding is a tensor of size $2 \times 4$, where the first sequence (here a sample) is on the first row, and the second sequence on the second row.

The embedding of these two sequences $y$ is of size $2 \times 4 \times 3$, where the last dimension is the dimension of the embedding space, here $\mathbb{R}^3$. 

François Fleuret
Our CBOW model has as parameters two embeddings \( E \in \mathbb{R}^{W \times D} \) and \( M \in \mathbb{R}^{W \times D} \).

Its forward gets as input a pair \((c, d)\) 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 \( R \) words for which we want predicted scores.

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().__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, keepdim = True).transpose(1, 2)
        w_M = self.embed_M(d)
        return w_M.bmm(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 (e.g. punctuation),
- 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:

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

The corpus will be encoded as

the black cat plays with the black ball

and the data and label tensors will be

<table>
<thead>
<tr>
<th>Words</th>
<th>IDs</th>
<th>c</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>the black cat plays with the black ball</td>
<td>0 1 2 3 4</td>
<td>0, 1, 3, 4</td>
<td>2</td>
</tr>
<tr>
<td>black cat plays with the black</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.randint(voc_size, (w.size(0), 1 + nb_neg_samples))
    d[:, 0] = w

    targets = torch.zeros(d.size())
    targets[:, 0] = 1.0

    output = model(c, d)
    loss = bce_loss(output, targets)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

**Notes**

`extract_batch` is a function which returns the indices for the context and the index of the word to predict, given the context size and the word in the middle.

`d` contains the indices of words which will be used to compute the loss, the first index `d[:, 0]` being the word to predict, and the rest being random negative words, as described in previous slides.

The target first component is 1 because it corresponds to the correct word we want to predict, and 0 for the other negative words.
As results, 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>parisi</th>
<th>bike</th>
<th>cat</th>
<th>fortress</th>
<th>powerful</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.61</td>
<td>0.61</td>
<td>0.55</td>
<td>0.61</td>
<td>0.47</td>
</tr>
<tr>
<td>0.59</td>
<td>0.51</td>
<td>0.54</td>
<td>0.55</td>
<td>0.44</td>
</tr>
<tr>
<td>0.55</td>
<td>0.51</td>
<td>0.49</td>
<td>0.55</td>
<td>0.44</td>
</tr>
<tr>
<td>0.53</td>
<td>0.49</td>
<td>0.44</td>
<td>0.52</td>
<td>0.40</td>
</tr>
<tr>
<td>0.51</td>
<td>0.47</td>
<td>0.42</td>
<td>0.51</td>
<td>0.40</td>
</tr>
<tr>
<td>0.51</td>
<td>0.43</td>
<td>0.40</td>
<td>0.50</td>
<td>0.39</td>
</tr>
<tr>
<td>0.51</td>
<td>0.42</td>
<td>0.40</td>
<td>0.49</td>
<td>0.38</td>
</tr>
<tr>
<td>0.49</td>
<td>0.41</td>
<td>0.39</td>
<td>0.49</td>
<td>0.38</td>
</tr>
<tr>
<td>0.49</td>
<td>0.41</td>
<td>0.39</td>
<td>0.48</td>
<td>0.38</td>
</tr>
<tr>
<td>0.48</td>
<td>0.41</td>
<td>0.38</td>
<td>0.48</td>
<td>0.37</td>
</tr>
</tbody>
</table>

**Notes**

Once the embedding has been trained, we can compute the cosine similarity between [the embedding \(E_w\) of] a given word \(w\) (for instance “paris”), and [the embeddings of] all the words in the dictionary, and list the most similar.

What we see is that even with this small dataset (**100MB**), the embedding already exhibits a nice semantic-rich behavior.

These embedding provides a key piece of technology to bridge those categorical objects as geometrical objects and manipulating them as such.
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).

The CBOW model presented so far aims at predicting a word given the embedding vectors of the words around it (see left part of the picture).

The skip-gram model aims at predicting the embedding vectors of the word around given the embedding vector of the word in the middle.

This model is harder to train to give better results.

(Mikolov et al., 2013a)
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)

Notes

The embedding vectors have captured several semantic concepts such as

- capital of countries,
- grammar of adjectives,
- first and last names,
- head of states,
- jobs,
- etc.
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
Sequences pose a challenge for DNNs because they require that the dimensionality of the inputs and outputs is known and fixed. In this paper, we show that a straightforward application of the Long Short-Term Memory (LSTM) architecture [16] can solve general sequence to sequence problems. The idea is to use one LSTM to read the input sequence, one time step at a time, to obtain large fixed-dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector (fig. 1). The second LSTM is essentially a recurrent neural network language model [28, 23, 30] except that it is conditioned on the input sequence. The LSTM’s ability to successfully learn on data with long range temporal dependencies makes it a natural choice for this application due to the considerable time lag between the inputs and their corresponding outputs (fig. 1).

There have been a number of related attempts to address the general sequence to sequence learning problem with neural networks. Our approach is closely related to Kalchbrenner and Blunsom [18] who were the first to map the entire input sentence to vector, and is related to Cho et al. [5] although the latter was used only for rescoring hypotheses produced by a phrase-based system. Graves [10] introduced a novel differentiable attention mechanism that allows neural networks to focus on different parts of their input, and an elegant variant of this idea was successfully applied to machine translation by Bahdanau et al. [2]. The Connectionist Sequence Classification is another popular technique for mapping sequences to sequences with neural networks, but it assumes a monotonic alignment between the inputs and the outputs [11].

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, 304M 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.

Notes

In practice, the output words are not generated one by one, but multiple candidates are kept at each time step to generate multiple possible sentences.
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>34.81</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)
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.

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 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.

"Les téléphones portables sont véritablement un problème, non seulement parce qu'ils pourraient potentiellement 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 téléphone mobile s'ils sont utilisés à bord", a déclaré Rosenker.

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".

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".

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.

(Sutskever et al., 2014)
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 potentiellement 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.

Il y a, avec la crémation, "une violence faite au corps aimé" qui va être "réduite à un tas de cendres" en très peu de temps, et non après un processus de décomposition, qui "accompagnerait les phases du deuil".

"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 cellulaires lorsque'ils sont dans l'air", dit UNK.

Figure 3: The left plot shows the performance of our system as a function of sentence length, where the x-axis corresponds to the test sentences sorted by their length and is marked by the actual sequence lengths. 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”.

(Sutskever et al., 2014)
Method test BLEU score (ntst14)

Bahdanau et al. [2] 28.45
Baseline System [29] 33.30
Single forward LSTM, beam size 12 26.17
Single reversed LSTM, beam size 12 30.59
Ensemble of 5 reversed LSTMs, beam size 1 33.00
Ensemble of 2 reversed LSTMs, beam size 12 33.27
Ensemble of 5 reversed LSTMs, beam size 2 34.50
Ensemble of 5 reversed LSTMs, beam size 12 34.81

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.

Method test BLEU score (ntst14)

Cho et al. [5] 34.54
Best WMT'14 result [9] 37.0
Rescoring the baseline 1000-best with a single forward LSTM 35.61
Rescoring the baseline 1000-best with a single reversed LSTM 35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs 36.50
Oracle Rescoring of the Baseline 1000-best lists ~45

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

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

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)
References


