Neural Networks for Named Entity Recognition

Viktor Hangya
(slides originally by Dr. Fabienne Braune)

CIS, LMU Munich

WS 2019-2020
Outline

1. Named Entity Recognition
2. Feedforward Neural Networks: recap
3. Neural Networks for Named Entity Recognition
4. Adding Pre-trained Word Embeddings
5. Sequentiality in NER
6. Bilingual Word Embeddings
Named Entity Recognition
Task

Find segments of entity mentions in input text and tag with labels.

Example inputs:
- *Trump attacks BMW and Mercedes*
- *U.N. official Ekeus heads for Baghdad*

Example labels (coarse grained):
- persons PER
- locations LOC
- organizations ORG
- names NAME
- other MISC
Rule-based approaches

- A collection of rules to detect entities
- High precision vs. low recall
- Interpretable
- Time consuming to build and domain knowledge is needed

(Fabio Ciravegna, University of Sheffield)
Classification-based approaches

Given input segment, train classifier to tell:
- Is this segment a Named Entity?
- Give the corresponding Tag

Classification task:

*Trump attacks BMW and Mercedes*

Is Trump a named entity?
Yes, it is a person (PER)

Desired outputs:
- *Trump PER attacks BMW ORG and Mercedes ORG*
- *U.N. ORG official Ekeus PER heads for Baghdad LOC*
### Labeled data

Example annotations (CoNLL-2003):

<table>
<thead>
<tr>
<th>Surface</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.N.</td>
<td>I-ORG</td>
</tr>
<tr>
<td>official</td>
<td>O</td>
</tr>
<tr>
<td>Ekeus</td>
<td>I-PER</td>
</tr>
<tr>
<td>heads</td>
<td>O</td>
</tr>
<tr>
<td>for</td>
<td>O</td>
</tr>
<tr>
<td>Baghdad</td>
<td>I-LOC</td>
</tr>
<tr>
<td></td>
<td>O</td>
</tr>
</tbody>
</table>

**Scheme**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Begin</th>
<th>Inside</th>
<th>End</th>
<th>Single</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOB</td>
<td>B-X</td>
<td>I-X</td>
<td>I-X</td>
<td>B-X</td>
<td>O</td>
</tr>
<tr>
<td>IOE</td>
<td>I-X</td>
<td>I-X</td>
<td>E-X</td>
<td>E-X</td>
<td>O</td>
</tr>
<tr>
<td>IOBES</td>
<td>B-X</td>
<td>I-X</td>
<td>E-X</td>
<td>S-X</td>
<td>O</td>
</tr>
</tbody>
</table>

(Colloberb et al., 2011)
Classification-based approaches

- Classifier combination with engineered features (Florian et al., 2003)
  - Manually engineer features
    - words
    - POS tags
    - prefixes and suffixes
    - large (external) gazetteer
  - 88.76 F1

- Semi-supervised learning with linear models (Ando and Zhang, 2005)
  - Train linear model on annotated data
  - Add non-annotated data
  - 89.31 F1
Classification-based approaches

- Differences to rule-based:
  - Feature sets vs. rules
  - Less domain knowledge is needed
  - Faster to adapt systems
  - Annotated data is needed

- Next: neural networks
  - even less manual work
Feedforward Neural Networks: Recap
Motivation

Linear models not suited to learn non-linear decision boundaries.

Neural networks can do that

→ Through composition of non-linear functions
→ Learn relevant features from (almost) raw text
  → No need for manual feature engineering
  → learned by network
Feedforward Neural Network

Computation of hidden layer $H$:

- $A_1 = \sigma(X \cdot \Theta_1)$
- $A_2 = \sigma(X \cdot \Theta_2)$
- $B_0 = 1$ (bias term)

Computation of output unit $h(X)$:

- $h(X) = \sigma(H \cdot \Theta_3)$
Non-linear activation function

The **sigmoid function** $\sigma(Z)$ is often used.
Learning features from raw input

(Lee et al., 2009)
Feedforward neural network

*Trump* attacks *BMW and Mercedes*

**Binary NER task:** Is the segment from position 1 to 2 a Named Entity?

**Neural network:** $h(X) = \sigma(H \cdot \Theta_n)$, with:

\[
H = \begin{bmatrix}
B_0 &= 1 \\
A_1 &= \sigma(X \cdot \Theta_1) \\
A_2 &= \sigma(X \cdot \Theta_2) \\
\vdots & \\
A_j &= \sigma(X \cdot \Theta_j)
\end{bmatrix}
\]

**Prediction:** If $h(X) > 0.5$, yes. Otherwise, no.
Feedforward Neural Network

If weights are all random output will be random

→ Predictions will be bad

→ Get the right weights
Getting the right weights

Training: Find weight matrices $U = (\Theta_1, \Theta_2)$ and $V = \Theta_3$ such that $h(X)$ is the correct answer as many times as possible.

→ Given a set $T$ of training examples $t_1, \cdots t_n$ with correct labels $y_i$, find $U = (\Theta_1, \Theta_2)$ and $V = \Theta_3$ such that $h(X) = y_i$ for as many $t_i$ as possible.
→ Computation of $h(X)$ called forward propagation
→ $U = (\Theta_1, \Theta_2)$ and $V = \Theta_3$ with error back propagation
Multi-class classification

- More than two labels
- Instead of “yes” and “no”, predict \( c_i \in C = \{ c_1, \cdots, c_k \} \)
- **NER**: Is this segment a location, name, person ...

- **Use k output units**, where \( k \) is number of classes
  - Output layer instead of unit
  - Use softmax to obtain value between 0 and 1 for each class
  - Highest value is right class
Multi-class classification

\[ w_1, w_2, w_3, w_4 \]

\[ A_1, \ldots, A_{100} \]

\[ Z_1, \ldots, Z_K \]

\[ h(X) \]

input \quad U \quad hidden \quad V \quad output
Neural Networks for NER
Feedforward Neural Network for NER

Training example: *Trump attacks BMW (ORG) and Mercedes*

Neural network input:
- Look at word window around BMW
  - → **Trump**$_{-2}$ **attacks**$_{-1}$ **BMW** and$_{1}$ **Mercedes**$_{2}$
  - → each word $w_i$ is represented as one-hot vector
  - → $w_i = [0, 1, 0, 0, ..., 0]$

Neural network training:
- Predict corresponding label (forward propagation)
  - → should be **organization** (ORG)
- Train weights by backpropagating error
Feedforward Neural Network for NER

- **Input**: one-hot word representations $w_i$
- **Hidden layer**: learns to detect higher level features
  - e.g.: *at* ... *pm*
- **Output**: predicted label

Diagram:
- Input: $w_1, w_2, w_3, w_4$
- Hidden layer: $A_1, \ldots, A_{100}$
- Output: $Z_1, \ldots, Z_K$
- Function: $h(X)$

**Network Architecture**:
- Input: $U$ (one-hot word representations)
- Hidden layer: $A_1, \ldots, A_{100}$
- Output: $V$ (predicted label)
Weight training

**Training:** Find weight matrices $U$ and $V$ such that $h(X)$ is the **correct answer** as many times as possible.

→ Given a set $T$ of training examples $t_1, \cdots t_n$ with **correct labels** $y_i$, find $U$ and $V$ such that $h(X) = y_i$ for as many $t_i$ as possible.

→ Computation of $h(X)$ with **forward propagation**

→ $U$ and $V$ with error **back propagation**
Backpropagation

Goal of training: adjust weights such that correct label is predicted

→ Error between correct label and prediction is minimal

Compute error at output:

Compare

- output: \( h(x^i) = [0.01, 0.1, 0.001, 0.95, \ldots, 0.01] \)
- correct label: \( y^i = [0, 0, 1, 0, \ldots, 0] \)

\[
E = \frac{1}{2} \sum_{j=1}^{n} (y^i_j - h(x^i)_j)^2 \text{ (mean squared)}
\]

Search influence of weight on error:

\[
\frac{\partial E}{\partial w_{ij}}
\]

\( w_{ij} \): single weight in weight matrix
Backpropagation:

→ $E$ needs to go through output neuron.

→ Chain rule: $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}$
Weight training

Gradient descent: for each batch of training examples

1. Forward propagation to get predictions
2. Backpropagation of error
   - Gives gradient of $E$ given input
3. Modify weights
4. Goto 1 until convergence
Outcome

- Hidden layer is able to learn higher level features of words
  - *Cars are produced at BMW*
- Not enough to get good performance
- A simple index does not carry much information about a given word
  - \( w_{BMW} = [1, 0, 0, 0, ..., 0] \)
  - \( w_{Mercedes} = [0, 1, 0, 0, ..., 0] \)
  - \( w_{happiness} = [0, 0, 1, 0, ..., 0] \)

- This would be better
  - \( w_{BMW} = [1, 0, 0, 0, ..., 0] \)
  - \( w_{Mercedes} = [1, 0, 0, 0, ..., 0] \)
  - \( w_{happiness} = [0, 0, 1, 0, ..., 0] \)
Lookup (Embedding) Layer

- Learn features for words as well
- Similar words have similar features
- Lookup table layer:
  - embeds each one-hot encoded word $w_i$
  - to a feature vector $LT_i$

- $w_{BMW} = [0.5, 0.5, 0.0, 0.0, ... , 0.0]$
- $w_{Mercedes} = [0.5, 0.0, 0.5, 0.0, ... , 0.0]$
Dot product with (trained) weight vector

\[ W = \{ \text{the, cat, on, table, chair} \} \]

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
0
\end{bmatrix} \quad \begin{bmatrix}
0.02 & 0.1 & 0.05 & 0.03 & 0.01 \\
0.15 & 0.2 & 0.01 & 0.02 & 0.11 \\
0.03 & 0.1 & 0.04 & 0.04 & 0.12 \\
\end{bmatrix}
\]

\[ LT_{\text{table}} = w_{\text{table}} \cdot C^T = \begin{bmatrix}
0.03 \\
0.02 \\
0.04
\end{bmatrix} \]

Words get mapped to lower dimension
→ Hyperparameter to be set
Feedforward Neural Network with Lookup Table

\[ w_1 \rightarrow LT_1 \rightarrow A_1 \rightarrow Z_1 \]
\[ w_2 \rightarrow LT_2 \rightarrow A_1 \rightarrow Z_1 \]
\[ w_3 \rightarrow LT_3 \rightarrow A_1 \rightarrow Z_1 \]
\[ w_4 \rightarrow LT_4 \rightarrow A_1 \rightarrow Z_1 \]

\[ \text{word} \quad C \quad \text{word feats} \quad U \quad \text{hidden} \quad V \quad \text{output} \]

C is shared!
Dot product with (initial) weight vector

\[ W = \{ \text{the, cat, on, table, chair} \} \]

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
0
\end{bmatrix}
\]

\[
C =
\begin{bmatrix}
0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\
0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\
0.01 & 0.01 & 0.01 & 0.01 & 0.01
\end{bmatrix}
\]

\[
LT_{\text{table}} = w_{\text{table}} \cdot C^T =
\begin{bmatrix}
0.01 \\
0.01 \\
0.01
\end{bmatrix}
\]

Feature vectors same for all words.
Weight training

Training: Find weight matrices $C$, $U$ and $V$ such that $h(X)$ is the **correct answer** as many times as possible.

→ Given a set $T$ of training examples $t_1, \ldots, t_n$ with **correct labels** $y_i$, find $C$, $U$ and $V$ such that $h(X) = y_i$ for as many $t_i$ as possible.

→ Computation of $h(X)$ with **forward propagation**

→ $C$, $U$ and $V$ with **error back propagation**

→ **Lookup matrix** $C$ trained with NER training data

→ **Word feature vectors** are trained towards NER
Results

Classifier combination with engineered features (Florian et al. 2003)
- 88.76 F1

Semi-supervised learning with linear models (Ando and Zhang 2005)
- 89.31 F1

Feedforward Neural Networks for NER (Collobert et al., 2011):
- With raw words 81.74
NER trained word embeddings

Word embeddings trained on NER task

- (Collobert et al. 2011)

→ Small amount of annotated data.

- Closest words to France
  - Persuade
  - Faw
  - Blackstock

- Closest words to XBOX
  - Decadent
  - Divo
  - Versus
Adding Pre-trained Word Embeddings
Word Embeddings

- Representation of words in vector space
Word Embeddings

- Similar words are close to each other
  \[\rightarrow \text{Similarity is the cosine of the angle between two word vectors}\]
Learning word embeddings

Count-based methods:
- Compute cooccurrence statistics
- Learn high-dimensional representation
- Map sparse high-dimensional vectors to small dense representation
- Matrix factorization approaches: SVD

Neural networks:
- Predict a word from its neighbors
- Learn (small) embedding vectors
- Word2Vec: CBOW and skipgram Mikolov et al. (2013)
- Language Modeling Task
- ELMo, BERT Peters et al. (2018); Devlin et al. (2018)
Learning word embeddings with CBOW

Training example:  *Trump attacks BMW and Mercedes*

\[
\begin{align*}
L_{T_{t-2}} &= w_{t-2} \\
L_{T_{t-1}} &= w_{t-1} \\
L_{T_{t+1}} &= w_{t+1} \\
L_{T_{t+2}} &= w_{t+2}
\end{align*}
\]

\[
\sum_{C} \text{word feats} \Rightarrow U
\]

\[
\sum \text{word feats} \Rightarrow U
\]
Learning word embeddings with skip-gram

Training example: Trump attacks BMW and Mercedes

\[ W_t \rightarrow L_t \rightarrow W_{t-2} \rightarrow W_{t-1} \rightarrow W_{t+1} \rightarrow W_{t+2} \]

\[ U \rightarrow \text{word feats} \rightarrow C \rightarrow \text{word} \]
Learning word embeddings with Language Modeling

Training example: *Trump attacks BMW and Mercedes*

\[
\begin{align*}
& \mathbf{w}_{t-4} \rightarrow L T_{t-4} \\
& \mathbf{w}_{t-3} \rightarrow L T_{t-3} \\
& \mathbf{w}_{t-2} \rightarrow L T_{t-2} \\
& \mathbf{w}_{t-1} \rightarrow L T_{t-1} \\
& A_1 \rightarrow \mathbf{w}_t \\
\end{align*}
\]

\[
\begin{align*}
& \text{word} \quad C \quad \text{word feats} \quad U \quad V
\end{align*}
\]
Word Embeddings for NER

- Train word embeddings in advance:
  - Use large amounts of non-annotated data
  - No need for NER training data
  - Labels are words $w_t$

- Replace lookup table $C$ (randomly initialized) with $C$ (pre-trained)
NER trained word embeddings

Word embeddings trained on NER task

- (Collobert et al. 2011)

→ Small amount of annotated data.

- Closest words to France
  - Persuade
  - Faw
  - Blackstock

- Closest words to XBOX
  - Decadent
  - Divo
  - Versus
NER trained word embeddings

Pre-trained word embeddings trained
→ Large amount of non-annotated data.

- Closest words to France
  - Austria
  - Belgium
  - Germany

- Closest words to XBOX
  - Amiga
  - Playstation
  - MSX
Results

Classifier combination with engineered features (Florian et al. 2003)
- 88.76 F1

Semi-supervised learning with linear models (Ando and Zhang 2005)
- 89.31 F1

Feedforward Neural Networks for NER (Collobert et al. 2011):
- With raw words 81.74
- With pre-trained word embeddings 88.67
- Using a gazetteer 89.59
Results

- **Pre-trained** word embeddings yield significant improvements
- Hidden layer is able to learn higher level features of words
  - *Cars are produced at BMW*
- Word features:
  - $w_{BMW} = [0.5, 0.5, 0.0, 0.0, ..., 0.0]$
  - $w_{Mercedes} = [0.5, 0.0, 0.5, 0.0, ..., 0.0]$
  - $w_{happiness} = [0.0, 0.0, 0.0, 1.0, ..., 0.0]$
- It also helps the problem of out-of-vocabulary words

- The power is in exploiting large unlabeled data
- Instead of relying only on small labeled data
Sequence Tagging with RNNs and CRFs
NER as sequence tagging

- Sequential input
  - Classification approaches (linear or NN) looked at a window around the input word
  - Limitation of window size
    - too small $\rightarrow$ loosing information
    - too large $\rightarrow$ noise or data scarcity

  Let’s have a party at JFK

  - Read words sequentially and keep relevant information only

- Sequence of tags
  - IOB format: beginning and inside tags
  - Some tags shouldn’t follow each other
  - Output labels sequentially word-by-word

  The seminar starts tomorrow 4pm
Recurrent Neural Network (RNN)

(Huang et al., 2015)

- Reads the input sequentially
- At time step $t$:
  - $h_t = f(h_{t-1}, x_t; \theta_1)$
    - e.g. $h_t = \sigma(h_{t-1} \ast U + x_t \ast V)$
  - $o_t = g(h_t; \theta_2)$
    - e.g. $o_t = \sigma(h_t \ast W)$
- Parameters are shared for each time step
- Multiple variations: LSTM, GRU, etc.
RNNs for NER

- **Input:** words
- **Lookup layer**
  - learn embeddings from scratch
  - or used pre-trained embeddings
- **Probabilities of each NER tag**
- **Example:** *I’m traveling to the EU*

*(Huang et al., 2015)*
Bidirectional RNNs

(Huang et al., 2015)

The **EU** is very far from the **US**
- Read the input both from left-to-right and right-to-left
- Concatenate the hidden states to get the output
  \[ o_t = g(\vec{h_t}; \vec{h_t}; \theta_2) \]
Conditional Random Fields (CRF)

- Tag at time step $t$ should be dependent on the RNN output at $t$ and the tag at $t - 1$ as well.
- CRF adds (soft) constraints on the final predicted tags ensuring they are valid given previous tags.
  - Transition matrix $T_{i,j}$: probability of tag $j$ given that previous tag was $i$.

(Huang et al., 2015)
### CRF transition matrix

<table>
<thead>
<tr>
<th>From \ To</th>
<th>O</th>
<th>B-LOC</th>
<th>I-LOC</th>
<th>B-MISC</th>
<th>I-MISC</th>
<th>B-ORG</th>
<th>I-ORG</th>
<th>B-PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>3.281</td>
<td>2.204</td>
<td>0.0</td>
<td>2.101</td>
<td>0.0</td>
<td>3.468</td>
<td>0.0</td>
<td>2.325</td>
</tr>
<tr>
<td>B-LOC</td>
<td>-0.259</td>
<td>-0.098</td>
<td>4.058</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.212</td>
</tr>
<tr>
<td>I-LOC</td>
<td>-0.173</td>
<td>-0.609</td>
<td>3.436</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>B-MISC</td>
<td>-0.673</td>
<td>-0.341</td>
<td>0.0</td>
<td>0.0</td>
<td>4.069</td>
<td>-0.308</td>
<td>0.0</td>
<td>-0.331</td>
</tr>
<tr>
<td>I-MISC</td>
<td>-0.803</td>
<td>-0.998</td>
<td>0.0</td>
<td>-0.519</td>
<td>4.977</td>
<td>-0.817</td>
<td>0.0</td>
<td>-0.611</td>
</tr>
<tr>
<td>B-ORG</td>
<td>-0.096</td>
<td>-0.242</td>
<td>0.0</td>
<td>-0.57</td>
<td>0.0</td>
<td>-1.012</td>
<td>4.739</td>
<td>-0.306</td>
</tr>
<tr>
<td>I-ORG</td>
<td>-0.339</td>
<td>-1.758</td>
<td>0.0</td>
<td>-0.841</td>
<td>0.0</td>
<td>-1.382</td>
<td>5.062</td>
<td>-0.472</td>
</tr>
<tr>
<td>B-PER</td>
<td>-0.4</td>
<td>-0.851</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-1.013</td>
<td>0.0</td>
<td>-0.937</td>
</tr>
<tr>
<td>I-PER</td>
<td>-0.676</td>
<td>-0.47</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.659</td>
</tr>
</tbody>
</table>

**CRF State Transition Matrix**

(Image taken from https://eli5.readthedocs.io sklearn tutorial)
Prediction: tag sequence probability is calculated using RNN and transition probabilities (Viterbi algorithm)
Results

Classifier combination with engineered features (Florian et al. 2003)
- 88.76 F1

Semi-supervised learning with linear models (Ando and Zhang 2005)
- 89.31 F1

Feedforward Neural Networks for NER (Collobert et al. 2011):
- With raw words 81.74
- With pre-trained word embeddings 88.67
- Using a gazetteer 89.59

BI-LSTM-CRF
- 90.10
Bilingual Word Embeddings
Bilingual transfer learning

- For many low-resource languages we do not have enough training data for NER
- Use knowledge from resource rich languages
- Translate data to the target language
  - Parallel data is needed for the translation system
- Target language words are OOVs for a system trained on the source language
  - Similarity of source and target words $\rightarrow$ bilingual word embeddings
Bilingual Word Spaces

Representation of words in two languages in same semantic space:

→ **Similar** words are **close** to each other
→ **Given by cosine**

![Diagram showing Bilingual Word Spaces](image)
Learning Bilingual Word Embeddings

- Learn bilingual embeddings from parallel data
  Need for parallel data

- Learn bilingual embeddings or lexicon from document-aligned data
  Vulic and Moens (2015); Vulic and Korhonen (2016)
  Need document-aligned data

- Learn monolingual word embeddings and map using seed lexicon
  Mikolov et al. (2013); Faruqui and Dyer (2014); Lazaridou et al. (2015)
  Need seed lexicon
Post-hoc mapping (with seed lexicon)

- Learn monolingual word embeddings
- Learn a linear mapping $W$

\[ \begin{align*}
\text{poor} & \quad \text{disease} \\
\text{silver} & \quad \text{rich} \\
\text{Reich} & \quad \text{Silber} \\
\text{Gesellschaft} & \quad \text{Krankheit} \\
\text{Arm} & \quad \text{poor} \\
\end{align*} \]
Post-hoc mapping

- Project source words into target space

- Poor
- Rich
- Silver
- Silber
- Disease
- Gesellschaft
- Krankheit
- Society
- Arm
- Poor

Viktor Hangya (CIS)
Neural Networks for Named Entity Recognition
WS 2019-2020
Post-hoc Mapping with seed lexicon

1. Train **monolingual** word embeddings (Word2vec) in **English**
   ▶ Need **English** monolingual data

2. Train **monolingual** word embeddings (Word2vec) in **German**
   ▶ Need **German** monolingual data

3. Learn mapping $\mathbf{W}$ using a seed lexicon
   ▶ Need a list of **5000 English words and their translation**
Learning $W$ with Regression

Regression (Mikolov et al. (2013))

$$W^* = \arg \min_W \sum_{i}^{n} ||x_i W - y_i||^2$$

$x_i$ : embedding of i-th source (English) word in the seed lexicon.

$y_i$ : embedding of i-th target (German) word in the seed lexicon.
Learning $W$ with Ridge Regression

Regression (Mikolov et al. (2013))

$$W^* = \arg\min_W \sum_{i=1}^{n} ||x_iW - y_i||^2$$

- Predict projection $y^*$ by computing $x_iW$
- Compute squared error between $y^*$ and $y_i$
  - Correct translation $t_i$ given in seed lexicon
  - Vector representation $y_i$ is given by embedding of $t_i$
- Find $W$ such that squared error over training set is minimal
Bilingual lexicon induction

- Task to evaluate bilingual word embeddings intrinsically
- Given a set of source words, find the corresponding translations:
  - Given silver, find its vector in the BWE
  - Retrieve the German word whose vector is closest (cosine distance)
Bilingual lexicon induction with ridge regression

**Data:** WMT 2011 training data for English, Spanish, Czech

**Seed:** 5000 most frequent words translated with Google Translate

**Test:** 1000 next frequent words translated with Google Translate

→ Removed digits, punctuation and transliterations

<table>
<thead>
<tr>
<th>Languages</th>
<th>top-1</th>
<th>top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>En-Es</td>
<td>33 %</td>
<td>51 %</td>
</tr>
<tr>
<td>Es-En</td>
<td>35 %</td>
<td>50 %</td>
</tr>
<tr>
<td>En-Cz</td>
<td>27 %</td>
<td>47 %</td>
</tr>
<tr>
<td>Cz-En</td>
<td>23 %</td>
<td>42 %</td>
</tr>
<tr>
<td>+ Es-En</td>
<td>53 %</td>
<td>80 %</td>
</tr>
</tbody>
</table>

→ with spanish google news
NER Results

- Use the bilingual word embeddings to initialize the lookup table in the NER classifier

- Ni et al. (2017)
  - Spanish:
    - supervised: 80.6
    - transfer learning: 57.4

- Dutch:
  - supervised: 82.3
  - transfer learning: 60.3

- German:
  - supervised: 71.8
  - transfer learning: 54.4
Summary

- Using neural networks for NER yields good results using (almost) raw representations of words
- Word embeddings can be learned automatically on large amounts of non-annotated data
- Giving pre-trained word embeddings as input to neural networks improve end-to-end task
- Bilingual word embeddings make it possible to transfer knowledge from resource rich languages
Thank you!
References I


References II


