

# Neural Networks for Named Entity Recognition

Viktor Hangya  
(slides originally by Dr. Fabienne Braune)

CIS, LMU Munich

WS 2020-2021

# 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:

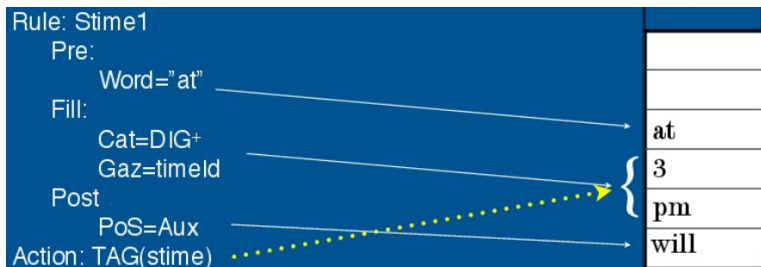
- *Trump attacks BMW and Mercedes*
- *U.N. official Ekeus heads for Baghdad*

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):

Surface	Tag
United	B-ORG
Nations	I-ORG
official	O
Ekeus	B-PER
heads	O
for	O
Baghdad	B-LOC
.	O

Scheme	Begin	Inside	End	Single	Other
IOB	B-X	I-X	E-X	B-X	O
IOE	I-X	I-X	E-X	E-X	O
IOBES	B-X	I-X	E-X	S-X	O

(Collobert 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

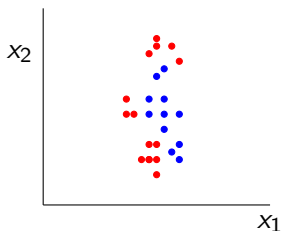


# Classification-based approaches

- 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

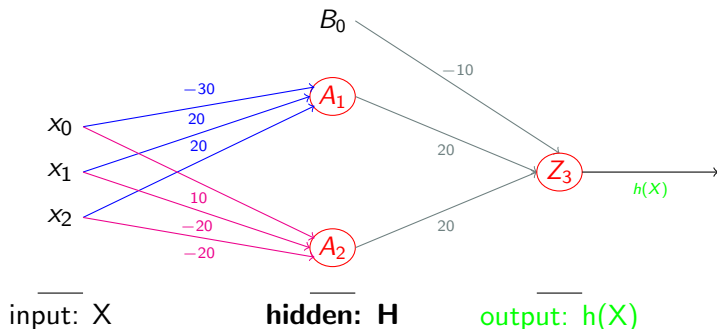


Linear models not suited to learn non-linear decision boundaries.

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  $H$ :

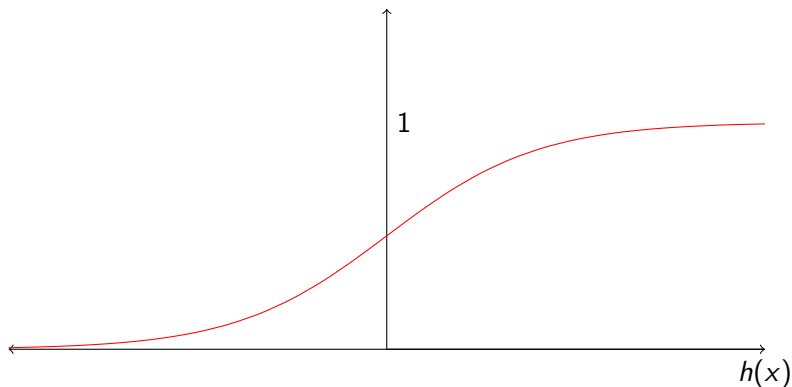
- $A_1 = \sigma(X \cdot \Theta_1)$
- $A_2 = \sigma(X \cdot \Theta_2)$
- $B_0 = 1$  (bias term)

Computation of output unit  $h(X)$ :

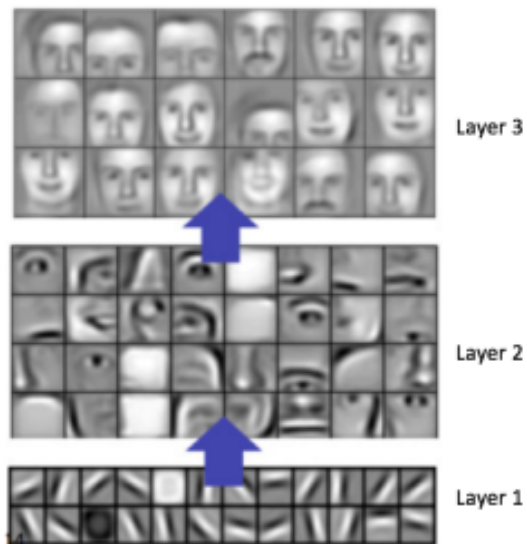
- $h(X) = \sigma(H \cdot \Theta_3)$

# 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*

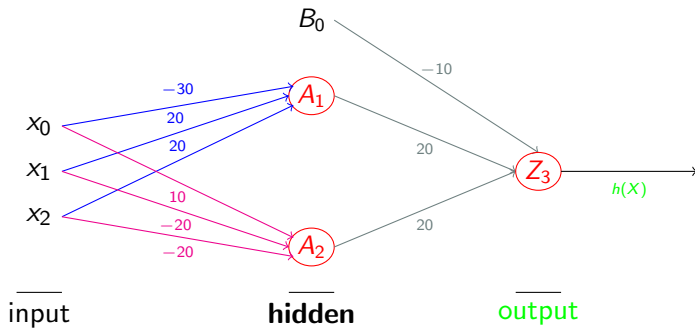
Binray NER task: Is the segment from position 1 to 2 a Named Entity?

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

$$\mathbf{H} = \begin{bmatrix} B_0 = 1 \\ A_1 = \sigma(X \cdot \Theta_1) \\ A_2 = \sigma(X \cdot \Theta_2) \\ \dots \\ A_j = \sigma(X \cdot \Theta_j) \end{bmatrix}$$

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

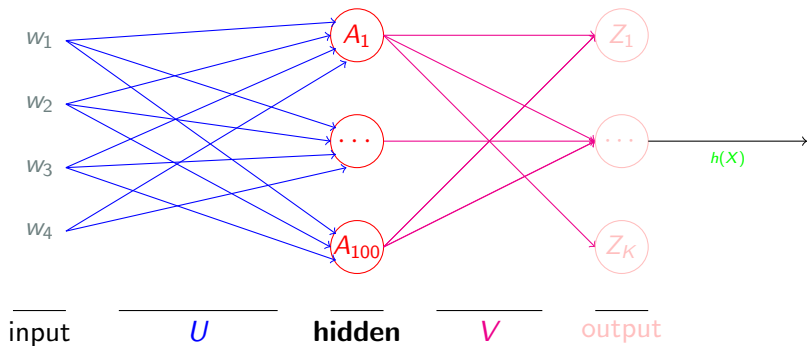
**Training:** Find weight matrices  $U = (\Theta_1, \Theta_2)$  and  $V = \Theta_3$  such that  $h(X)$  is the **correct answer** as many times as possible.

- Given a set  $T$  of training examples  $t_1, \dots, t_n$  with **correct labels**  $\mathbf{y}_i$ , find  $U = (\Theta_1, \Theta_2)$  and  $V = \Theta_3$  such that  $h(X) = \mathbf{y}_i$  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  $c_i \in C = \{c_1, \dots, c_k\}$
- **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



# 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**<sub>-2</sub> **attacks**<sub>-1</sub> **BMW** **and**<sub>1</sub> **Mercedes**<sub>2</sub>

→ each word  $w_i$  is represented as one-hot vector

→  $w_i = [0, 1, 0, 0, \dots, 0]$

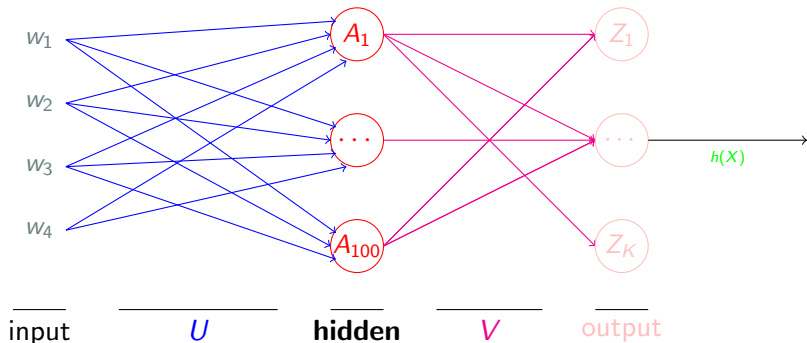
Neural network training:

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**

# 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  $t_1, \dots, t_n$  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:  $h(x^i) = [0.01, 0.1, 0.001, 0.95, \dots, 0.01]$

▶ correct label:  $y^i = [0, 0, 1, 0, \dots, 0]$

$$E = \frac{1}{2} \sum_{j=1}^n (y_j^i - h(x^i)_j)^2 \text{ (mean squared)}$$

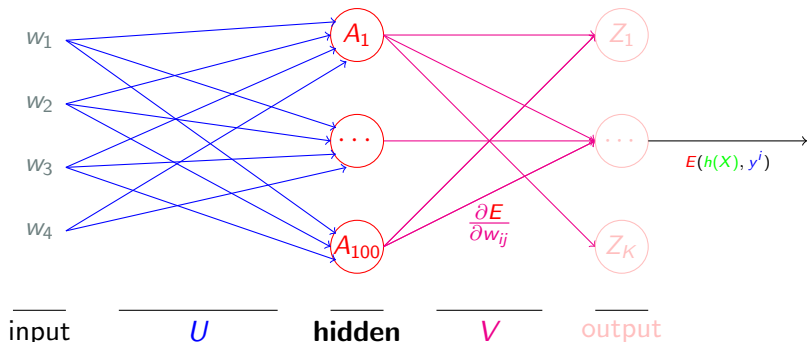
Search **influence of weight on error**:

$$\frac{\partial E}{\partial w_{ij}}$$

$w_{ij}$ : single weight in weight matrix



# Backpropagation



## 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
  - ▶  $w_{BMW} = [1, 0, 0, 0, \dots, 0]$
  - ▶  $w_{Mercedes} = [0, 1, 0, 0, \dots, 0]$
  - ▶  $w_{happiness} = [0, 0, 1, 0, \dots, 0]$
  
- This would be better
  - ▶  $w_{BMW} = [1, 0, 0, 0, \dots, 0]$
  - ▶  $w_{Mercedes} = [1, 0, 0, 0, \dots, 0]$
  - ▶  $w_{happiness} = [0, 0, 1, 0, \dots, 0]$

# 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  $LT_i$

- ▶  $w_{BMW} = [0.5, 0.5, 0.0, 0.0, \dots, 0.0]$

- ▶  $w_{Mercedes} = [0.5, 0.0, 0.5, 0.0, \dots, 0.0]$

## Dot product with (trained) weight vector

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

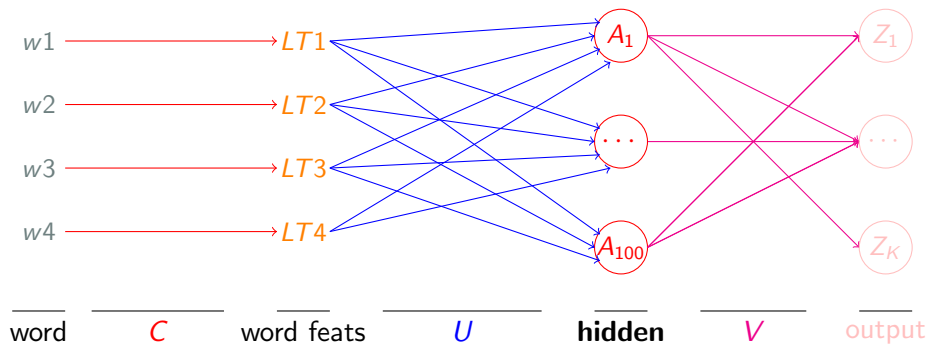
$$w_{table} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 0.02 & 0.1 & 0.05 & 0.03 & 0.01 \\ 0.15 & 0.2 & 0.01 & 0.02 & 0.11 \\ 0.03 & 0.1 & 0.04 & 0.04 & 0.12 \end{bmatrix}$$

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

Words get mapped to lower dimension

→ Hyperparameter to be set

# Feedforward Neural Network with Lookup Table



C is shared!

## Dot product with (initial) weight vector

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

$$w_{table} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \end{bmatrix}$$

$$LT_{table} = w_{table} \cdot C^T = \begin{bmatrix} 0.01 \\ 0.01 \\ 0.01 \end{bmatrix}$$

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  $t_1, \dots, t_n$  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)

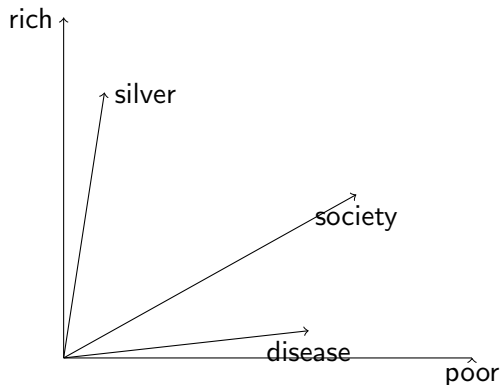
→ **Small** amount of **annotated** data.

- Closest words to **France**
  - ▶ Persuade
  - ▶ Faw
  - ▶ Blackstock
- Closest words to **XBOX**
  - ▶ Decadent
  - ▶ Divo
  - ▶ Versus

# ADDING PRE-TRAINED WORD EMBEDDINGS

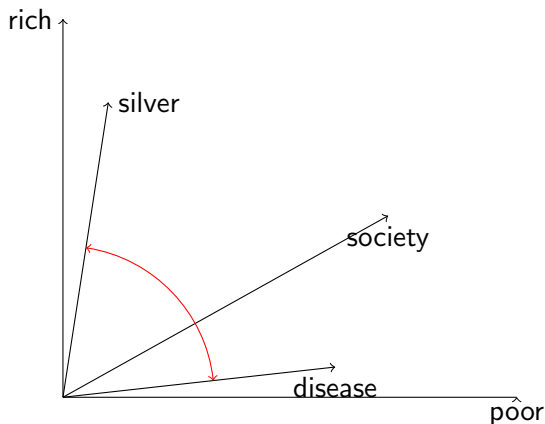
# Word Embeddings

- Representation of words in vector space



# Word Embeddings

- Similar words are close to each other  
→ Similarity is the cosine of the angle between two word vectors



# Learning word embeddings

## Count-based methods:

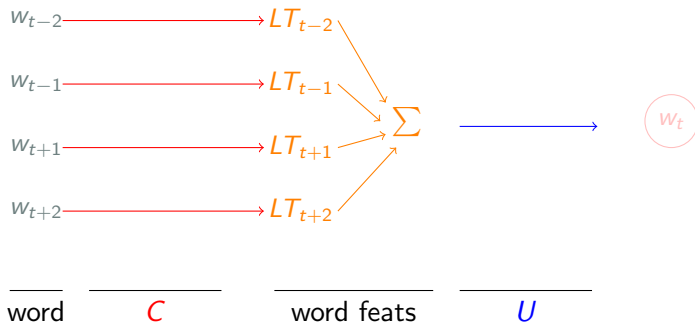
- Compute cooccurrence statistics
- Learn high-dimensional representation
- Map sparse high-dimensional vectors to small dense representation
- Matrix factorization approaches: SVD

## 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)

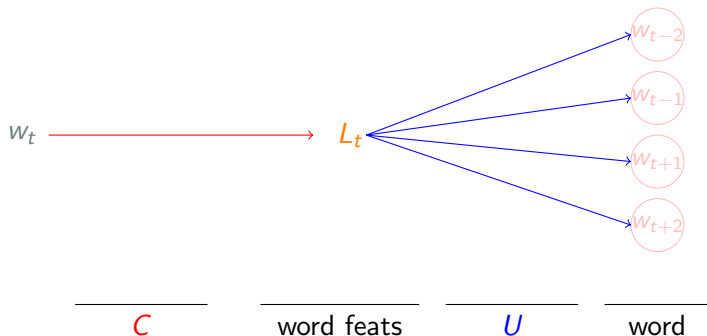
# Learning word embeddings with CBOW

Training example: *Trump attacks BMW and Mercedes*



# Learning word embeddings with skip-gram

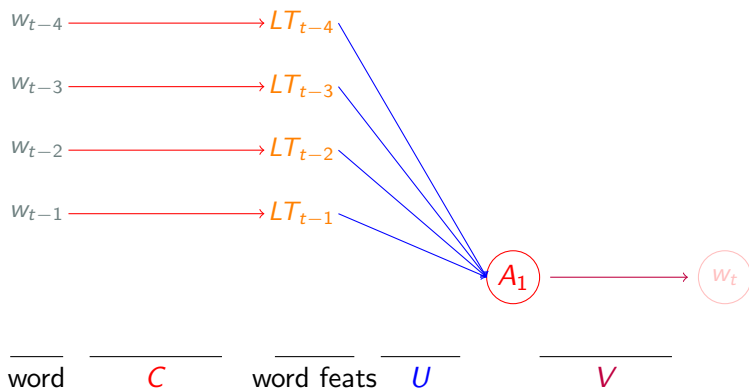
Training example: *Trump attacks BMW and Mercedes*





# Learning word embeddings with Language Modeling

Training example: *Trump attacks BMW and Mercedes*



# 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:
  - ▶  $w_{BMW} = [0.5, 0.5, 0.0, 0.0, \dots, 0.0]$
  - ▶  $w_{Mercedes} = [0.5, 0.0, 0.5, 0.0, \dots, 0.0]$
  - ▶  $w_{happiness} = [0.0, 0.0, 0.0, 1.0, \dots, 0.0]$
- 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 → losing information
    - ★ too large → noise or data scarcity

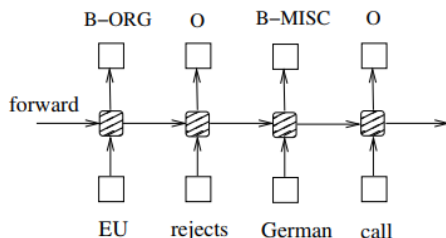
**Let's have a party at JFK**

- ▶ 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**



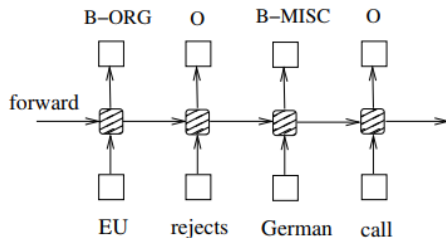
# Recurrent Neural Network (RNN)



(Huang et al., 2015)

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

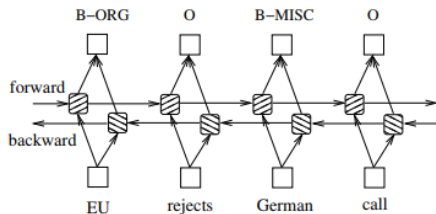
# RNNs for NER



(Huang et al., 2015)

- Input: words
- Lookup layer
  - ▶ learn embeddings from scratch
  - ▶ or used pre-trained embeddings
- Probabilities of each NER tag

# Bidirectional RNNs



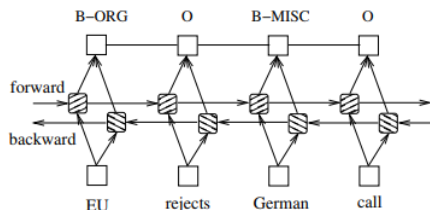
(Huang et al., 2015)

**The European Union 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

$$\blacktriangleright o_t = g(\vec{h}_t | \overleftarrow{h}_t; \theta_2)$$

# Conditional Random Fields (CRF)



(Huang et al., 2015)

- 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  $i$

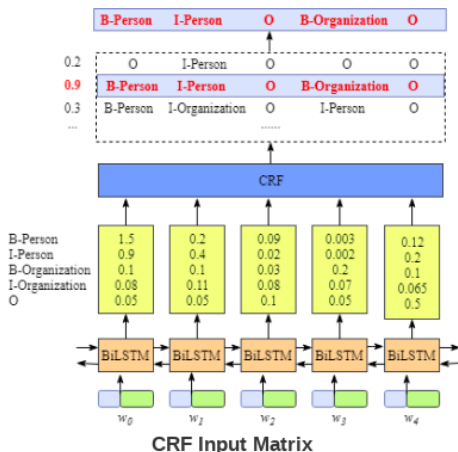
# CRF transition matrix

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

**CRF State Transition Matrix**

(Image taken from <https://eli5.readthedocs.io> sklearn tutorial)

# RNN + CRF for NER



(Image taken from <https://createmomo.github.io/>)

- 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



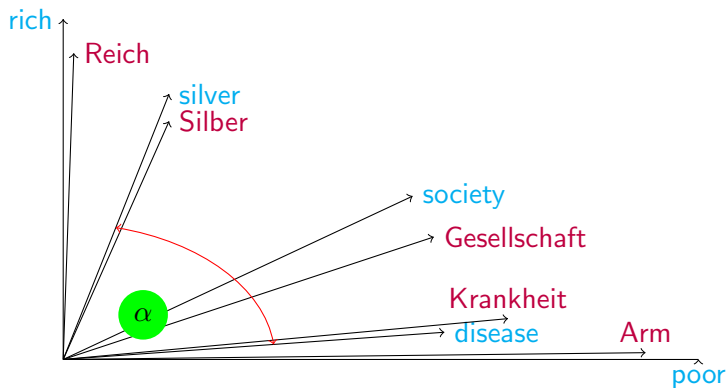
# Bilingual transfer learning

- 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
  - ▶ Training data is needed for the translation system
- Target language words are out-of-vocabulary words for a system trained on the source language
  - ▶ similarity of source and target words → 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

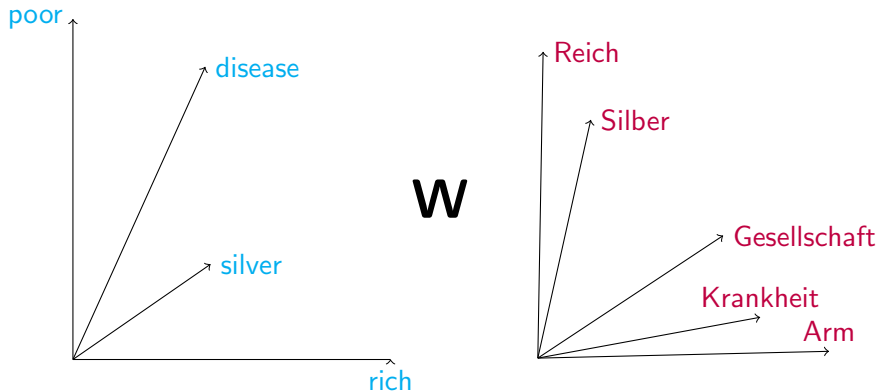


# Learning Bilingual Word Embeddings

- Learn bilingual embeddings from parallel sentences  
Hermann and Blunsom (2014), Gouws et al. (2015), Gouws and Søgaard (2015), Duong et al. (2016)  
Need for parallel sentences
- Learn bilingual embeddings from aligned documents  
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

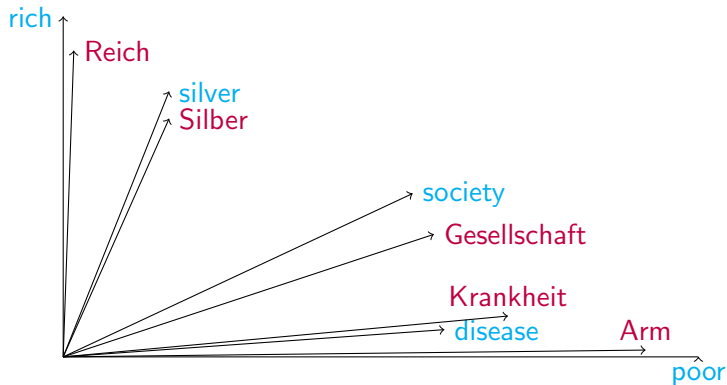
# Post-hoc mapping with seed lexicon

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



# Post-hoc mapping with seed lexicon

- Project source words into target space



# 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 **W** using a seed lexicon
  - ▶ Need a list of **5000 English words and their translation**

# Learning $W$ with Regression



(Conneau et al., 2017)

## Regression (Mikolov et al. (2013))

$$\mathbf{W}^* = \arg \min_{\mathbf{W}} \sum_i^n \| \mathbf{x}_i \mathbf{W} - \mathbf{y}_i \|^2$$

$\mathbf{x}_i$  : **embedding** of i-th **source** (English) word in the seed lexicon.

$\mathbf{y}_i$  : **embedding** of i-th **target** (German) word in the seed lexicon.

# Learning $W$ with Ridge Regression

Regression (Mikolov et al. (2013))

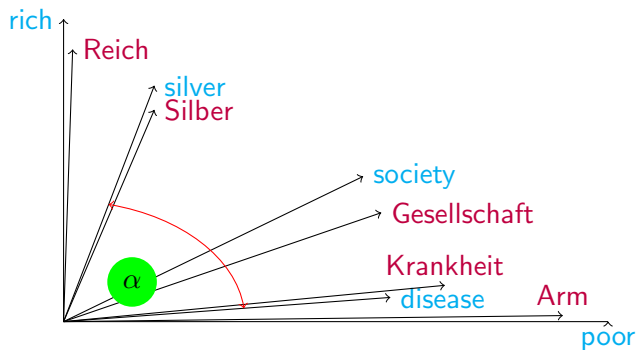
$$\mathbf{W}^* = \arg \min_{\mathbf{W}} \sum_i^n \| \mathbf{x}_i \mathbf{W} - \mathbf{y}_i \|^2$$

- Predict projection  $\mathbf{y}^*$  by computing  $\mathbf{x}_i \mathbf{W}$
- Compute **squared error** between  $\mathbf{y}^*$  and  $\mathbf{y}_i$ 
  - ▶ Correct translation  $t_i$  given in seed lexicon
  - ▶ Vector representation  $\mathbf{y}_i$  is given by embedding of  $t_i$
- Find  $\mathbf{W}$  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

Languages	top-1	top-5
En-Es	33 %	51 %
Es-En	35 %	50 %
En-Cz	27 %	47 %
Cz-En	23 %	42 %
+ Es-En	53 %	80 %

→ 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:
  - ▶ Spanish training: 80.6
  - ▶ English training: 57.4
- Dutch:
  - ▶ Dutch training: 82.3
  - ▶ English training: 60.3
- German:
  - ▶ German training: 71.8
  - ▶ English training: 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

- Ando, R. K. and Zhang, T. (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*.
- Conneau, A., Lample, G., Ranzato, M., Denoyer, L., and Jégou, H. (2017). Word translation without parallel data. *arXiv preprint arXiv:1710.04087*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Duong, L., Kanayama, H., Ma, T., Bird, S., and Cohn, T. (2016). Learning crosslingual word embeddings without bilingual corpora. In *Proc. EMNLP*.

## References II

- Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proc. EACL*.
- Florian, R., Ittycheriah, A., Jing, H., and Zhang, T. (2003). Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*.
- Gouws, S., Bengio, Y., and Corrado, G. (2015). Bilbowa: Fast bilingual distributed representations without word alignments. In *Proc. ICML*.
- Gouws, S. and Søgaard, A. (2015). Simple task-specific bilingual word embeddings. In *Proc. NAACL*.
- Hermann, K. M. and Blunsom, P. (2014). Multilingual models for compositional distributed semantics. In *Proc. ACL*, pages 58–68, Baltimore, Maryland. Association for Computational Linguistics.
- Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

## References III

- Lazaridou, A., Dinu, G., and Baroni, M. (2015). Hubness and pollution: Delving into cross-space mapping for zero-shot learning. In *Proc. ACL*.
- Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*.
- Mikolov, T., Le, Q. V., and Sutskever, I. (2013). Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168.
- Ni, J., Dinu, G., and Florian, R. (2017). Weakly supervised cross-lingual named entity recognition via effective annotation and representation projection. *arXiv preprint arXiv:1707.02483*.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of NAACL*.



## References IV

- Vulic, I. and Korhonen, A. (2016). On the Role of Seed Lexicons in Learning Bilingual Word Embeddings. In *Proc. ACL*, pages 247–257.
- Vulic, I. and Moens, M. (2015). Bilingual word embeddings from non-parallel document-aligned data applied to bilingual lexicon induction. In *Proc. ACL*.