Word Embeddings for Named Entity Recognition

Fabienne Braune[1]

[1]LMU Munich

January 25th, 2016

Outline

1. Named Entity Recognition
2. Feedforward Neural Networks: recap
3. Neural Networks for Named Entity Recognition
4. Example
5. Adding Pre-trained Word Embeddings
6. Word2Vec

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

Labeled data

Desired outputs:

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

Example annotations (CoNLL-2003):

Surface	POS	Sh-synt	Tag
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

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)

Classification-based approaches

- Classifier combination with engineered features (Florian et al. 2003)
  - Manually engineer features
  - Use large (external) gazetteer
  - Combine classifiers (ME, MRR, HMM) trained on annotated data
  - 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

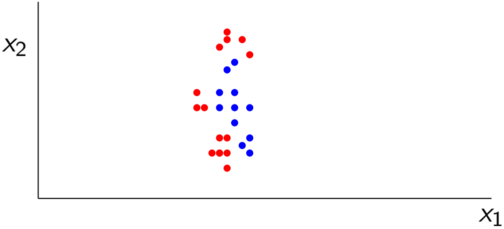
Classification-based approaches

- Use feedforward neural networks (Collobert et al. 2011):
  - With raw words 81.74
  - With pre-trained word embeddings 88.67
  - Using a gazetteer 89.59
- Use sequential models:
  - Linear Chain CRF (linear)
  - LSTM networks (deep)

    $\rightarrow$ Achieve best performance but not covered here

Feedforward Neural Networks: Recap

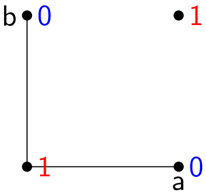
Motivation



Cannot be solved using a linear model

Motivation

a	b	a *XNOR* b
0	0	1
0	1	0
1	0	0
1	1	1



Features : a, b

Feature values : binary

Cannot be solved using a linear model

Motivation

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

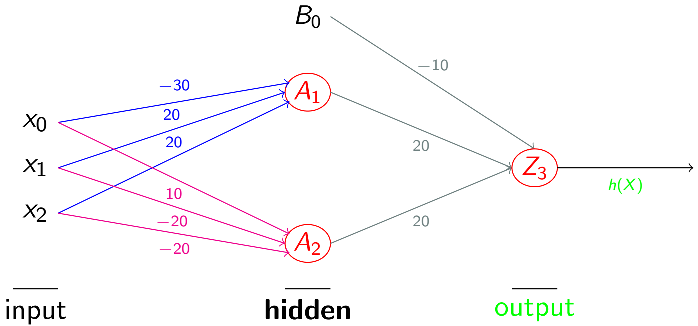
Neural networks can do that

$\rightarrow$ Through composition of non-linear functions

$\rightarrow$ Learn relevant features from (almost) raw text

- $\rightarrow$ No need for manual feature engineering
- $\rightarrow$ 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):

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

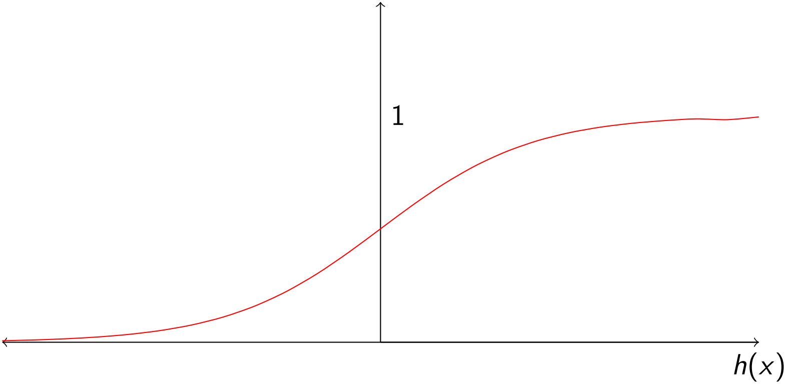
Feedforward Neural Network

Feedforward neural network with:

- 1 input layer X (feature vector)
- 2 weight matrices $U = (\Theta_1, \Theta_2)$ and $V = \Theta_3$
- 1 hidden layer **H** composed of:
  - 2 activations $A_1 = \sigma(Z_1)$ and $A_2 = \sigma(Z_2)$ where:
    - $Z_1 = X \cdot \Theta_1$
    - $Z_2 = X \cdot \Theta_2$
- 1 output unit $h(X) = \sigma(Z_3)$ where:
  - $Z_3 = \mathbf{H} \cdot \Theta_3$

Non-linear activation function

The **sigmoid function** $\sigma(Z)$ is often used



Feedforward neural network

Trump attacks BMW and Mercedes

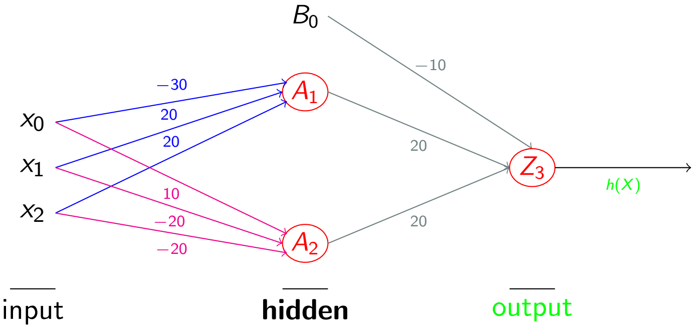
Binray NER task: Is the segment from position 1 to 2 a Named Entity?

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

$\rightarrow$ Predictions will be bad

$\rightarrow$ 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.

$\rightarrow$ Given a set T of training examples $t_1, \cdots t_n$ 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.

$\rightarrow$ Computation of $h(X)$ called forward propagation

$\rightarrow$ $U = (\Theta_1, \Theta_2)$ 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

Neural Networks for NER

Classification-based NER

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)

Labeled data

Desired outputs:

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

Annotation:

Surface	POS	Sh-synt	Tag
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

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**$_{2}$

→ Lookup feature representation (LT_i) for window

Give LT_i as input to Feedforward Neural Network

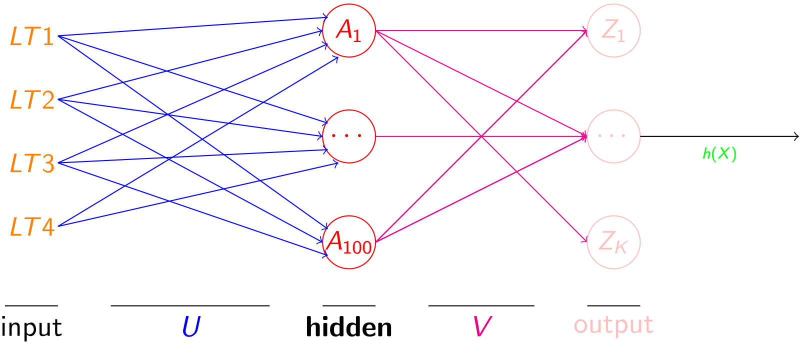
Neural network training:

Predict corresponding label (forward propagation)

→ should be organization (ORG)

Train weights by backpropagating error

Feedforward Neural Network for NER



Input: word features LT_i
Output: predicted label

Note: Bias terms omitted for simplicity

Feedforward Neural Network

Input layer (X): Word features LT1, LT2, LT3, LT4

Weight matrices U, V

Hidden layer (H): $\sigma(X \cdot U + d)$

Output layer (0): $H \cdot V + b$

Prediction: $h(X) = softmax(0)$

- Predicted class is the one with highest probability (given by softmax)

Weight training

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

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

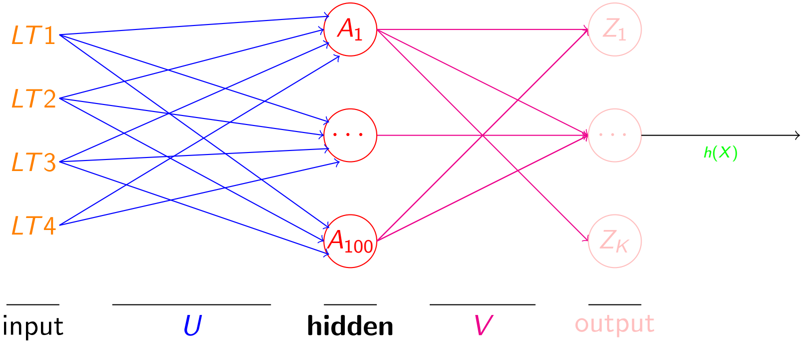
$\rightarrow$ Computation of $h(X)$ with forward propagation

$\rightarrow$ C, U and V with error back propagation

Training data

Training example	POS	Sh-synt	Tag
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

Forward Propagation



Forward propagation:

$\rightarrow$ Perform all operations to get $h(X)$ from input LT.

Backpropagation

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

$\rightarrow$ Error between correct label and prediction is minimal

Compute error at output:

Compare output unit with y^i

- y^i vector with 1 in correct class, 0 otherwise

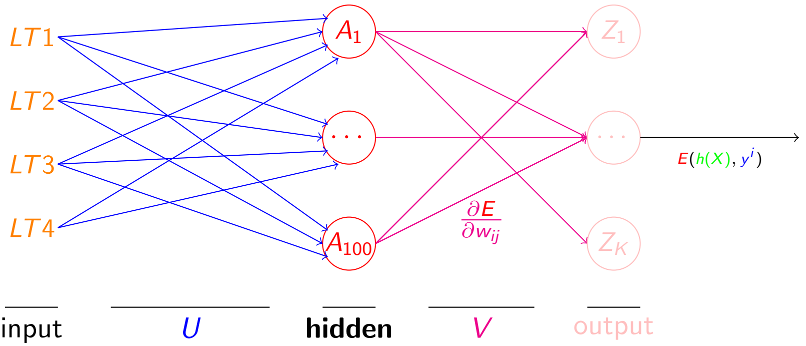
$E = \frac{1}{2}\sum_{i=1}^{n}(y_i - o_i)^2$ (mean squared)

Search influence of weight on error:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial Z_j}\frac{\partial Z_j}{\partial w_{ij}}$$

w_{ij}: single weight in weight matrix

Backpropagation



Backpropagation:

→ E needs to go through output neuron.

→ Chain rule: $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial Z_j} \frac{\partial Z_j}{\partial w_{ij}}$

Backpropagation

Search influence of weight on error:

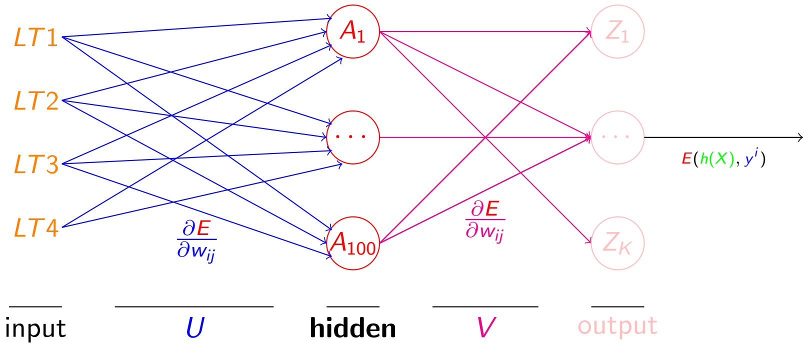
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial Z_j}\frac{\partial Z_j}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial Z_j}x_i, \text{ where } x_i \text{ is input to neuron}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_j}\sigma'(Z_i)x_i \text{ if output neuron}$$

- $\frac{\partial E}{\partial w_{ij}} = (O_j - y_j^i)\sigma'(Z_j)x_i$ if output neuron
- $\delta_j x_i$ if output neuron

Backpropagation



Backpropagation:

Compute error for weights leading to output unit

Compute all other weights

Backpropagation

Search influence of weight on error:

Compute error for weights leading to output unit

Compute all other weights

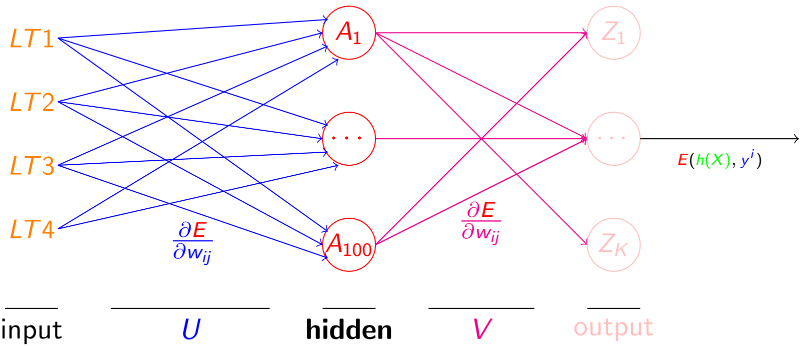
$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial H_j} \frac{\partial H_j}{\partial Z_j} \frac{\partial Z_j}{\partial w_{ij}}$

$\rightarrow$ Use **recursion**:

$\frac{\partial E}{\partial w_{ij}} = \sum_k \delta_k w_{jk} \sigma'(Z_i) x_i$

δ_k is error of preceding unit.

Backpropagation



$\frac{\partial E}{\partial w_{ij}} = \sum_k \delta_k w_{jk} \sigma'(Z_i) x_i$

δ_k is error of preceding unit.

Weight training

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

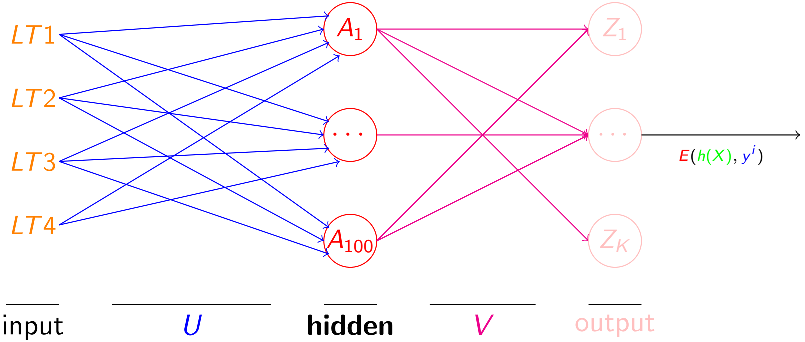
$\rightarrow$ Computation of $h(X)$ with forward propagation

$\rightarrow$ U and V with error back propagation

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 (gradient descent)
4. Goto 1 until convergence

Lookup Layer



Lookup Layer

- Each word encoded into index vector $w_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

- LT_i is dot product of weight matrix C with index of w_i
  $\rightarrow$ C is **shared** among all words

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
$\rightarrow$ Hyperparameter to be set

Dot product with (initial) weight vector

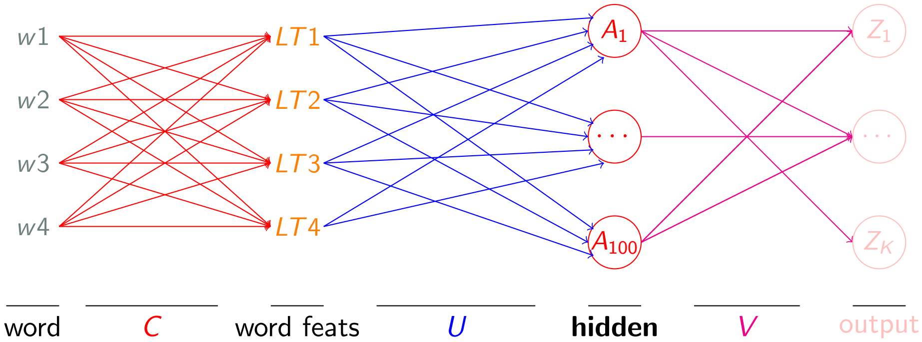
$W = \{$the,cat,on,table,chair$\}$

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

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

Feature vectors same for all words.

Feedforward Neural Network with Lookup Table



Note: Bias terms omitted for simplicity

Weight training

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

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

$\rightarrow$ Computation of $h(X)$ with forward propagation

$\rightarrow$ C, U and V with error back propagation

Dot product with (trained) weight vector

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

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

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

Each word gets a specific feature vector

Training data

Training example	POS	Sh-synt	Tag
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

- Lookup vector C trained with NER training data
- Word feature vectors are trained towards NER

Example

Example

Trump PER attacks BMW ORG and Mercedes ORG

$W = \{\text{Trump,BMW,Mercedes,attacks,and}\}$

$$w_{Trump} = \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} \quad w_{attacks} = \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \quad w_{BMW} = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}$$

$$w_{and} = \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix} \quad w_{Mercedes} = \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}$$

Example

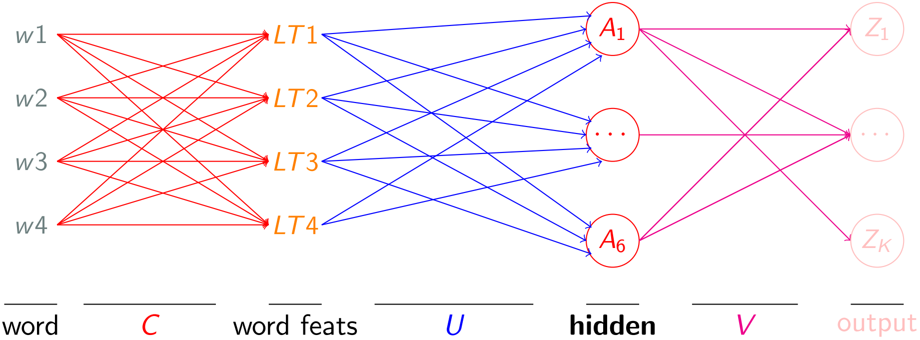
Window: Trump attacks BMW and Mercedes

$$w_{window} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0.01 & 0.8 & 0.05 & 0.02 & 0.01 \\ 0.03 & 0.2 & 0.08 & 0.01 & 0.02 \\ 0.04 & 0.1 & 0.04 & 0.02 & 0.04 \end{bmatrix}$$

C is **randomly initialized**

$LT = w_{window} \cdot C^T$

Example



Output of lookup table given as input
Note: Bias terms omitted for simplicity

Example

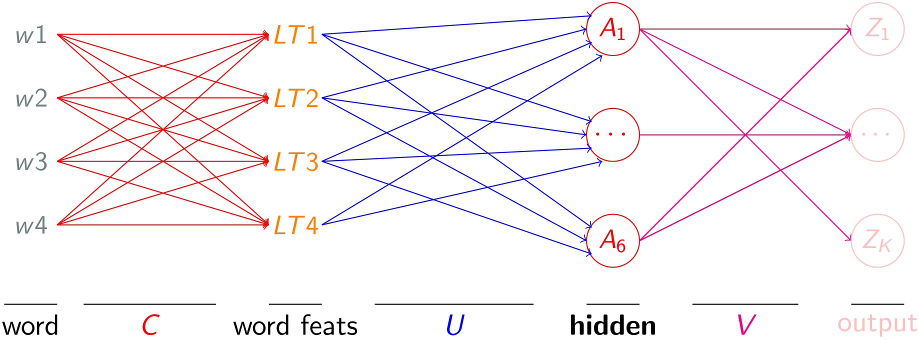
$$LT = \begin{bmatrix} 0.01 & 0.03 & 0.04 \\ 0.05 & 0.08 & 0.04 \\ 0.01 & 0.02 & 0.04 \\ 0.8 & 0.2 & 0.1 \\ 0.02 & 0.01 & 0.4 \end{bmatrix} \quad U = \begin{bmatrix} 0.04 & 0.6 & 0.01 & 0.02 & 0.06 & 0.03 \\ 0.01 & 0.9 & 0.02 & 0.05 & 0.03 & 0.05 \\ 0.02 & 0.3 & 0.05 & 0.07 & 0.09 & 0.01 \\ 0.02 & 0.4 & 0.02 & 0.03 & 0.04 & 0.02 \\ 0.01 & 0.8 & 0.01 & 0.01 & 0.03 & 0.07 \end{bmatrix}$$

U is **randomly initialized**

$\mathbf{Z} = LT^T \cdot U^T$

$A = \sigma(\mathbf{Z})$

Example



Output of lookup table given as input
Note: Bias terms omitted for simplicity

Example

- Repeat same procedure for each hidden layer
- Apply softmax on output (last) layer
- Predict label

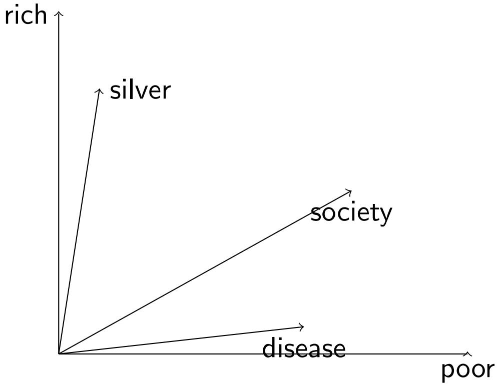
Example

- Compute error between prediction (e.g. LOCATION) and true label
  → Given in training data (BMW is ORG)
- Backpropagate error through network and adjust weights
- Redo same procedure with adjusted weights
- Stop at convergence
  → Or early stopping on held-out dataset

Adding Pre-trained Word Embeddings

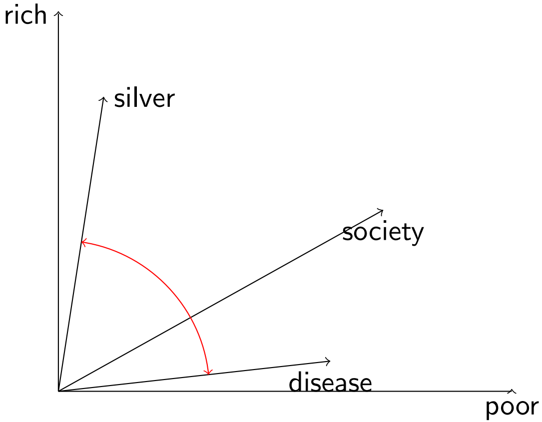
Word Embeddings

- Representation of words in vector space



Word Embeddings

- Similar words are close to each other
  $\rightarrow$ 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

Neural networks:

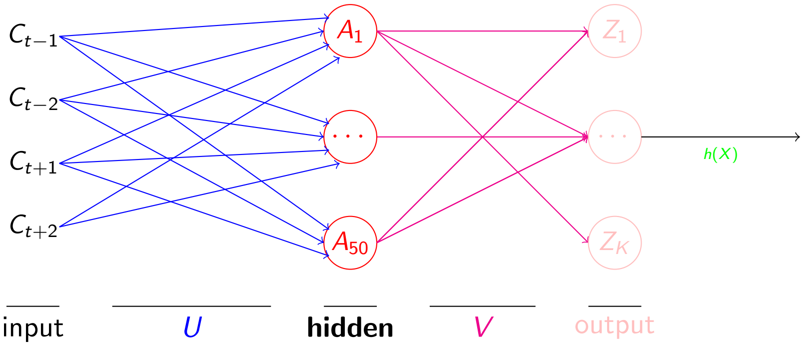
- Predict a word from its neighbors
- Learn (small) embedding vectors

Word vectors with Neural Networks

- LM Task: Given k previous words, predict the current word
  $\rightarrow$ For each word w in V, model $P(w_t | w_{t-1}, w_{t-2}, ..., w_{t-n})$
  $\rightarrow$ **Learn embeddings C of words**
  $\rightarrow$ Input for task

- Task: Given k context words, predict the current word
  $\rightarrow$ **Learn embeddings C of words**

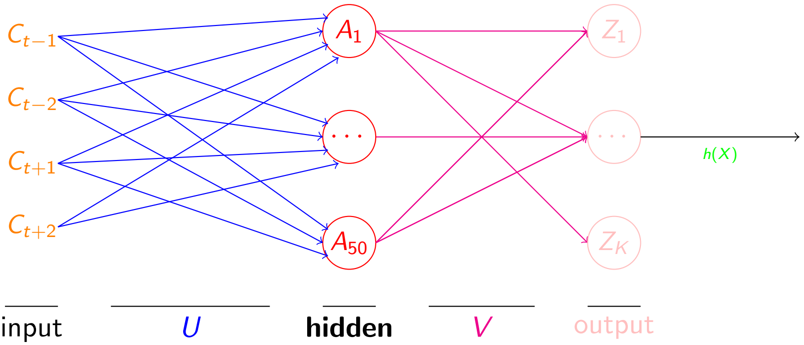
Network architecture



Given words w_{t-2}, w_{t-1}, w_{t+1} and w_{t+2}, predict w_t

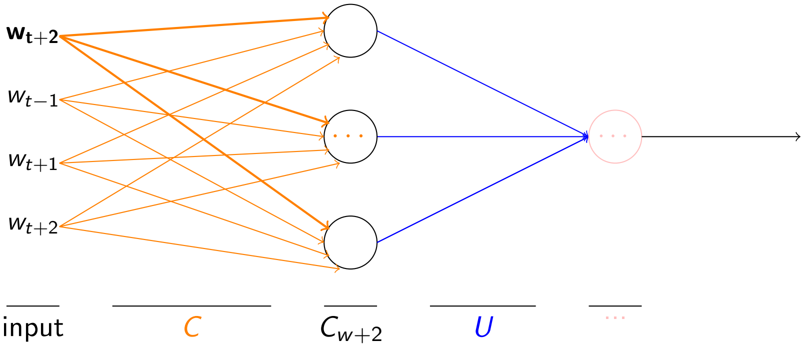
Note: Bias terms omitted for simplicity

Network architecture



We want the context vectors $\rightarrow$ embed words in shared space
Note: Bias terms omitted for simplicity

Getting the Word Embeddings



Same as lookup table but trained on a language model task (predict w_t)

NER lookup table was trained on NER task (predict NE label)

Word Embeddings for NER

- Train word embeddings using language model task:
  - $\rightarrow$ Labels are words w_t
  - $\rightarrow$ No need for NER training data
  - $\rightarrow$ Use large amounts of non-annotated data

- Replace lookup table C (randomly initialized) with C (pre-trained)

Example

Window: Trump attacks BMW and Mercedes

$$w_{window} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0.01 & 0.8 & 0.05 & 0.02 & 0.01 \\ 0.03 & 0.2 & 0.08 & 0.01 & 0.02 \\ 0.04 & 0.1 & 0.04 & 0.02 & 0.04 \end{bmatrix}$$

C is **randomly initialized**

Before NER training, word embeddings are very bad.
After NER training, word embeddings are good for NER.

Example

Window: Trump attacks BMW and Mercedes

$$w_{window} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0.01 & 0.8 & 0.05 & 0.02 & 0.01 \\ 0.03 & 0.2 & 0.08 & 0.01 & 0.02 \\ 0.04 & 0.1 & 0.04 & 0.02 & 0.04 \end{bmatrix}$$

C is **pre-trained** on LM task

Before NER training, word embeddings are good word embeddings.

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

Word embeddings trained on LM task
→ Large amount of **non-annotated** data.

- Closest words to France
  - Austria
  - Belgium
  - Germany
- Closest words to XBOX
  - Amiga
  - Playstation
  - MSX

Results

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

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

Results

- Pre-trained word embeddings yield comparable results to state of the art NER systems

- To beat the best system, additional features are needed
  - Indicate if word is in Gazetteer or not

Word2Vec

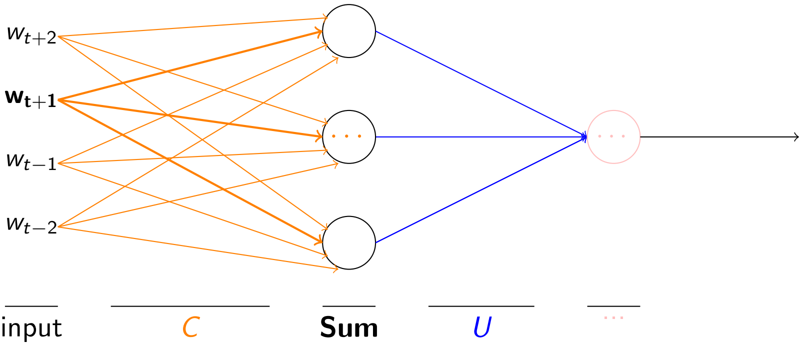
Word2Vec

- Software train word embeddings (Mikolov. 2013)
  $\rightarrow$ very fast
- Two models:
  - Skip-gram model:
    - Input is w_t
    - Prediction is w_{t+2}, w_{t+1}, w_{t-1} and w_{t-2}
  - BOW model:
    - Input is is w_{t+2}, w_{t+1}, w_{t-1} and w_{t-2}
    - Prediction is w_t

Fast computation of Word Embeddings

- Inner workings of BOW same as language model architecture
- Some components are changed to speed up computation
  - Remove hidden layer
  - Sum over all projections
  - Replace softmax by logistic unit with negative sampling

Simplifications

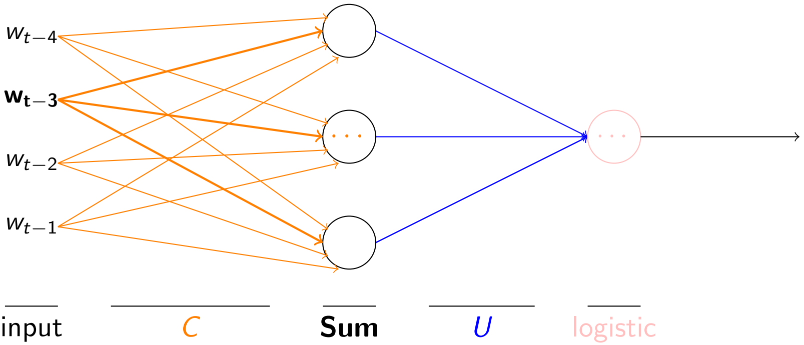


Remove hidden layer and sum over context
Note: Bias terms omitted for simplicity

Simplifications

- Single logistic unit instead of output layer
  $\rightarrow$ No need for distribution over words (only vector representation)
  $\rightarrow$ Task as binary classification problem:
  - Given input and weight matrix say if w_t is current word
  - We know the correct w_t, how do we get the wrong ones?
    $\rightarrow$ negative sampling

Simplifications



Remove hidden layer and sum over context
Note: Bias terms omitted for simplicity

Word2Vec for NER

- Quickly train word embeddings on very large amounts of non-annotated data
- Give pre-trained word embeddings as input to NER network

Recap

- Using neural networks for NER yields good results using (almost) raw representations of words
- Example feedforward neural network for NER
- 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

Thank you !