An introduction to Neural Networks

Fabienne Braune[1]

[1]LMU Munich

December 6th, 2016

Outline

1. Linear models
2. Limitations of linear models
3. Neural networks
4. Word embeddings
5. A neural language model
6. Training word embeddings

LINEAR MODELS

Binary Classification with Linear Models

Example: the seminar at < time > 4 pm will

Classification task: Do we have an < time > tag in the current position?

Word	Lemma	LexCat	Case	SemCat	Tag
the	the	Art	low		
seminar	seminar	Noun	low		
at	at	Prep	low		stime
4	4	Digit	low		
pm	pm	Other	low	timeid	
will	will	Verb	low		

Feature Vector

Encode context into feature vector:

1	bias term		**1**
2	-3_lemma_the		**1**
3	-3_lemma_giraffe		**0**
...	...	...	
102	-2_lemma_seminar		**1**
103	-2_lemma_giraffe		**0**
...	...	...	
202	-1_lemma_at		**1**
203	-1_lemma_giraffe		**0**
...	...	...	
302	+1_lemma_4		**1**
303	+1_lemma_giraffe		**0**
...	...	...	

Dot product with (initial) weight vector

$$h(X) = X \cdot \Theta^T \qquad X = \begin{bmatrix} x_0 = 1 \\ x_1 = 1 \\ x_2 = 0 \\ \dots \\ x_{101} = 1 \\ x_{102} = 0 \\ \dots \\ x_{201} = 1 \\ x_{202} = 0 \\ \dots \\ x_{301} = 1 \\ x_{302} = 0 \\ \dots \end{bmatrix} \qquad \Theta = \begin{bmatrix} w_0 = 1.00 \\ w_1 = 0.01 \\ w_2 = 0.01 \\ \dots \\ x_{101} = 0.01 \\ x_{102} = 0.01 \\ \dots \\ x_{201} = 0.01 \\ x_{202} = 0.01 \\ \dots \\ x_{301} = 0.01 \\ x_{302} = 0.01 \\ \dots \end{bmatrix}$$

Prediction with dot product

$$
\begin{aligned}
h(X) &= X \cdot \Theta^T \\
&= x_0 w_0 + x_1 w_1 + \cdots + x_n w_n \\
&= 1 * 1 + 1 * 0.01 + 0 * 0.01 + ... + 0 * 0.01 + 1 * 0.01
\end{aligned}
$$

Predictions with linear models

Example: the seminar at $<$ time $>$ 4 pm will

Classification task: Do we have an $<$ time $>$ tag in the current position?

Linear Model: $h(X) = X \cdot \Theta^T$

Prediction: If $h(X) > 0.5$, yes. Otherwise, no.

Getting the right weights

Training: Find weight vector Θ 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 y_i, find Θ such that $h(X(t_i)) = y_i$ for as many t_i as possible.

$\rightarrow$ $X(t_i)$ is the feature vector for the i-th training example t_i

Dot product with trained weight vector

$$h(X) = X \cdot \Theta^T \qquad X = \begin{bmatrix} x_0 = 1 \\ x_1 = 1 \\ x_2 = 0 \\ \dots \\ x_{101} = 1 \\ x_{102} = 0 \\ \dots \\ x_{201} = 1 \\ x_{202} = 0 \\ \dots \\ x_{301} = 1 \\ x_{302} = 0 \\ \dots \end{bmatrix} \qquad \Theta = \begin{bmatrix} w_0 = 1.00 \\ w_1 = 0.001 \\ w_2 = 0.02 \\ \dots \\ x_{101} = 0.012 \\ x_{102} = 0.0015 \\ \dots \\ x_{201} = 0.4 \\ x_{202} = 0.005 \\ \dots \\ x_{301} = 0.1 \\ x_{302} = 0.04 \\ \dots \end{bmatrix}$$

Working with real-valued features

E.g. measure semantic similarity:

Word	sim(time)
the	0.0014
seminar	0.0014
at	0.1
4	2.01
pm	3.02
will	0.5

Working with real-valued features

$$h(X) = X \cdot \Theta^T \qquad X = \begin{bmatrix} x_0 = 1.0 \\ x_1 = 50.5 \\ x_2 = 52.2 \\ \dots \\ x_{101} = 45.6 \\ x_{102} = 60.9 \\ \dots \\ x_{201} = 40.4 \\ x_{202} = 51.9 \\ \dots \\ x_{301} = 40.5 \\ x_{302} = 35.8 \\ \dots \end{bmatrix} \qquad \Theta = \begin{bmatrix} w_0 = 1.00 \\ w_1 = 0.001 \\ w_2 = 0.02 \\ \dots \\ x_{101} = 0.012 \\ x_{102} = 0.0015 \\ \dots \\ x_{201} = 0.4 \\ x_{202} = 0.005 \\ \dots \\ x_{301} = 0.1 \\ x_{302} = 0.04 \\ \dots \end{bmatrix}$$

Working with real-valued features

$$
\begin{aligned}
h(X) &= X \cdot \Theta^T \\
&= x_0 w_0 + x_1 w_1 + \cdots + x_n w_n \\
&= 1.0 * 1 + 50.5 * 0.001 + ... + 40.5 * 0.1 + 35.8 * 0.04 \\
&= 540.5
\end{aligned}
$$

Working with real-valued features

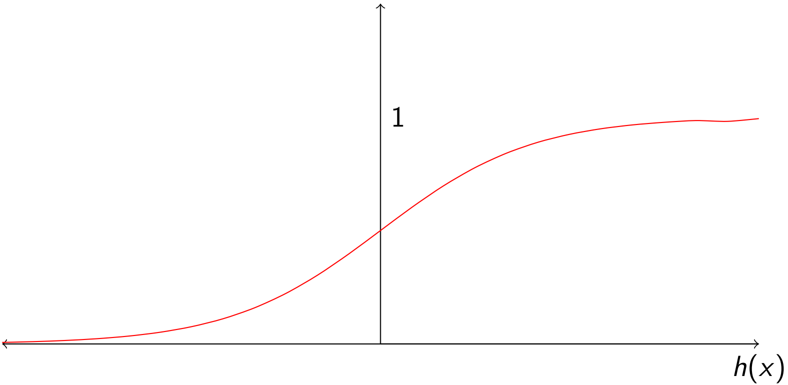
Classification task: Do we have an $<$ time $>$ tag in the current position?

Prediction: $h(X) = 540.5$

- What does 540.5 mean?

Sigmoid function

We can push $h(X)$ between 0 and 1 using a **non-linear** activation function
The **sigmoid function** $\sigma(Z)$ is often used



Logistic Regression

Classification task: Do we have an < time > tag in the current position?

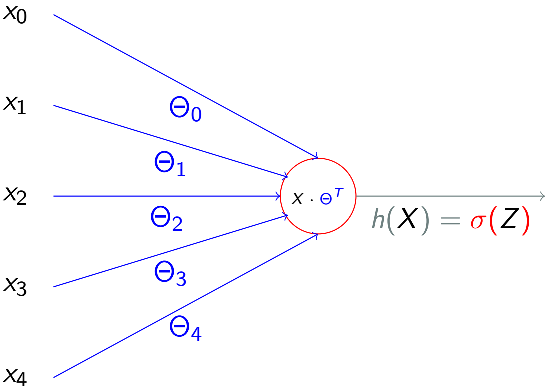
Linear Model: $Z = X \cdot \Theta^T$

Prediction: If $\sigma(Z) > 0.5$, yes. Otherwise, no.

Logistic regression:

- Use a **linear model** and squash values between 0 and 1.
  - Convert real values to probabilities
- Put threshold to 0.5.
- Positive class above threshold, negative class below.

Logistic Regression



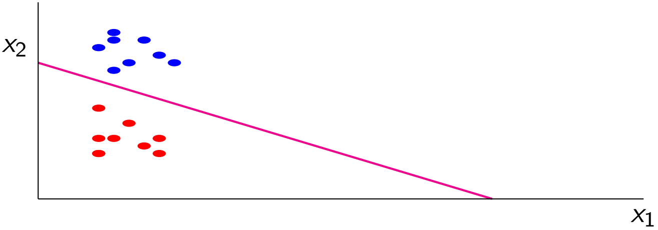
Linear Models: Limitations

Decision Boundary

What do **linear** models do?

- $\sigma(Z) > 0.5$ when $Z(= X \cdot \Theta^T) \geq 0$
- Model defines a decision boundary given by $X \cdot \Theta^T = 0$

  positive examples (have time tag)
  negative examples (no time tag)

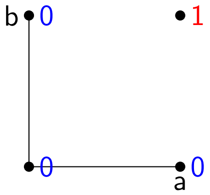


Exercise

When we model a task with linear models, what assumption do we make about positive/negative examples?

Modeling 1: Learning a predictor for $\wedge$

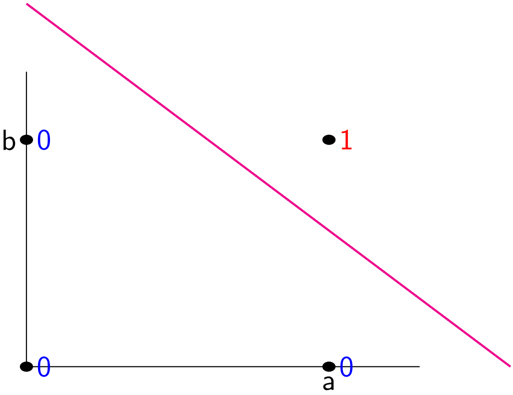
a	b	a $\wedge$ b
0	0	0
0	1	0
1	0	0
1	1	1



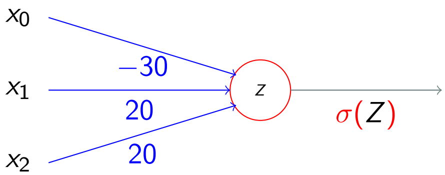
Features : a, b   Feature values : binary

Can we learn a linear model to solve this problem?

Modeling 1: Learning a predictor for $\wedge$



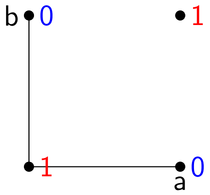
Modeling 1: Logistic Regression



x_0	x_1	x_2	$x_1 \wedge x_2$
1	0	0	$\sigma(1 * -30 + 0 * 20 + 0 * 20) = \sigma(-30) \approx 0$
1	0	1	$\sigma(1 * -30 + 0 * 20 + 1 * 20) = \sigma(-10) \approx 0$
1	1	0	$\sigma(1 * -30 + 1 * 20 + 1 * 20) = \sigma(-10) \approx 0$
1	1	1	$\sigma(1 * -30 + 1 * 20 + 1 * 20) = \sigma(10) \approx 1$

Modeling 2: Learning a predictor for *XNOR*

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

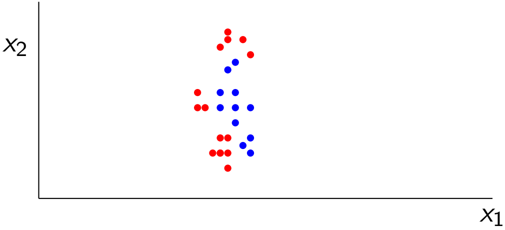


Features : a, b

Feature values : binary

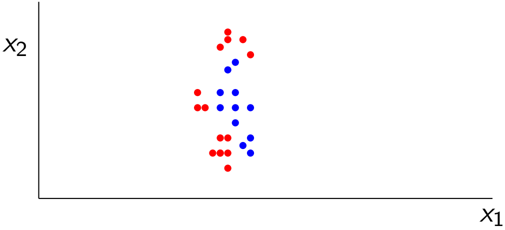
Can we learn a linear model to solve this problem?

Non-linear decision boundaries



Can we learn a linear model to solve this problem?

Non-linear decision boundaries



Can we learn a linear model to solve this problem?

No! Decision boundary is **non-linear**.

Learning a predictor for *XNOR*

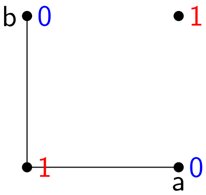
Linear models not suited to learn non-linear decision boundaries.

Neural networks can do that.

Neural Networks

Learning a predictor for *XNOR*

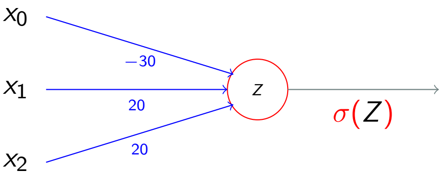
a	b	a *XNOR* b
0	0	1
0	1	0
1	0	0
1	1	1



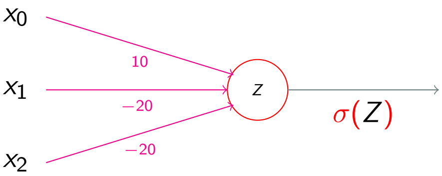
Features : a, b   Feature values : binary

Can we learn a **non-linear model** to solve this problem?
Yes! E.g. through function composition.

Function Composition

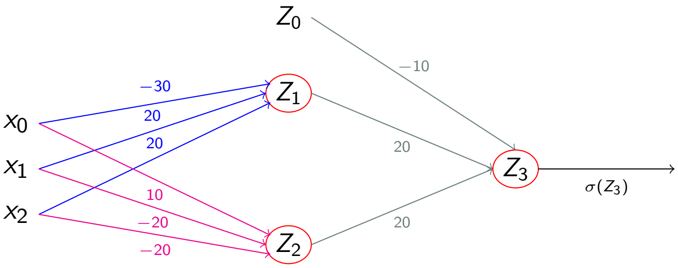


x_0	x_1	x_2	$x_1 \wedge x_2$
1	0	0	≈ 0
1	0	1	≈ 0
1	1	0	≈ 0
1	1	1	≈ 1



x_0	x_1	x_2	$\neg x_1 \wedge \neg x_2$
1	0	0	≈ 1
1	0	1	≈ 0
1	1	0	≈ 0
1	1	1	≈ 0

Function Composition



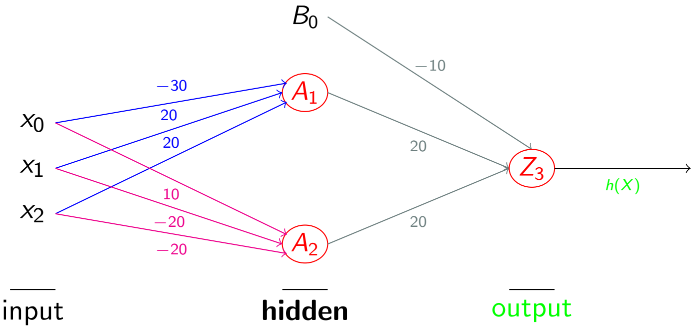
x_0	x_1	x_2	$\sigma(Z_1)$	$\sigma(Z_2)$	$\sigma(Z_3)$
1	0	0	≈ 0	≈ 1	$\sigma(1 * -10 + 0 * 20 + 1 * 20) = \sigma(10) \approx 1$
1	0	1	≈ 0	≈ 0	$\sigma(1 * -10 + 0 * 20 + 0 * 20) = \sigma(-10) \approx 0$
1	1	0	≈ 0	≈ 0	$\sigma(1 * -10 + 0 * 20 + 0 * 20) = \sigma(-10) \approx 0$
1	1	1	≈ 1	≈ 0	$\sigma(1 * -10 + 1 * 20 + 1 * 20) = \sigma(30) \approx 1$

Feedforward Neural Network

We just created a **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:
  - $Z_3 = \mathbf{H} \cdot \Theta_3$

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

Classification task: Do we have an $<$ time $>$ tag in the current position?

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.

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.

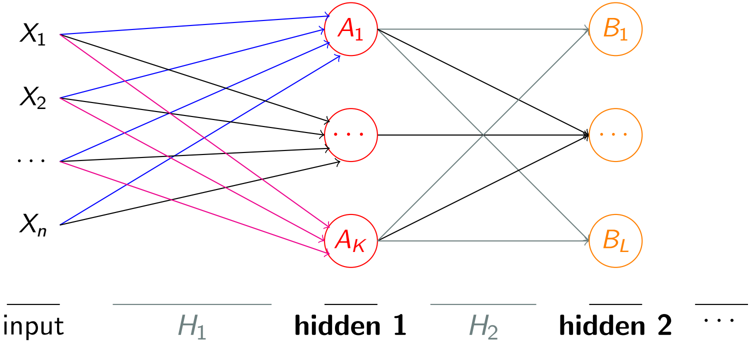
$\rightarrow$ Computation of $h(X)$ called forward propagation

$\rightarrow$ $U = (\Theta_1, \Theta_2)$ and $V = \Theta_3$ with error back propagation

Will be covered in lecture about training of neural networks

Network architectures

Depending on task, a particular network architecture can be chosen:



Note: Bias terms omitted for simplicity

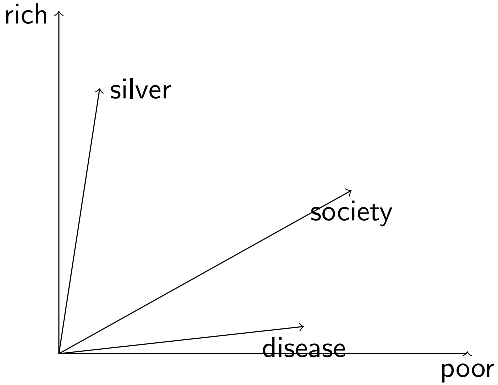
Multi-class classification

- More than two labels
- Instead of "yes" and "no", predict $c_i \in C = \{c_1, \cdots, c_k\}$
- Not just <time> label but also <etime>,<\etime>,...
- **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

Word Embeddings

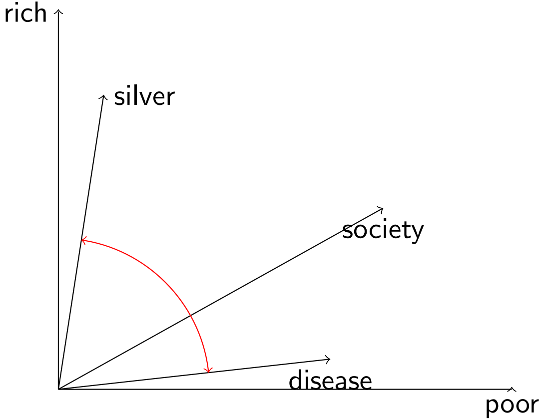
Word Embeddings

- 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



Underlying thoughts

- Assume the equivalence of:
  - Two words are semantically similar.
  - Two words occur in similar contexts (Miller & Charles, roughly).
  - Two words have similar word neighbors in the corpus.
- Elements of this are from Leibniz, Harris, Firth, and Miller.
- Strictly speaking, similarity of neighbors is neither necessary nor sufficient for semantic similarity.
- But perhaps this is good enough.

Adapted slide from Hinrich Schütze

Learning word embeddings

Count-based methods:

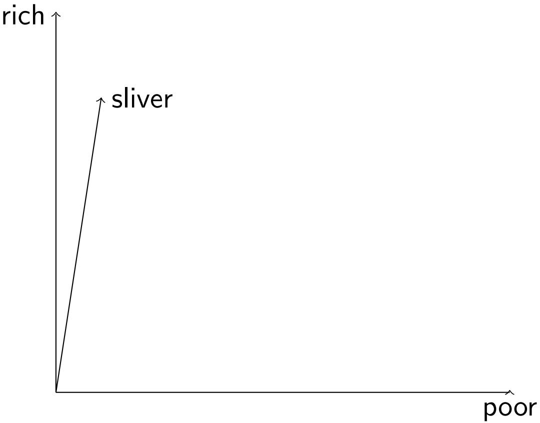
- Compute cooccurrence statistics
- Learn high-dimensional representation
- Map sparse high-dimensional vectors to small dense representation

Word cooccurrence in Wikipedia

- corpus = English Wikipedia
- cooccurrence defined as occurrence within $k = 10$ words of each other
  - cooc.(rich,silver) = 186
  - cooc.(poor,silver) = 34
  - cooc.(rich,disease) = 17
  - cooc.(poor,disease) = 162
  - cooc.(rich,society) = 143
  - cooc.(poor,society) = 228

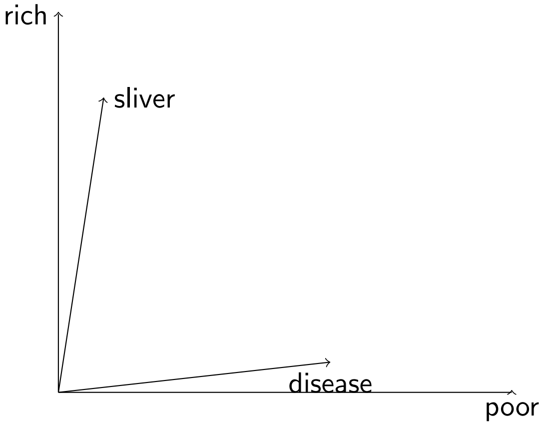
Adapted slide from Hinrich Schütze

Coocurrence-based Word Space



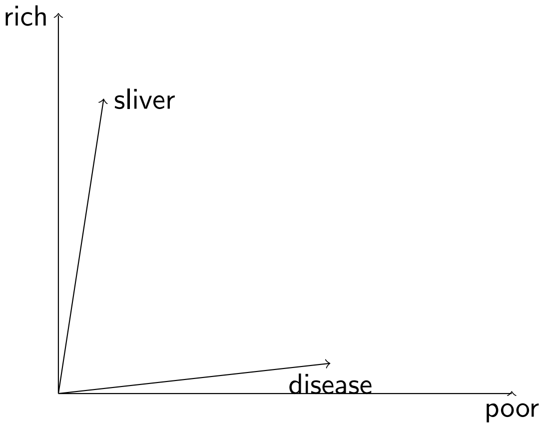
cooc.(poor,silver)=34,cooc.(rich,silver)=186

Coocurrence-based Word Space



cooc.(poor,disease)=162,cooc.(rich,disease)=17.

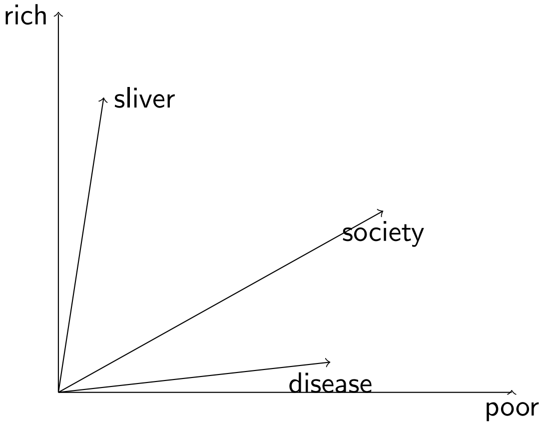
Exercise



ccooc.(poor,society)=228, cooc.(rich,society)=143

How is it represented?

Coocurrence-based Word Space



cooc.(poor,society)=228, cooc.(rich,society)=143

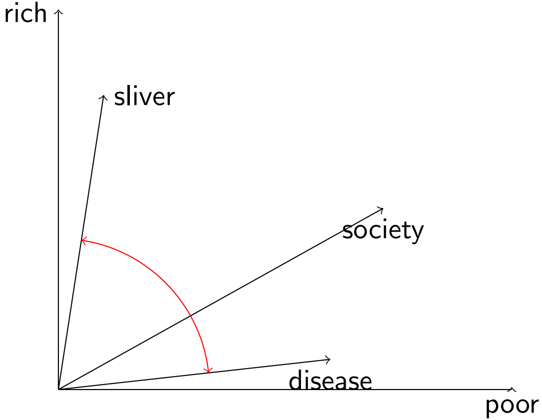
Dimensionality of word space

- Up to now we've only used two dimension words: rich and poor.
- Do this for all possible words in a corpus $\rightarrow$ high-dimensional space
- Formally, there is no difference to a two-dimensional space with three vectors.

- Note: a word can have a dual role in word space.
  - Each word can, in principle, be a dimension word, an axis of the space.
  - But each word is also a vector in that space.

Adapted slide from Hinrich Schütze

Semantic similarity



Similarity is the cosine of the angle between two word vectors

A neural language model

Neural language model

- Early application of neural networks (Bengio et al. 2003)
- Task: Given k previous words, predict the current word

  Estimate: $P(w_t | w_{t-k}, \cdots, w_{t-2}, w_{t-1})$

- Previous (non-neural) approaches:

  Problem: Joint distribution of consecutive words difficult to obtain
  → chose small history to reduce complexity (n=3)
  → predict for unseen history through back-off to smaller history

  Drawbacks:

  Takes into account small context
  Does not model similarity between words

Word similarity for language modeling

1. The cat is walking in the bedroom
2. The dog was running in a room
3. A cat was running in a room
4. A dog was walking in a bedroom

→ Model similarity between (cat,dog), (room, bedroom)

→ Generalize from 1 to 2 etc.

Neural language model

- Solution:

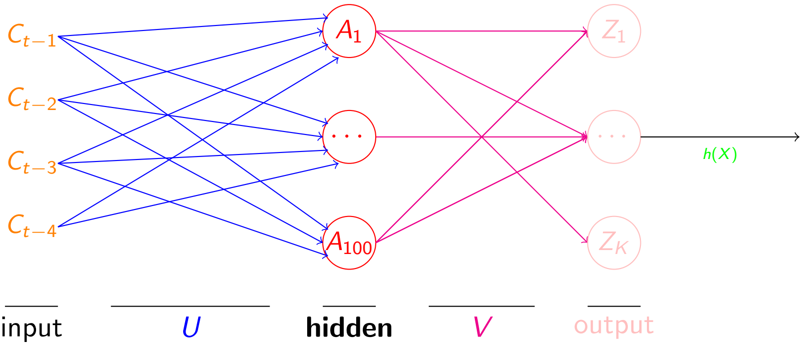
  Use word embeddings to represent each word in history

  $\rightarrow$ Each word is represented in relation to the others

  $\rightarrow$ Distributed word feature vector

  Feed to a neural network to learn parameters for the LM task

Feedforward Neural Network



Given words w_{t-4}, w_{t-3}, w_{t-2} and w_{t-1}, predict w_t
Note: Bias terms omitted for simplicity

Feedforward Neural Network

Input layer (X): Word embeddings C_{t-4}, C_{t-3}, C_{t-2} and C_{t-1}

Weight matrices U, V

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

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

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

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

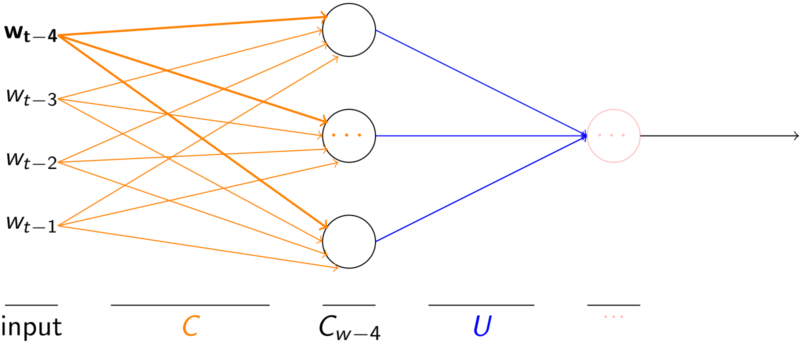
Getting the Word Embeddings

How are word embeddings $C_{t-4}, C_{t-3}, \cdots$ obtained?
$\rightarrow$ Parameter C of the model **learned** together with others (U and V)

- $C(i)$ is dot product of weight matrix C with index of w_i
- C is **shared** among all words
  - $W = \{\text{dog,cat,kitchen,table,chair}\}$, $w_{table} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

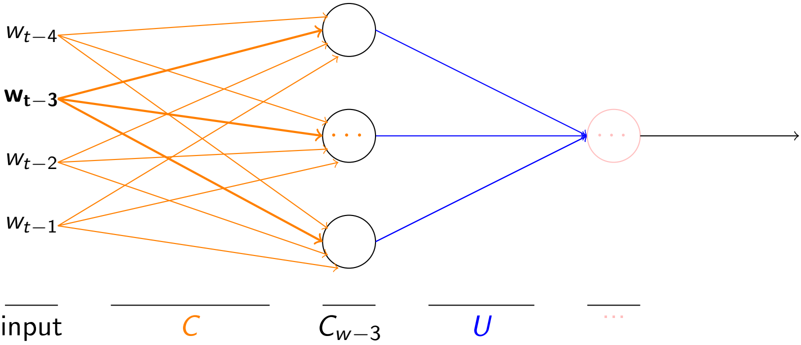
Note: There is **no** non-linearity here

Getting the Word Embeddings



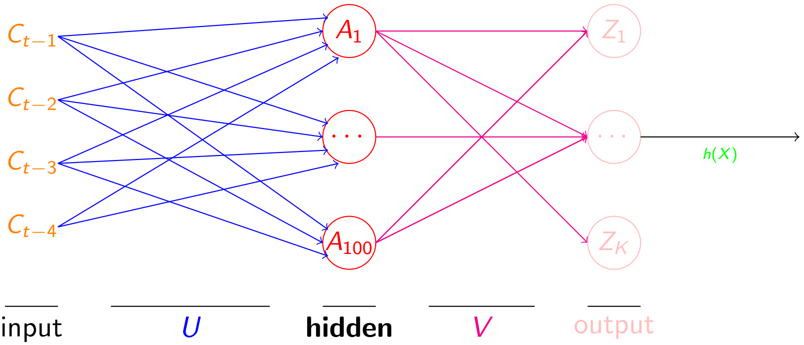
Note: Bias terms omitted for simplicity

Context vectors



Note: Bias terms omitted for simplicity

Feedforward Neural Network



Given words w_{t-4}, w_{t-3}, w_{t-2} and w_{t-1}, predict w_t

Note: Bias terms omitted for simplicity

Getting the right weights

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

- → **correct answer:** word at position t
- → Given a set T of training examples $t_1, \cdots t_n$ with **correct labels $\mathbf{y_i}$** ($\mathbf{w_t}$), find C, U, V (and biases b, d) such that $h(X) = \mathbf{y_i}$ for as many t_i as possible.
  - → *forward propagation* to compute $h(X)$
  - → *back propagation*of error to find best C, U, V (and biases b, d)

Neural language model

- Beats benchmarks
- Learned matrix C gives good word embeddings!

Learning word embeddings

Learning word embeddings

Count-based methods:

- Compute cooccurrence statistics
- Learn high-dimensional representation
- Map sparse high-dimensional vectors to small dense representation

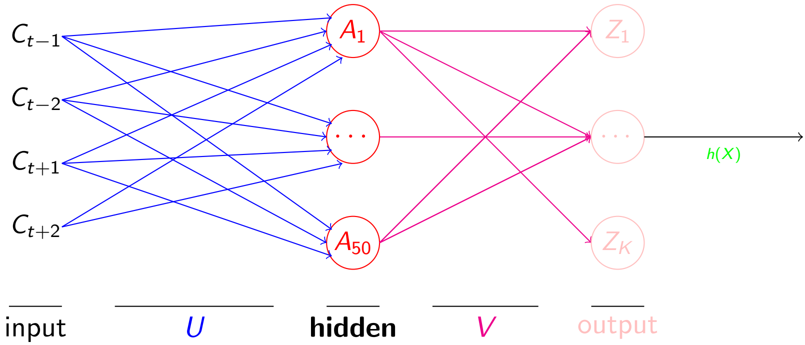
Neural networks:

- 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**

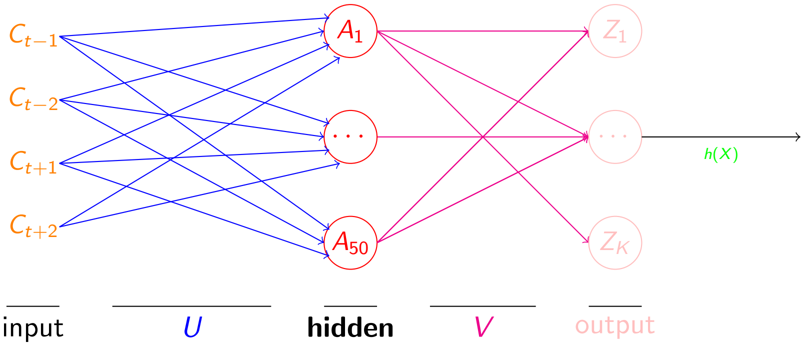
Network architecture



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

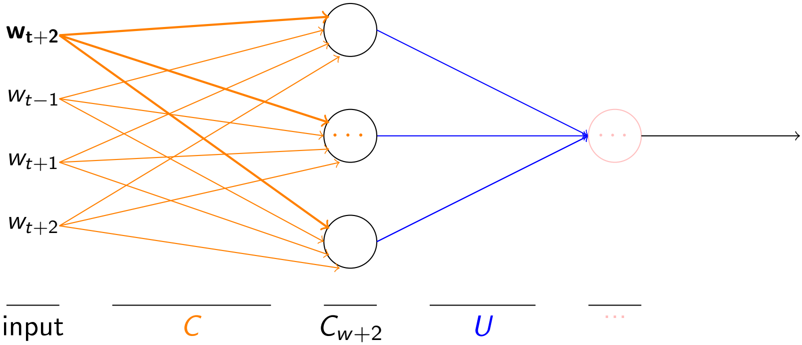
Note: Bias terms omitted for simplicity

Network architecture



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

Getting the Word Embeddings

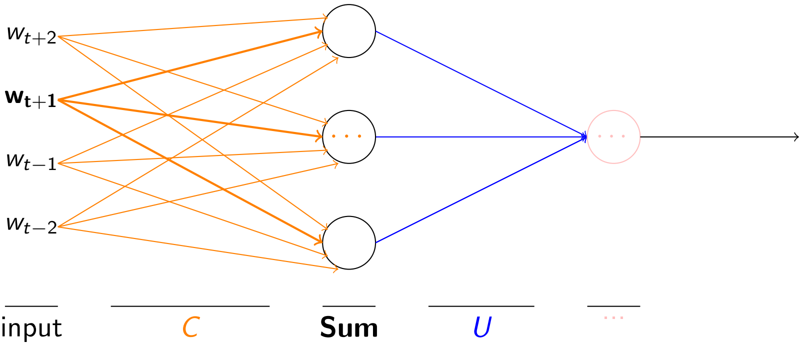


Note: Bias terms omitted for simplicity

Simplifications

- Remove hidden layer
- Sum over all projections

Simplifications



Remove hidden layer and sum over context

Note: Bias terms omitted for simplicity

Simplifications

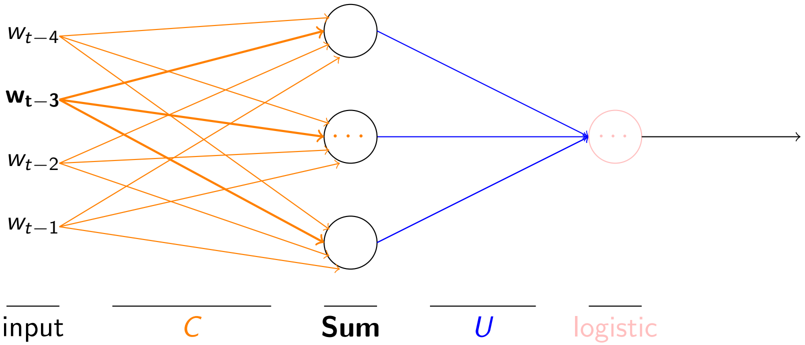
- Single logistic unit instead of output layer

  $\rightarrow$ 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

- BOW model (Mikolov. 2013)
- Skip-gram model:
  - Input is w_t
  - Prediction is w_{t+2}, w_{t+1}, w_{t-1} and w_{t-2}

Applications

Semantic similarity:

- How similar are the words:
  - *coast* and *shore*; *rich* and *money*; *happiness* and *disease*; *close* and *open*
- WordSim-353 (Finkelstein et al. 2002)
  - Measure associations
- SimLex-999
  - Only measure semantic similarity

Other tasks:

- Use word embeddings as input features for other tasks (e.g. sentiment analysis, language modeling)

Recap

- Cannot fit data with **non-linear** decision boundary with linear models

  Solution: compose non-linear functions with neural networks

  → Successful in many NLP applications:
    - Language modeling
    - Learning word embeddings

- Feeding word embeddings into neural networks has proven successful in many NLP tasks
    - Sentiment analysis

Thank you !