Feedforward Neural Networks and Word Embeddings

Prof. Dr. Alexander Fraser
(slides originally by Dr. Fabienne Braune)

CIS, LMU Munich

WS 2023/2024

Outline

Linear Models

Binary Classification with Linear Models

Example: the seminar at < stime > 4 pm will

Classification task: Do we have an < stime > 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 weight vector

$$h(X) = X\Theta^T$$
$$= X \cdot \Theta$$

$$X = \begin{bmatrix} x_0 = 1 \\ x_1 = 1 \\ x_2 = 0 \\ \dots \\ x_{101} = 1 \\ x_{102} = 0 \\ \dots \\ x_{201} = 1 \\ x_{202} = 0 \\ \dots \\ x_{301} = 1 \\ x_{302} = 0 \\ \dots \end{bmatrix}$$

$$\Theta = \begin{bmatrix} w_0 = 1.00 \\ w_1 = 0.01 \\ w_2 = 0.01 \\ \dots \\ x_{101} = 0.01 \\ x_{102} = 0.01 \\ \dots \\ x_{201} = 0.01 \\ x_{202} = 0.01 \\ \dots \\ x_{301} = 0.01 \\ x_{302} = 0.01 \\ \dots \end{bmatrix}$$

Prediction with dot product

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

Predictions with linear models

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

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

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

Prediction: If $h(X) > 0$, 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.

- → 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.
- → $X(t_i)$ is the feature vector for the i-th training example t_i

Dot product with trained weight vector

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

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

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

Working with real-valued features

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

Working with real-valued features

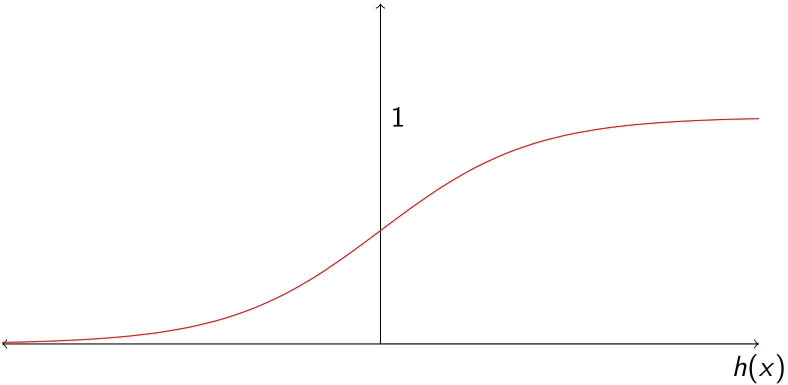
Classification task: Do we have an $< \text{stime} >$ tag in the current position?

Prediction: $h(X) = 540.5$

- Can we transform this into a probability?

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 $<$ stime $>$ tag in the current position?

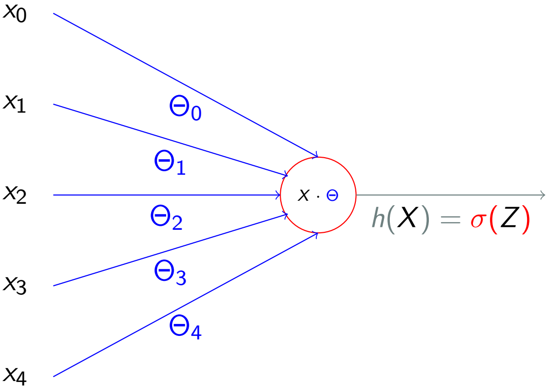
Linear Model: $Z = X \cdot \Theta$

Prediction: If $Z > 0$, 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

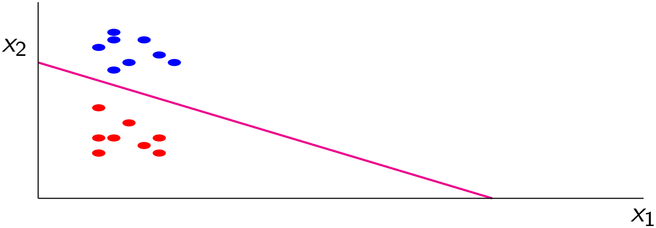
Decision Boundary

What do **linear** models do?

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

  positive examples (have stime tag)

  negative examples (no stime tag)

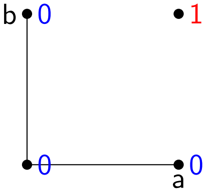


Exercise

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

Modeling 1: Learning a predictor for $\wedge$

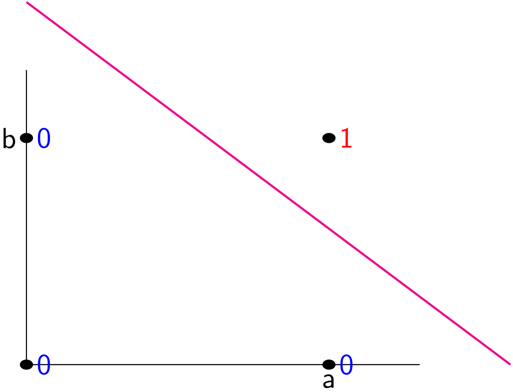
a	b	a $\wedge$ b
0	0	0
0	1	0
1	0	0
1	1	1



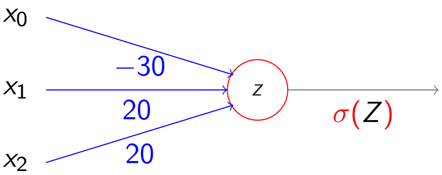
Features : a, b   Feature values : binary

Can we learn a linear model to solve this problem?

Modeling 1: Learning a predictor for $\wedge$



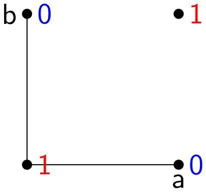
Modeling 1: Logistic Regression



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	$\sigma(1 * -30 + 1 * 20 + 0 * 20) = \sigma(-10) \approx 0$
1	1	1	$\sigma(1 * -30 + 1 * 20 + 1 * 20) = \sigma(10) \approx 1$

Modeling 2: Learning a predictor for *XNOR*

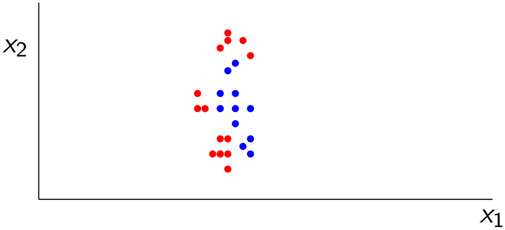
a	b	a *XNOR* b
0	0	1
0	1	0
1	0	0
1	1	1



Features : a, b   Feature values : binary

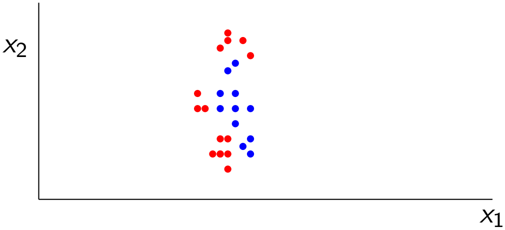
Can we learn a linear model to solve this problem?

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*

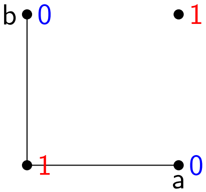
Linear models not suited to learn non-linear decision boundaries.

Neural networks can do that.

Neural Networks

Learning a predictor for *XNOR*

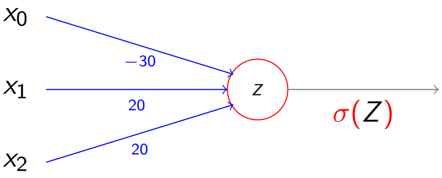
a	b	a *XNOR* b
0	0	1
0	1	0
1	0	0
1	1	1



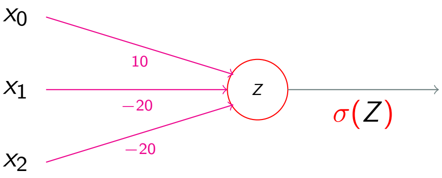
Features : a, b   Feature values : binary

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

Function Composition

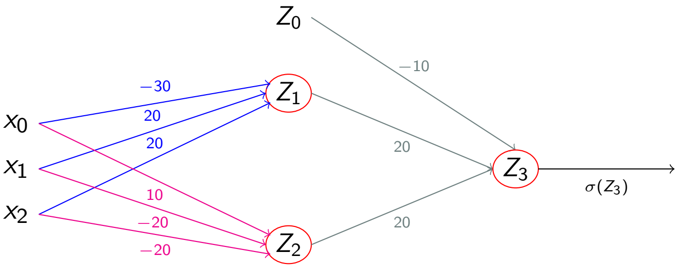


x_0	x_1	x_2	$x_1 \wedge x_2$
1	0	0	≈ 0
1	0	1	≈ 0
1	1	0	≈ 0
1	1	1	≈ 1



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



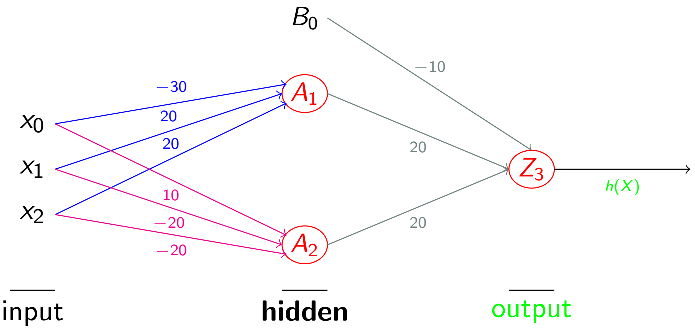
x_0	x_1	x_2	$\sigma(Z_1)$	$\sigma(Z_2)$	$\sigma(Z_3)$
1	0	0	≈ 0	≈ 1	$\sigma(1 * -10 + 0 * 20 + 1 * 20) = \sigma(10) \approx 1$
1	0	1	≈ 0	≈ 0	$\sigma(1 * -10 + 0 * 20 + 0 * 20) = \sigma(-10) \approx 0$
1	1	0	≈ 0	≈ 0	$\sigma(1 * -10 + 0 * 20 + 0 * 20) = \sigma(-10) \approx 0$
1	1	1	≈ 1	≈ 0	$\sigma(1 *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

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):

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

General Feedforward Neural Network

Classification task: Do we have an $<$ stime $>$ 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.

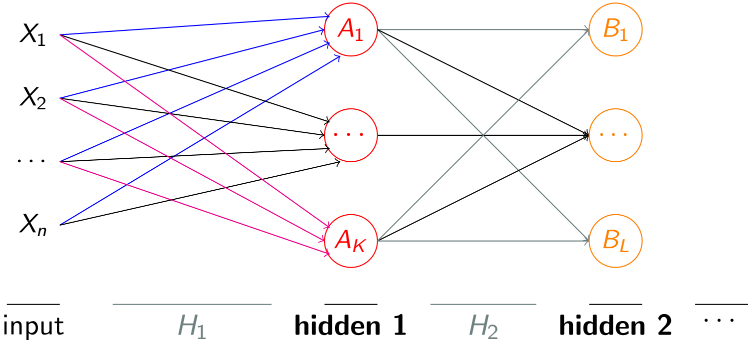
$\rightarrow$ Computation of $h(X)$ called forward propagation

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

The intuition behind back propagation is the same as the perceptron update!

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 $c_i \in C = \{c_1, \cdots, c_k\}$, where k is the number of classes
- For instance, if we want to detect border tags for stime and etime, then we don't only have the <stime> label but also: </stime>, <etime>, </etime>, **no tag**
- **Use 5 output units** (5 is the number of classes)
  - Output layer instead of a single output unit
  - The class with the highest activation is chosen
  - Probabilities can be obtained by dividing the exponentiated activation for a class by the sum of the exponentiated activations ("softmax")

Summary: Neural Networks

- We showed how to use neural networks to solve non-linear decision problems
- Neural networks are very powerful - much more powerful than linear models, even more powerful than decision trees
- But we have been working with very simple features (binary features so far in our example).
- Neural networks can combine these simple features into very complex features (as was done previously with feature selection)
- But now we will show how neural language modeling led to the development of very powerful features, “word embeddings”, which are associated with word types

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

$\rightarrow$ Model similarity between (cat,dog), (room, bedroom)

$\rightarrow$ Generalize from 1 to 2 etc.

Neural Language Model (LM)

- Solution:

  Use word embeddings to represent each word in history

  → Each word is represented in relation to the others

  → Distributed word feature vector

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

Feedforward Neural Network for LM

Training example: *The cat is walking in the bedroom*

Neural network input:

Look at words preceeding bedroom

$\rightarrow$ The cat **is**$_{-4}$ **walking**$_{-3}$ **in**$_{-2}$ **the**$_{-1}$ **bedroom**

$\rightarrow$ Create word embedding (LT_i) for window

Give LT_i as input to Feedforward Neural Network

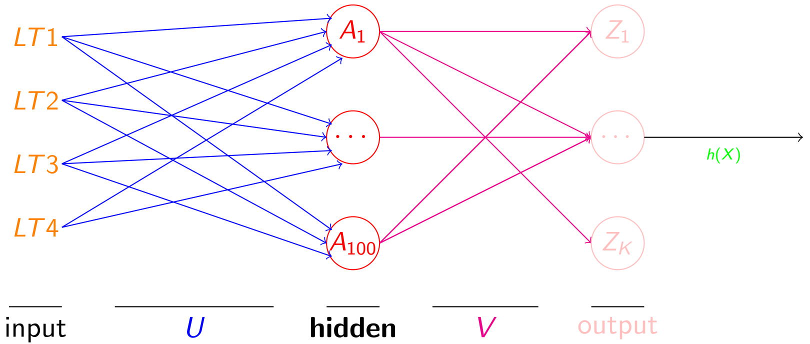
Neural network training:

Predict current word (forward propagation)

$\rightarrow$ should be bedroom

Train weights by backpropagating error

Feedforward Neural Network for LM



Input: word embeddings LT_i
Output: predicted label (current word)

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

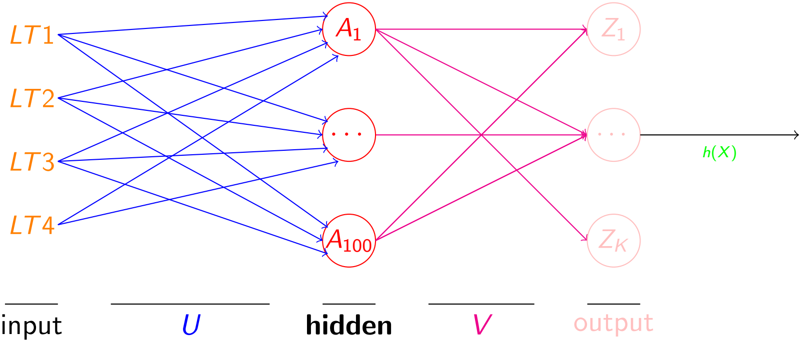
Training: Find weight matrices U and V such that $h(X)$ is the **correct answer** as many times as possible.

→ 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.

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

→ U and V with error back propagation

Forward Propagation



Forward propagation:

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

Forward Propagation

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

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)

Backpropagation

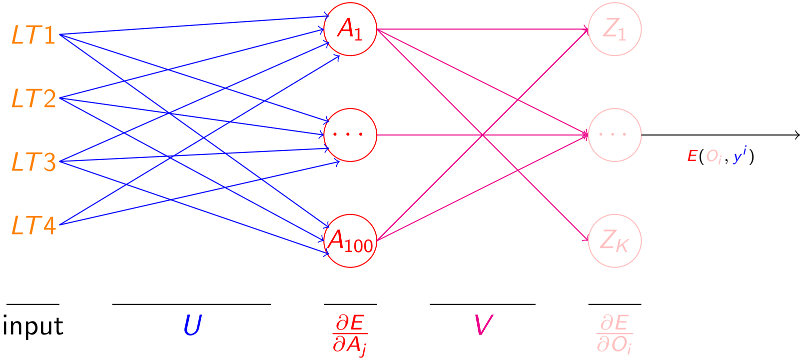
Goal of training: adjust weights such that correct label is predicted

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

Sketch:

- Convert difference between prediction and error into **derivatives**
- Compute **derivatives** in each hidden layer from layer above
  - Backpropagate the error derivative with respect to the output of a unit
- Use derivatives with respect to the activations to get error derivatives with respect to incoming weights

Backpropagation



Backpropagation:

$\rightarrow$ Compute E

$\rightarrow$ Compute $\frac{\partial E}{\partial O_i}$

Backpropagation

Compute error at output E:

Compare output unit with y^i

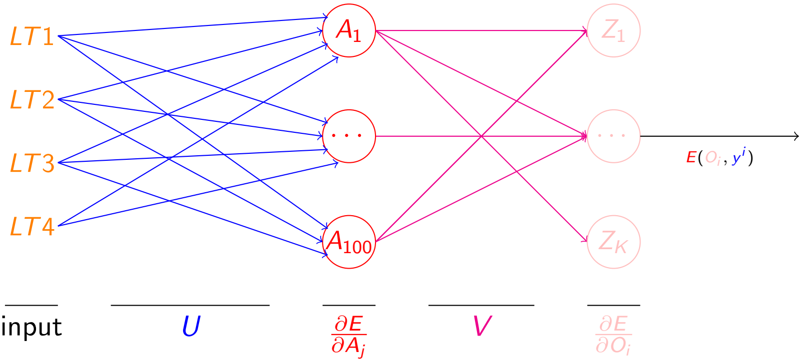
- y^i vector with 1 in correct class, 0 otherwise

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

Compute $\frac{\partial E}{\partial O_i}$:

$\frac{\partial E}{\partial O_i} = -(y_i - O_i)$

Backpropagation



Backpropagation:

$\rightarrow$ Compute $\frac{\partial E}{\partial A_j}$

Backpropagation

Compute **derivatives** in each hidden layer from layer above:

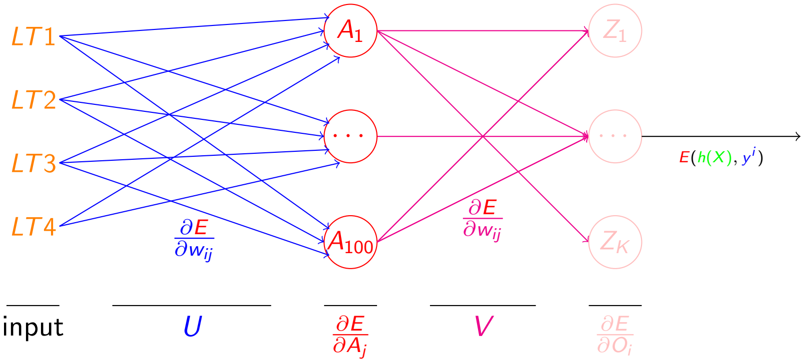
Compute derivative of error with respect to logit (output)

Compute derivative of error with respect to previous hidden unit

Compute derivative with respect to weights

→ Use **recursion** to do this for every layer

Backpropagation



Weight training

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

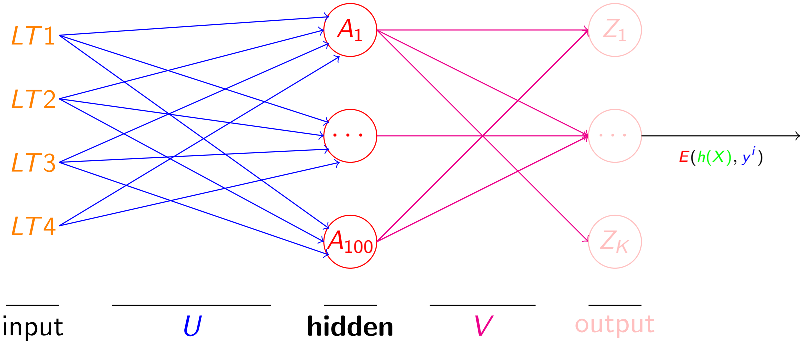
→ Computation of $h(X)$ with forward propagation

→ 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

Word Embedding Layer



Word Embedding Layer

- Each word type 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**. Each column in C is used for all words (tokens) of a particular word-type.

Dot product with (trained) weight vector

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

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

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

Words get mapped to lower dimension
$\rightarrow$ Hyperparameter to be set

Dot product with (initial) weight vector

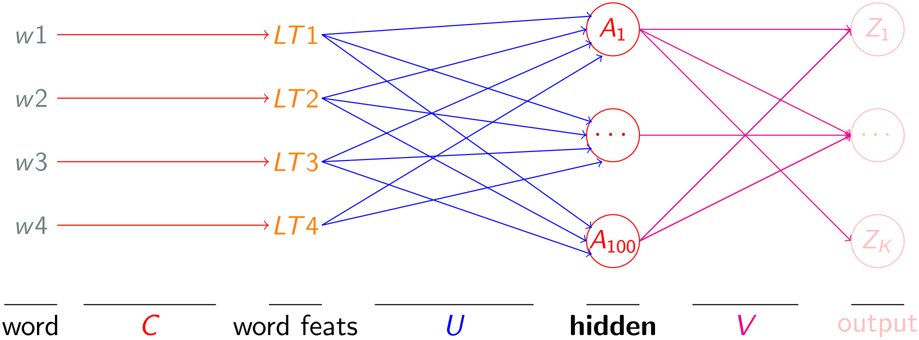
$W = \{\text{the,cat,on,table,chair}\}$

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

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

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.

→ 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.

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

→ Modify C, U and V with error back propagation

Dot product with (trained) weight matrix

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

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

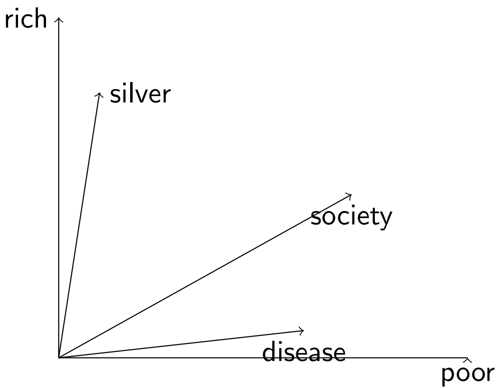
$$LT_{table} = w_{table} \cdot C = \begin{bmatrix} 0.03 \\ 0.02 \\ 0.04 \end{bmatrix}$$

Each word type gets a specific feature vector

Word Embeddings

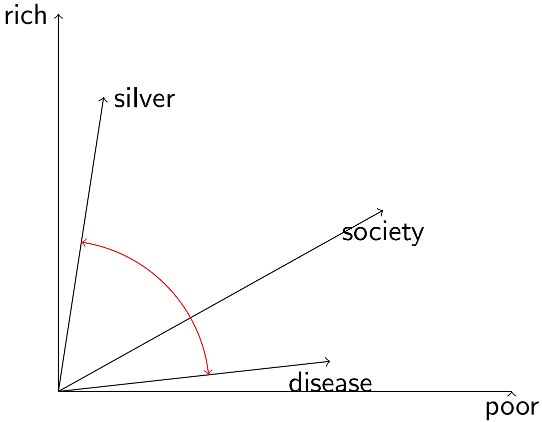
Word Embeddings

- 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



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:

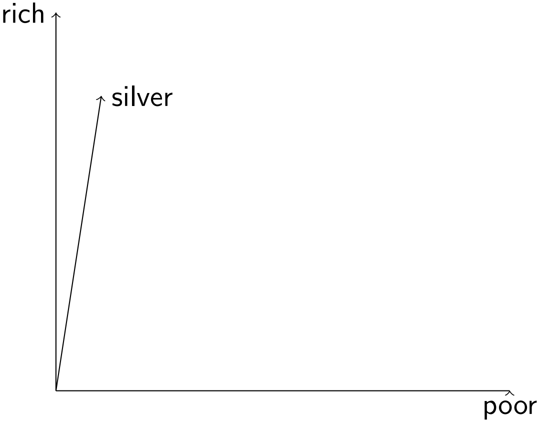
- Compute cooccurrence statistics
- Learn high-dimensional representation
- Map sparse high-dimensional vectors to small dense representation

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

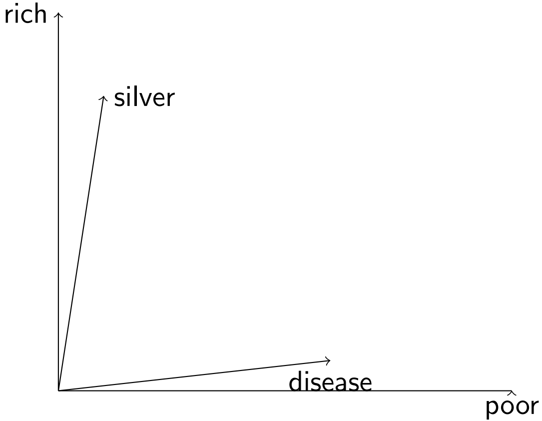
Adapted slide from Hinrich Schütze

Coocurrence-based Word Space



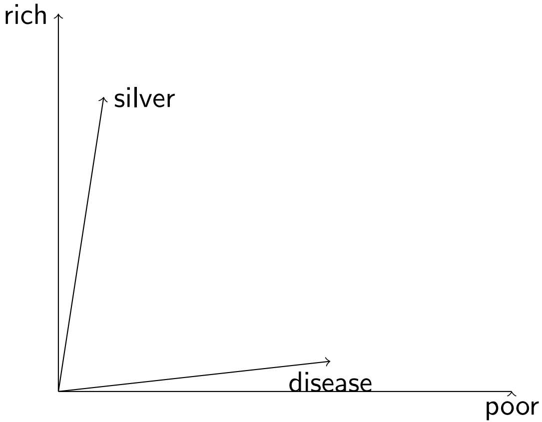
cooc.(poor,silver)=34,cooc.(rich,silver)=186

Coocurrence-based Word Space



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

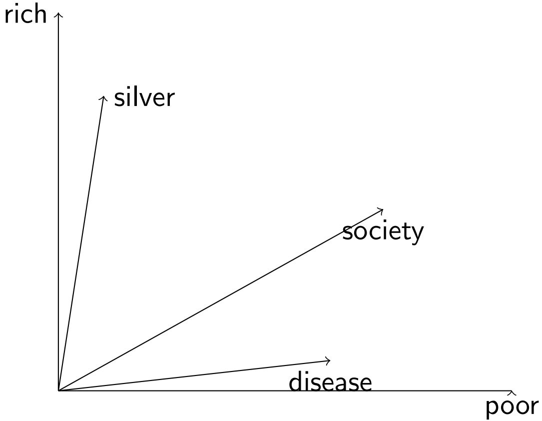
Exercise



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

How is it represented?

Coocurrence-based Word Space



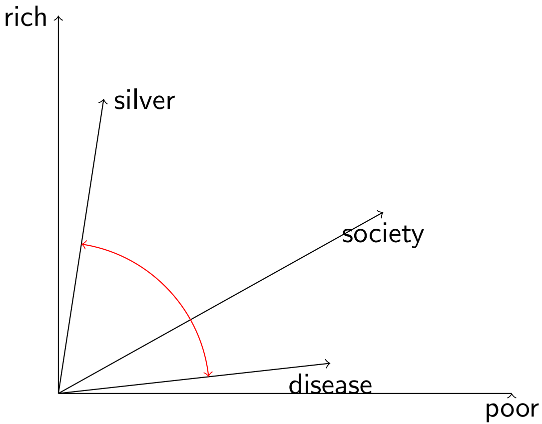
cooc.(poor,society)=228, cooc.(rich,society)=143

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 → 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

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

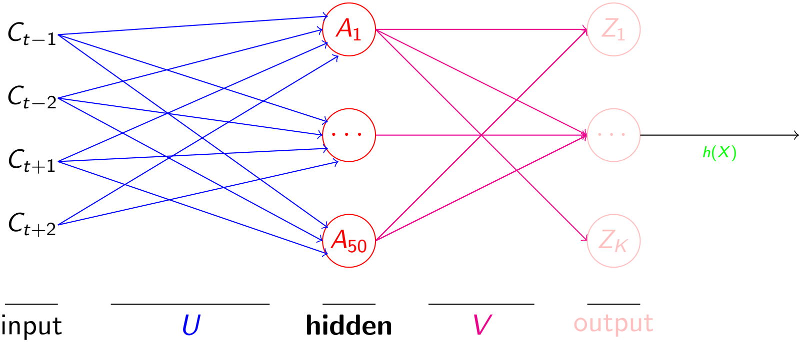
  → For each word w in V, model $P(w_t | w_{t-1}, w_{t-2}, ..., w_{t-n})$

  → **Learn embeddings C of words**

- Word embeddings learning task: Given *k* context words, predict the current word

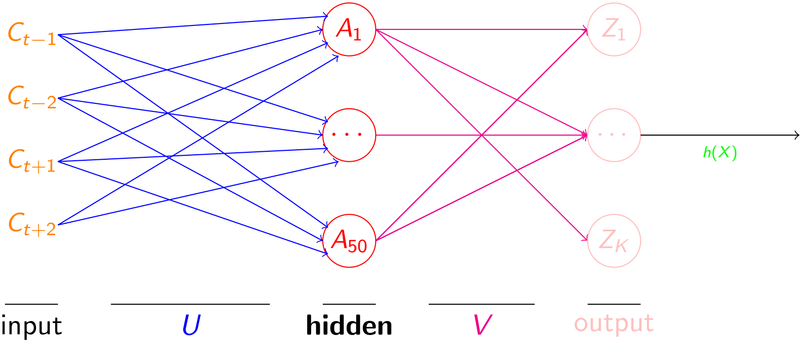
  → **Learn embeddings C of words**

Network architecture



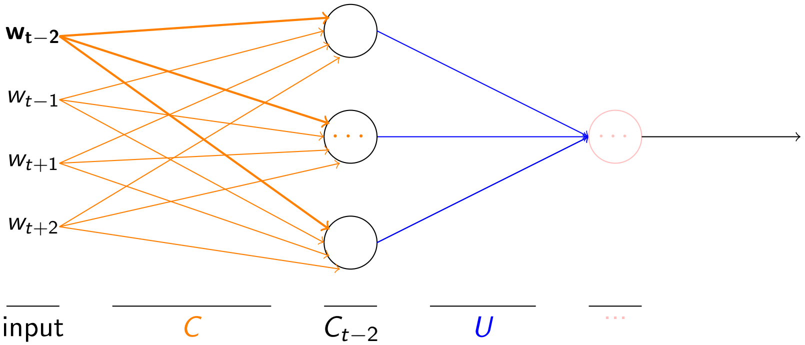
Given words w_{t-2}, w_{t-1}, w_{t+1} and w_{t+2}, predict w_t ("CBOW")
Note: Bias terms omitted for simplicity

Network architecture



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

Getting the Word Embeddings

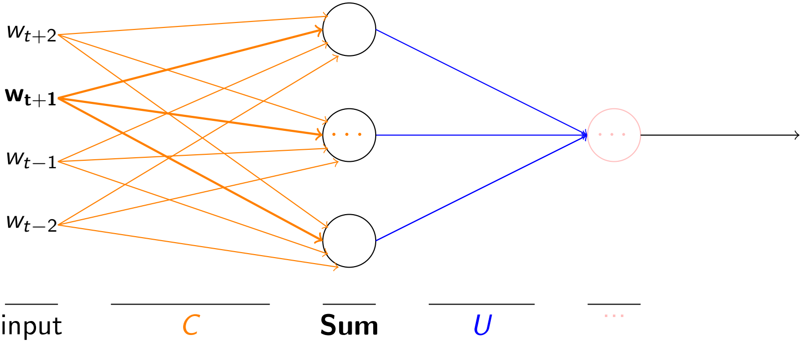


Note: Bias terms omitted for simplicity

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

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, named entity recognition)

Recap

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

  Solution: compose non-linear functions with neural networks
  $\rightarrow$ Successful in many NLP applications:
  - Language modeling
  - Learning word embeddings

- Feeding word embeddings into neural networks has proven successful in many NLP tasks, e.g.:
  - Sentiment Analysis
  - Named Entity Recognition

Questions?

Some Further Issues

- The backup slides (at the end) show the details of backpropagation, it is a good idea to look at these.
- Neural networks can be shown to approximate any function arbitrarily well. See the intuitive discussion of this property in this online book, in chapter 4:
    - Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015.
    - `http://neuralnetworksanddeeplearning.com/chap4.html`
- I also highly recommend the other chapters in this book!

Thank you for your attention.

Backpropagation - Details

- The next slide shows the actual computation of backpropagation, showing the derivatives that are computed.
- The actual updates are also shown, these are more intuitive than the derivatives for many people.

Backpropagation

Compute **derivatives** in each hidden layer from layer above:

Compute derivative of error with respect to logit (output)

$\frac{\partial E}{\partial Z_i} = \frac{\partial E}{\partial O_i}\frac{\partial O_i}{\partial Z_i} = \frac{\partial E}{\partial O_i} O_i(1 - O_i)$ (Note: $O_i = \frac{1}{1+e^{-Z_i}}$)

Compute derivative of error with respect to previous hidden unit

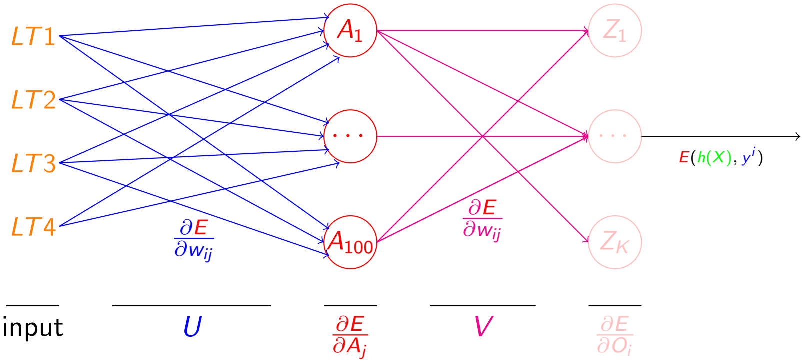
$\frac{\partial E}{\partial A_j} = \sum_i \frac{\partial Z_i}{\partial A_j}\frac{\partial E}{\partial Z_i} = \sum_i w_{ji}\frac{\partial E}{\partial Z_i}$

Compute derivative with respect to weights

$\frac{\partial E}{\partial w_{ji}} = \frac{\partial Z_i}{\partial w_{ji}}\frac{\partial E}{\partial Z_i} = O_i\frac{\partial E}{\partial Z_i}$

→ Use **recursion** to do this for every layer

Backpropagation



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