Recap: Machine Learning

Decision Tree

Naive Bayes

- Probabilistic method for classification
- Naive because we ignore feature dependencies
- Prediction model:

$$\underset{c \in C}{\operatorname{argmax}}\, p(c | f_1(x), f_2(x), \ldots, f_n(x))$$

- Training: Count relative frequencies

Logistic Regression

- Regression method for binary classification
- Output numbers interpreted as probabilities
- Prediction model:

$$h(x) = \frac{1}{1 + e^{-(ax+b)}}$$

linear, Gleichung

- Training: Gradient descent with loss function

UNIVERSITÄT ZU KÖLN

Neural Networks

Sprachverarbeitung (VL + Ü)

Nils Reiter

June 25, 2024

INSTITUT FÜR DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Today

Section 1

Neural Networks

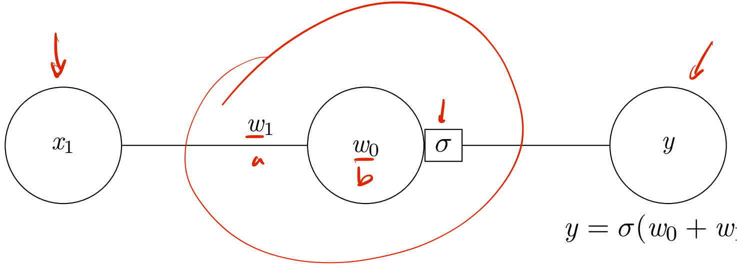
From a Logistic Regression to a Neuron

- Hypothesis function of logistic regression:

$$h(x) = \overset{1}{\sigma}(\underline{\overset{b}{w_0} + \overset{a}{w_1}\overset{x}{x_1}}) \qquad \text{with } \sigma(x) = \frac{1}{1+e^{-x}}$$

- Maps one value to another (just like many other functions)

What is a Neural Network?



$$y = \sigma(w_0 + w_1 x)$$

Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

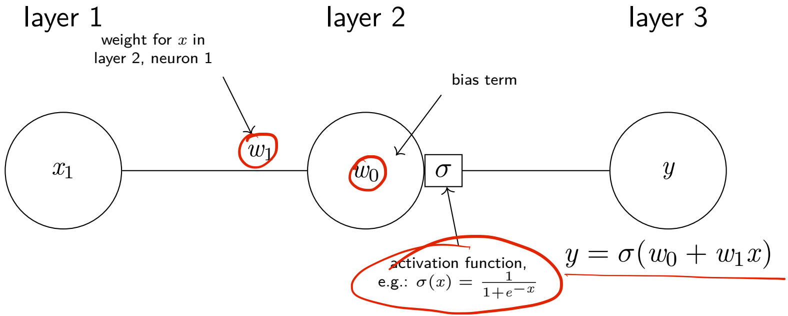


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

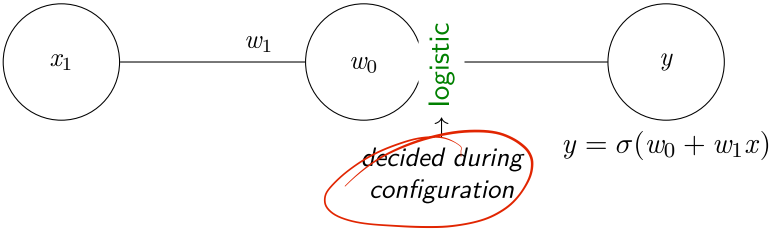


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

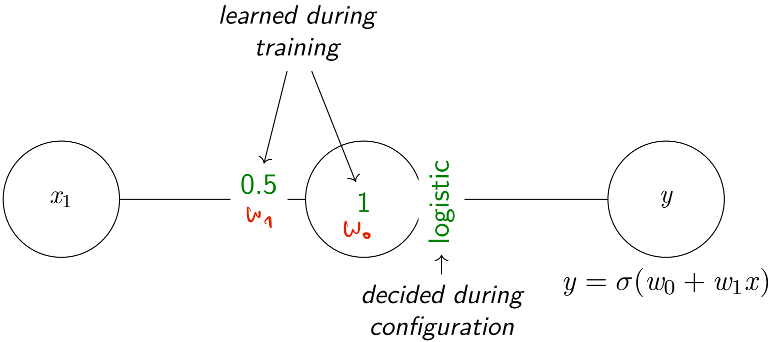


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

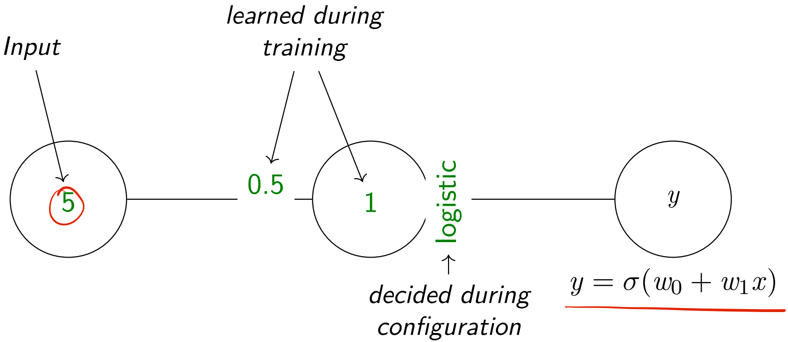


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

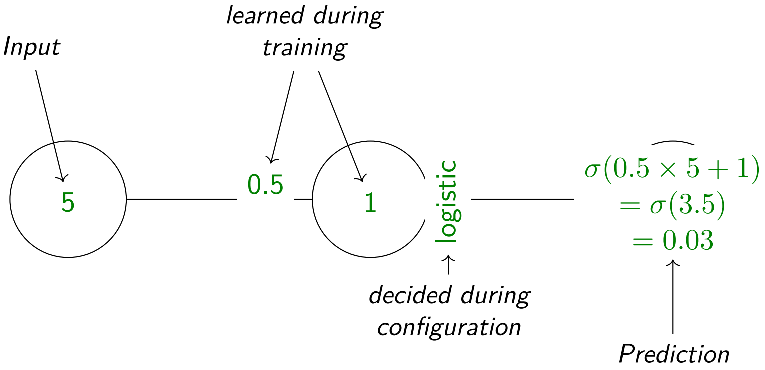


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Straightforward to extend to multiple features

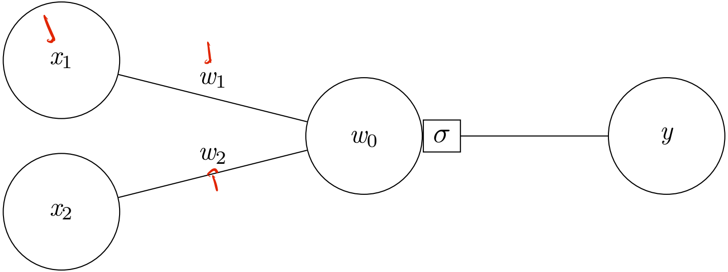
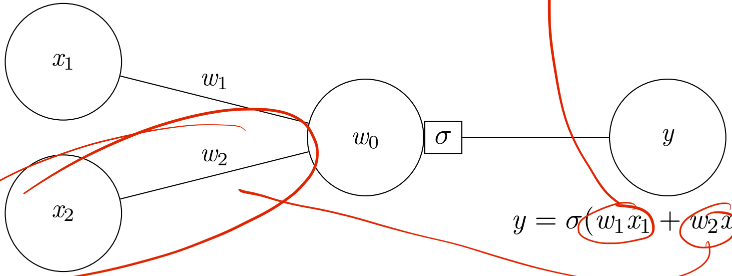


Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features

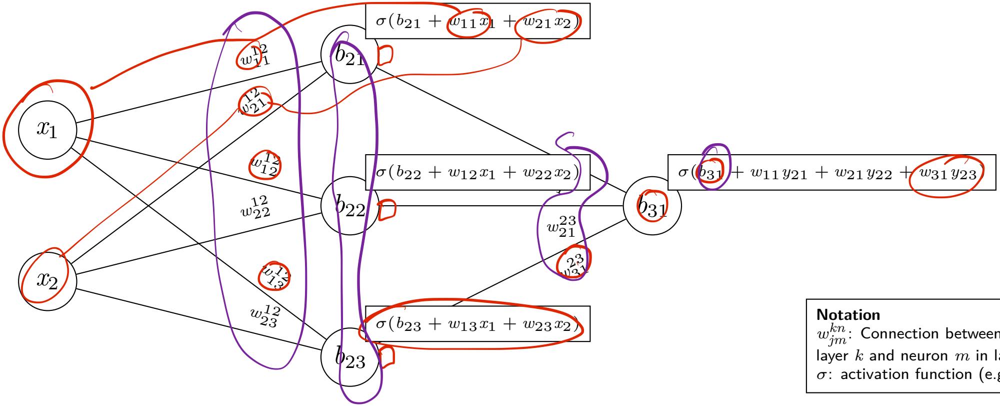


$$y = \sigma(w_1 x_1 + w_2 x_2 + w_0)$$

Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features and multiple regression nodes



Notation
w_{jm}^{kn}: Connection between neuron j in layer k and neuron m in layer n
σ: activation function (e.g., logistic)

Figure: A simple neural network with 1 hidden layer (and 13 parameters)

Prediction Model: Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)

Prediction Model: Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$

Prediction Model: Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation: $f_2(X) = \sigma((W_{1,2}^{\intercal} X) + B_2)$

Prediction Model: Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation: $f_2(X) = \sigma((W_{1,2}^{\intercal} X) + B_2)$
- Deep learning involves a lot of matrix multiplication
  - GPUs are highly optimized for this
  - Hint: Gaming-GPUs that support CUDA are also usable for deep learning

Feed-Forward Neural Networks

- The above is called a ›feed-forward neural network‹ (FFNN)
  - Information is fed only in forward direction

Feed-Forward Neural Networks

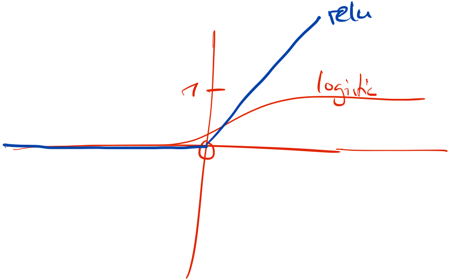
- The above is called a ›feed-forward neural network‹ (FFNN)
  - Information is fed only in forward direction
- Configuration choices
  - Activation function (next slide)
  - Layer size: Number of neurons in each layer
  - Number of layers
  - Loss function
  - Optimizer
- Training choices
  - Epochs/batches
  - Training status displays

demo

playground.tensorflow.org

Feed-Forward Neural Networks

Activation Functions



All neurons of one layer have the same

Popular choices:

- **logistic** $y = \sigma(x) = \frac{1}{1+e^{-x}}$ – ›squashes‹ everything to a value between 0 and 1
- **relu** $y = \max(0, x)$ – Makes everything negative to 0
- **softmax** Scales a vector such that values sum to 1 (probability distribution)

Training: »Back Propagation«

- Similar to gradient descent
- But
  - A lot more parameters
  - Because of multiple layers: Vanishing gradients
    - Back propagation involves a lot of multiplication
    - Factors are between zero and one
    - $\Rightarrow$ Numbers get very small very quickly

Training: »Back Propagation«

- Similar to gradient descent
- But
  - A lot more parameters
  - Because of multiple layers: Vanishing gradients
    - Back propagation involves a lot of multiplication
    - Factors are between zero and one
    - $\Rightarrow$ Numbers get very small very quickly
- Training choice: Batches and epochs

Training a Feedforward Neural Network I

Stochastic Gradient Descent (SGD)

- Gradient Descent
  - Apply θ to all training instances
  - Calculate loss over entire data set
- Stochastic Gradient Descent
  - Data set in random order
  - Calculate loss for every single instance, then update weights

Training a Feedforward Neural Network II

When to stop the training

- Logistic regression (last week): Stop in minimum
- In theory, that's what we want
- In practice
  - We usually are not exactly in the minimum
  - It's not important to be exactly in the minimum

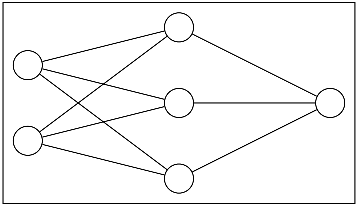
$\Rightarrow$ Fixed number of iterations over the data set (= number of epochs)

Batches vs. Epochs

batch Number of instances used before updating weights

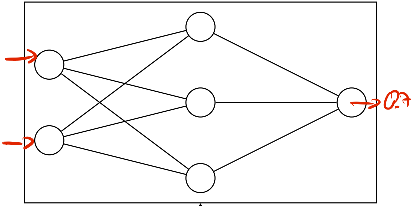
epochs Number of iterations over all instances

Dimensions

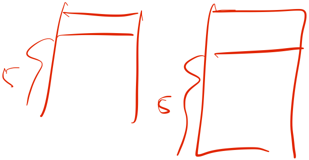


- Dimensionality of neural networks major source of confusion

Dimensions



- Dimensionality of neural networks major source of confusion
- In this example
  - Single input object represented with two numbers (= 1D)
  - Output is a single number
- Entire input data set: 2D (because multiple instances)

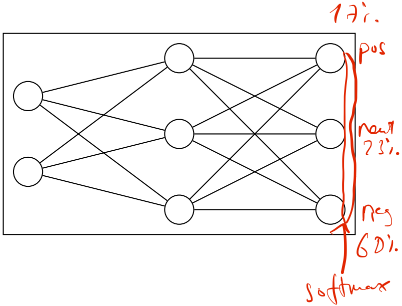


x_1	x_2	y
0.5	0.3	0.7
0.1	-0.2	0.2
⋮	⋮	⋮

Binary Classification

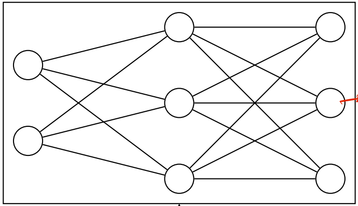
- So far: Binary classification
- Two classes, represented as 0 or 1, $Y = \{0, 1\}$
- Hypothesis function maps from n-dimensional input vector to $[0; 1]$
  - $h : \mathbb{R}^n \rightarrow [0; 1]$

Multi-class Classification



- Each class is represented by one output neuron
- Three classes (e.g., positive, neutral, negative)
- Activation function of last layer: softmax
  - Similar to sigmoid (i.e., everything is in $[0;1]$), *and*
  - Everything adds up to 1

Multi-class Classification



0.1
0.7
0.2

0
1
0

- Each class is represented by one output neuron
- Three classes (e.g., positive, neutral, negative)
- Activation function of last layer: softmax
  - Similar to sigmoid (i.e., everything is in $[0;1]$), *and*
  - Everything adds up to 1
- Input representation: One-hot-encoding
  - A vector with one dimension for each class
  - The element with the correct class is 1, all others are 0
  - E.g.: $[0,1,0]$ represents that the second class is correct

Section 2

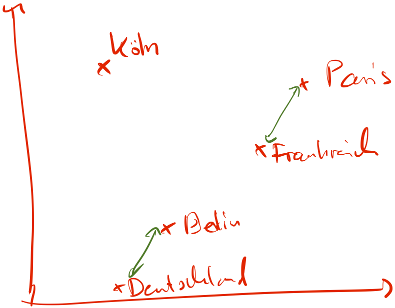
Word2Vec

Literature basis

- Two very influential papers by Mikolov et al.
  - T. Mikolov/K. Chen/G. Corrado/J. Dean (2013). »Efficient Estimation of Word Representations in Vector Space«. In: *ArXiv e-prints*
  - Tomas Mikolov/Ilya Sutskever/Kai Chen/Greg S Corrado/Jeff Dean (2013). »Distributed Representations of Words and Phrases and their Compositionality«. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges/L. Bottou/M. Welling/Z. Ghahramani/K. Q. Weinberger. Curran Associates, Inc., pp. 3111–3119
- Software package
  - word2vec – `https://github.com/tmikolov/word2vec`
    Originally published on »Google Code«

Basics

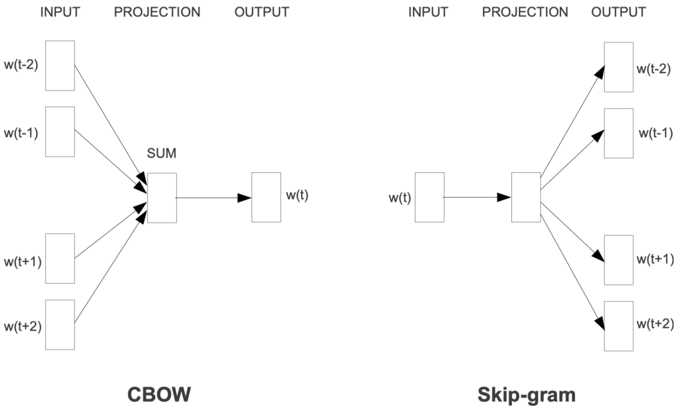
- Recap: First session
  - No interpretable dimensions
  - Dense vectors: No zeros, and much fewer dimensions than in count vectors



Basics

- Recap: First session
  - No interpretable dimensions
  - Dense vectors: No zeros, and much fewer dimensions than in count vectors
- Word2vec
  - Let's use the learned parameters as word vectors
    - (one parameter vector per word)
  - How to come up with a task that generates these parameters?
  - An application for neural networks

Two tasks



Continuous Bag of Words (CBOW)
Context words used to predict one word

Skip-Gram
One word used to predict its context

Skip-Gram

- Context: ± 2 words around target word t

```
... dogs, such as a  German Shepherd or a Labrador, ...
               c1 c2   t         c3    c4
```

Skip-Gram

- Context: ± 2 words around target word t

```
... dogs, such as a  German Shepherd or a Labrador, ...
              c1 c2   t         c3     c4
```

- Classifier:
  - Predict for any pair (t, c) wether c is *really* a context word for t
  - Formally: $p(+|\vec{t}, \vec{c})$
    - Probability of t and c being positive examples, using the respective vectors

Skip-Gram

- Context: ± 2 words around target word t

```
... dogs, such as a  German Shepherd or a Labrador, ...
              c1 c2   t        c3     c4
```

- Classifier:
  - Predict for any pair (t, c) wether c is *really* a context word for t
  - Formally: $p(+|\vec{t}, \vec{c})$
    - Probability of t and c being positive examples, using the respective vectors
- How can we determine probability, based on vectors?
- Vector similarity $\rightarrow$ probability
  - Measure for similarity of vectors? Dot product ➜
  - Dot product to probability? Logistic function ✔
  - »a word is likely to occur near the target if its embedding is similar to the target embedding«

Jurafsky/Martin (2023, 18 f.)

When are vectors similar?

- Operation that takes two vectors and returns a similarity score
- Linear algebra: dot product
  - A.k.a. scalar product, inner product, Skalarprodukt, Punktprodukt, inneres Produkt

$$\begin{aligned}\vec{a} \cdot \vec{b} &= |\vec{a}||\vec{b}| \cos \sphericalangle(\vec{a}, \vec{b}) \\ &= \sum_{i=1}^{N} \underline{a_i b_i}\end{aligned}$$

$[0, 1, 3] \cdot$

$[5, 4, 7]$

$= 0 \cdot 5 + 4 \cdot 1 + 3 \cdot 7$

$= 25$

Skip-gram

Notation
t, c: words
$\vec{t}$, $\vec{c}$: vectors for the words

$$p(+|t,c) = \sigma(\vec{t}\cdot\vec{c}) = \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}}$$

$$p(-|t,c) = 1-\sigma(\vec{t}\cdot\vec{c}) = 1-\frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}}$$

Skip-gram

Notation
t, c: words
$\vec{t}$, $\vec{c}$: vectors for the words

$$
\begin{aligned}
p(+|t,c) &= \sigma(\vec{t}\cdot\vec{c}) = \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} \\
p(-|t,c) &= 1-\sigma(\vec{t}\cdot\vec{c}) = 1-\frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}}
\end{aligned}
$$

but the context consists of more than one word!

Skip-gram

Notation

t, c: words

$\vec{t}$, $\vec{c}$: vectors for the words

$$
\begin{aligned}
p(+|t,c) &= \sigma(\vec{t}\cdot\vec{c}) = \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} \\
p(-|t,c) &= 1-\sigma(\vec{t}\cdot\vec{c}) = 1-\frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}}
\end{aligned}
$$

but the context consists of more than one word!
Assumption: They are independent, allowing multiplication

Skip-gram

Notation
t, c: words
$\vec{t}$, $\vec{c}$: vectors for the words

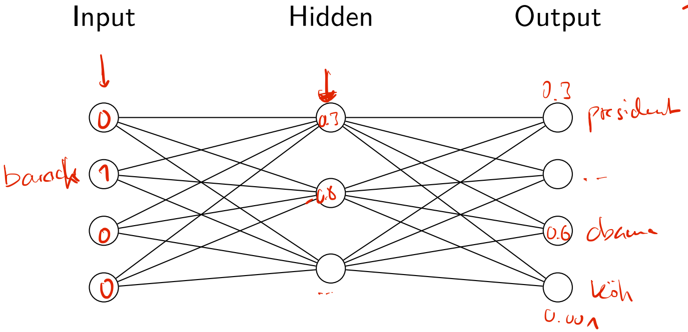
$$
\begin{aligned}
p(+|t,c) &= \sigma(\vec{t}\cdot\vec{c}) = \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} \\
p(-|t,c) &= 1-\sigma(\vec{t}\cdot\vec{c}) = 1-\frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}}
\end{aligned}
$$

but the context consists of more than one word!
Assumption: They are independent, allowing multiplication

$$
\begin{aligned}
p(+|t,c_{1:k}) &= \prod_{i=1}^{k}\frac{1}{1+e^{-\vec{t}\cdot\vec{c}_i}} \\
\log p(+|t,c_{1:k}) &= \sum_{i=1}^{k}\log\frac{1}{1+e^{-\vec{t}\cdot\vec{c}_i}}
\end{aligned}
$$

$$\begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix} \cdot [0.7 \quad 0.8 \quad -0.5] \Rightarrow$$

Neural Network Layout

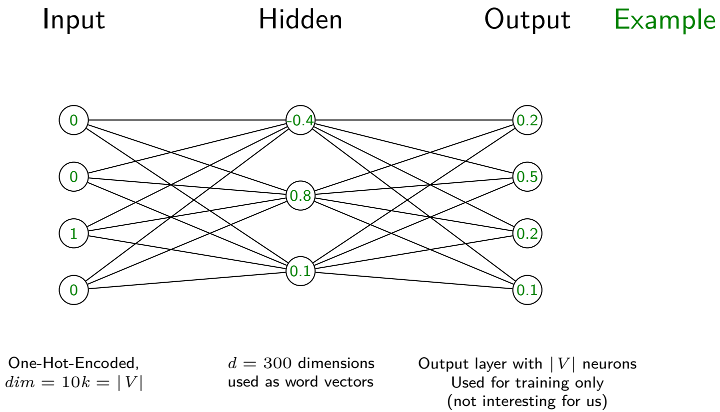


One-Hot-Encoded,
$dim = 10k = |V|$

$d = 300$ dimensions
used as word vectors

Output layer with $|V|$ neurons
Used for training only
(not interesting for us)

Neural Network Layout



Negative Sampling

- Negative examples
  - Training a classifier needs negative examples, i.e., words that are not in the context of each other

Negative Sampling

- Negative examples
  - Training a classifier needs negative examples, i.e., words that are not in the context of each other
- Negative sampling
  - For every positive tuple (t, c), we add k negative tuples
  - Negative tuple (t, c_n), with c_n randomly selected (and $t \neq c_n$)

Negative Sampling

- Negative examples
  - Training a classifier needs negative examples, i.e., words that are not in the context of each other
- Negative sampling
  - For every positive tuple (t, c), we add k negative tuples
  - Negative tuple (t, c_n), with c_n randomly selected (and $t \neq c_n$)
- New 'parameter' k on this slide
  - Different status than θ (the parameters we want to learn)
  - Therefore: Hyperparameters

Loss Function

- We also need a loss function
- Idea:
  - Maximize
    - $p(+|t, c)$ for positive samples (i.e., words that are in context of each other)
    - $p(-|t, c_n)$ for negative samples (i.e., words that are not in context of each other)

Loss Function

- We also need a loss function
- Idea:
  - Maximize
    - $p(+|t, c)$ for positive samples (i.e., words that are in context of each other)
    - $p(-|t, c_n)$ for negative samples (i.e., words that are not in context of each other)

$$L(\theta) = \sum_{(t,c)} \log p(+|t, c) + \sum_{(t,c_n)} \log p(-|t, c_n)$$

Loss Function

- We also need a loss function
- Idea:
  - Maximize
    - $p(+|t, c)$ for positive samples (i.e., words that are in context of each other)
    - $p(-|t, c_n)$ for negative samples (i.e., words that are not in context of each other)

$$L(\theta) = \sum_{(t,c)} \log p(+|t, c) + \sum_{(t,c_n)} \log p(-|t, c_n)$$

θ: Concatenation of all $\vec{t}$, $\vec{c}$, $\vec{c}_n$

Section 3

Summary

Summary

- Neural networks
  - Layered architecture
  - Output of one layer fed into the next
  - Layer contains neurons, a neuron represents a single calculation
  - Activation functions
- Word2Vec training
  - Two architectures
  - Train NN to predict words in contexts
  - Use learned weights as word vectors
  - From Scratch Guide

References I

- Jurafsky, Dan/James H. Martin (2023). *Speech and Language Processing*. 3rd ed. Draft of Janaury 7, 2023. Prentice Hall. URL: https://web.stanford.edu/~jurafsky/slp3/.
- Mikolov, T./K. Chen/G. Corrado/J. Dean (2013). »Efficient Estimation of Word Representations in Vector Space«. In: *ArXiv e-prints*.
- Mikolov, Tomas/Ilya Sutskever/Kai Chen/Greg S Corrado/Jeff Dean (2013). »Distributed Representations of Words and Phrases and their Compositionality«. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges/L. Bottou/M. Welling/Z. Ghahramani/K. Q. Weinberger. Curran Associates, Inc., pp. 3111–3119.