Deep Learning for Natural Language Processing

Lecture 1 – Kick-off

Dr. Ivan Habernal

April 12, 2021

Trustworthy Human Language Technologies
Department of Computer Science
Technical University of Darmstadt

TrustHLT 1001100110101

www.trusthlt.org

Table of contents

1. Administrative course issues

2. The TrustHLT Group

3. Deep Learning for Natural Language Processing

4. Perceptron

5. Neural Network Principles

6. Activation functions

Administrative course issues

Learning Goals

After completing this course, you are able to

- explain the basic concepts of **neural networks and deep learning**,
- explain the concept of **word embeddings**, train word embeddings and use them for solving NLP problems,
- understand and describe neural network architectures that are used to tackle classical **NLP problems** such as text/sentence classification and sequence tagging,
- **implement** neural networks for NLP problems using existing libraries in Python

General Information

Teaching material

Lectures, exercises/submissions etc. can be found in Moodle

Resources

- This lecture is mainly based on recent papers from top NLP conferences
    - Deep Learning is still evolving quickly
    - Knowledge gets outdated

Useful Additional Resources

Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep Learning, MIT Press (**freely available** online book!)

Stanford Lecture by Richard Socher: *Deep Learning for Natural Language Processing cs224d* (look it up on YouTube)

Yoav Goldberg: Neural Network Methods for Natural Language Processing (not free)

Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong: Mathematics for Machine Learning (**freely available** PDF!)

Jeremy Kun: A Programmer's Introduction to Mathematics (PDF ”**Pay what you want**”)

Recommended Readings

We provide literature for each topic of our course:

- Even if you have very limited time, please do read the **mandatory part** – we will assume that you know these works.
- The mandatory papers will be “prüfungsrelevant” (relevant for the exam).
- We will ask one/two questions from the mandatory papers in the homeworks.

We **encourage** you to read the optional parts as well (listed in each lecture).

If you cannot find/access a publication → feel free to ask.

Exercises and Homeworks (I)

Exercises (EX)

- Deepening and testing the understanding from the lecture
- Not graded

Homeworks (HW)

- Practically applying your knowledge from the lecture (and exercises)
- Mostly programming + a bit of math and paper reading
- Submit in groups of two
- First homework will be published today

EX and HW organized by Jonas Rikowski

Exercises and Homeworks (II)

Questions

- Moodle forums, Discord server
- Only for personal questions: Private Moodle message to the tutors

To do after this lecture

1. Read the exercise and homework **logistics** (PDF) on Moodle thoroughly
2. Select your homework **group** on Moodle (deadline: April 22)

Shared Task

The shared task is organized by **Philipp Marquardt**

You will work on **Conversational AI**

You will work in groups of up to 3 people

You will develop a solution to the problem and then compete with other groups

Shared task is graded and gives up to 100 points. Grading is based on:

- Written report about your system (60 points)

- Your ranking compared to other teams -> better system, more points (30 points)

- A short presentation in the last class (10 points)

Bonus System

By submitting the homeworks and participating in the shared task, you can get a bonus of 0.3 (or 0.4) for the exam.

You need to reach $\geq$ **70%**[1] of the summed homework and shared task points.

More details: → exercise and homework **logistics** on Moodle.

[1] subject to changes

Tutors

Jonas Rikowski

Philipp Marquardt

If you have personal questions, contact them via moodle

Final Exam

Save the date:

To be announced on Moodle

Register via TUCaN, see also:

`https://www.informatik.tu-darmstadt.de/de/studierende/studienbuero/pruefungsan-und-abmeldung/`

Exam questions will be in English. Answers may be in German or English.

Final Exam: Resources Allowed

Scope: Lecture, mandatory readings, and practice class

Final grade: Exam (100%) + Bonus System

You may bring a dictionary to the exam, if neither German nor English is your first language.

You may bring a non-programmable calculator to the exam.

No other resources (books, lecture notes etc.) are allowed during the exam.

Questions/Suggestions

Any questions, suggestions?

Post a message in the Moodle forum or chat on Discord

If that doesn‘t help: Write an E-Mail to the tutors or make an appointment with them

If that doesn‘t help also: Write an email to me

The TrustHLT Group

Trustworthy Human Language Technologies

TrustHLT 1001100110101

www.trusthlt.org

- Privacy-preserving NLP (differential privacy; deep learning; representation learning; graph networks)

- Argument mining ”that matters” (legal argument mining; ethical argumentation)

Master thesis? Get in touch!

Deep Learning for Natural Language Processing

Deep Learning for Natural Language Processing

Deep Learning = neural networks, sub-field of machine learning

NLP = Analysis of language

History of Deep Learning 1

Early booming (1950s – early 1960s)

- Rosenblatt (1958) – Perceptron
- Widrow and Hoff (1960, 1962) – Learning rule based on gradient descent

Setback (mid 1960s late 1970s)

- Serious problems with perceptron model (Minsky's book 1969)
- Can't even represent simple functions

Renewed enthusiasm (1980s)

- New techniques (Backpropagation for "deep nets")

History of Deep Learning 2

Out-of-fashion – again (1990s to mid 2000s)

- Other techniques were considered superior and more understandable; Support Vector Machines, Integer linear programming, etc.
- So much out of fashion, some good CS journals immediately rejected papers on neural networks without review (as reported by Geoffrey Hinton)
- Played no role in top NLP conferences

Since mid 2000: huge progress for “deep learning”

- Hinton and Salakhutdinov (2006): one can actually train deep nets
- Since 2013: Word embeddings, work of Mikolov et al.

History of Deep Learning 3

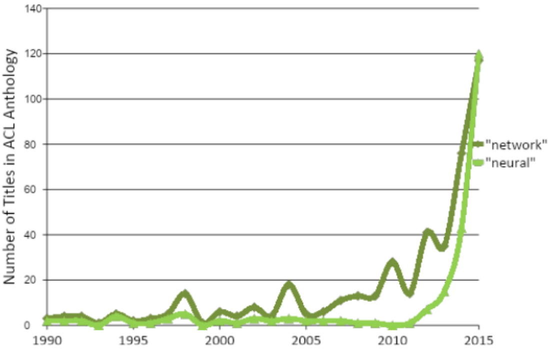
Since mid 2010: huge progress for “deep learning”

Why?

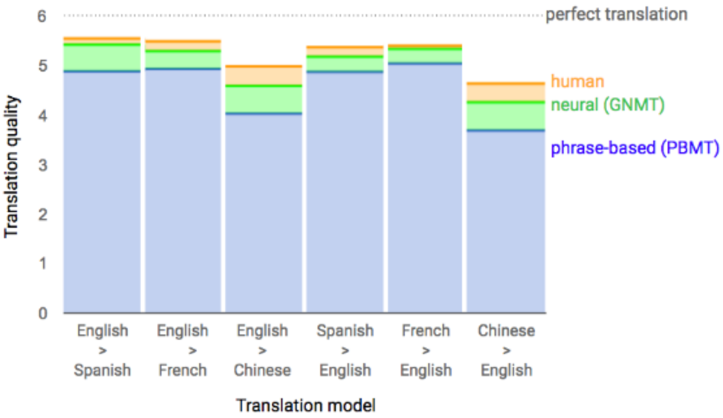
- More data
- Faster computers, better hardware (GPU)
- Unsupervised pre-training
- Better optimization techniques, better understanding of the approaches, …

Deep learning in NLP

Occurrences of “neural” and “network” in titles in the ACL Anthology



DL Example: Machine translation

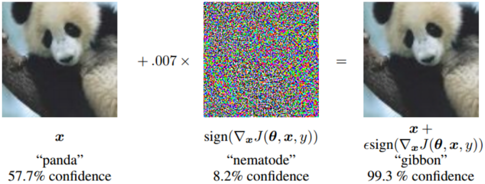


Not everything works!

Depending on characteristics of your task, traditional approaches may still be preferable or superior

- Neural nets seem to particularly excel on “difficult” tasks, less on simple tasks (task difficulty)
- Other factors such as training data size may play a crucial role

Neural Networks are also prone to fooling



$+ .007 \times$
$=$
$\boldsymbol{x}$
“panda”
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“nematode”
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”
99.3 % confidence

Examples of low-level NLP problems/tasks

Sequence tagging

- POS-tagging (part of speech)

Example

Time flies like an arrow.

Fruit flies like a banana.

Example

Time$_{\mathrm{NN}}$ flies$_{\mathrm{VBZ}}$ like$_{\mathrm{IN}}$ an$_{\mathrm{DT}}$ arrow$_{\mathrm{NN}}$.

Fruit$_{\mathrm{NN}}$ flies$_{\mathrm{NN}}$ like$_{\mathrm{VB}}$ a$_{\mathrm{DT}}$ banana$_{\mathrm{NN}}$.

IN = Preposition or subordinating conjunction (conjunction here); VBZ = Verb, 3rd person singular present; DT = determiner; NN = singular noun

Low-level NLP problems/tasks

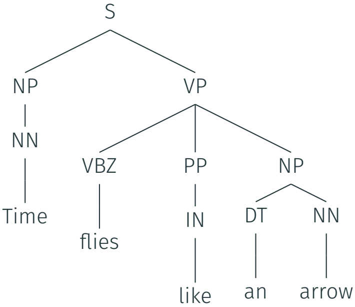
Sequence tagging

- POS-tagging (part of speech)

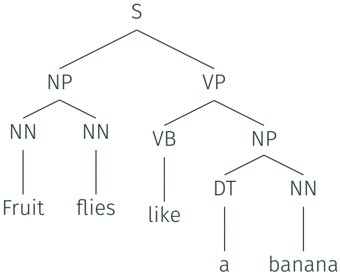
Structure prediction

- Chunking/Parsing

Example



Example



Examples of low-level NLP problems/tasks

Sequence tagging

- POS-tagging (part of speech)

Structure prediction

- Chunking/Parsing

Semantics

- Word sense disambiguation

Example

Time flies like an arrow.

Fruit flies like a banana.

High-level NLP tasks

- Information Extraction
  - search, event detection, textual entailment
- Writing Assistance
  - spell checking, grammar checking, auto-completion
- Text Classification
  - spam, sentiment, author, plagiarism
- Natural language understanding
  - metaphor analysis, argumentation mining, question-answering
- Natural language generation
  - summarization, tutoring systems, chat bots
- Multilinguality
  - machine translation, cross-lingual information retrieval

Example: Named Entity Recognition (NER) 1

Example

”… chancelor$_{\texttt{O}}$ Angela$_{\texttt{B-PER}}$ Merkel$_{\texttt{I-PER}}$ said$_{\texttt{O}}$ …”

”BIO”-tagging

- `B` = Begin of entity, e.g., `B-PER` (person), `B-LOC` (location)
- `I` = ”Inside” entity, e.g, `I-PER`, `I-LOC`
- `O` = Other (no entity)

Traditional feature engineering

Example – determining tag for *Angela*

”... chancelor$_{\texttt{O}}$ Angela$_{\texttt{B-PER}}$ Merkel$_{\texttt{I-PER}}$ said$_{\texttt{O}}$ ...”

Each token is represented by a feature vector

Feature type	Value
uppercase	1
is Noun	1
previousWord DET	0
previousWord uppercased	0
following word N	1
following word uppercased	1
...	...

Traditional feature engineering (cont.)

Example – determining tag for *Merkel*

”… chancelor$_{\text{O}}$ Angela$_{\text{B-PER}}$ Merkel$_{\text{I-PER}}$ said$_{\text{O}}$ …”

Each token is represented by a feature vector

Feature type	Value
uppercase	1
is Noun	1
previousWord DET	0
previousWord uppercased	1
following word N	0
following word uppercased	0
…	…

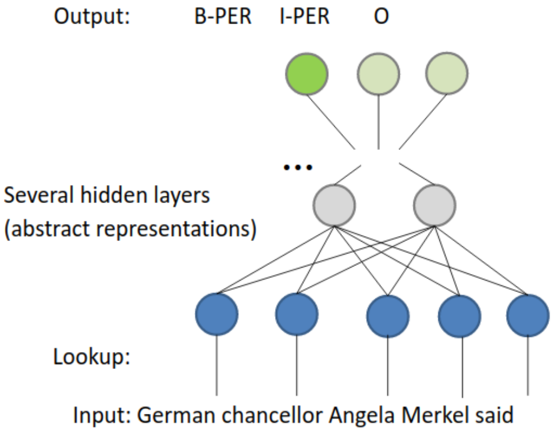
Traditional feature engineering (cont.)

Word specific features

- previous word is minister
- previous word is chancellor
- previous word is president
- previous word is company
- previous word is product
- following word is says
- following word is declares
- following word is claims

Problem: Similar words are represented as distinct features.

DL for NER


Output:
B-PER
I-PER
O
...
Several hidden layers
(abstract representations)
Lookup:
Input: German chancellor Angela Merkel said

Architecture of neural networks: Lecture 3/8-

Mapping words into vector representations: Lecture 4-7

Perceptron

Origins

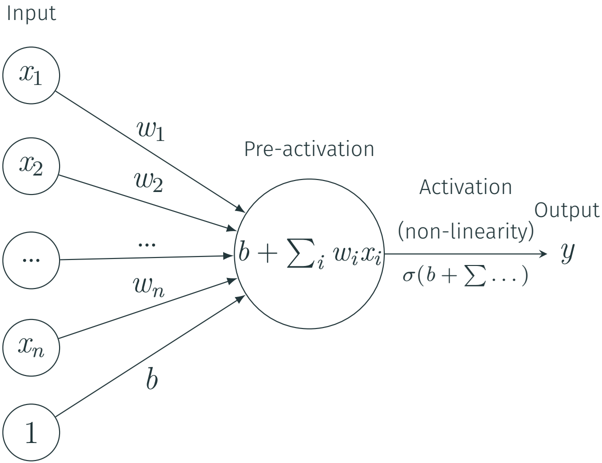
‘Simplest’ form of a neural network

Perceptron is an algorithm for learning a binary classifier

Introduced by McCulloch and Pitts (1943) and Frank Rosenblatt (1958)

Figure 1: Frank Rosenblatt; Source: Wikipedia

Perceptron in detail



Formal description

Given

- weight vector and bias $\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$

- non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$

Input to network

- Input vector $\mathbf{x} \in \mathbb{R}^n$

- and constant 1

Output unit

- Input: $\underbrace{\mathbf{x} \cdot \mathbf{w}}_{\text{Dot product}} + b = \sum_{i=1}^{n} (w_i x_i) + b$

- Output: $\sigma(\mathbf{x} \cdot \mathbf{w} + b)$

Simplification

Given

- weight vector (incl. bias term) $\mathbf{w} \in \mathbb{R}^{(n+1)}$

- non-linear function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$

Input to network

- Input vector $\tilde{\mathbf{x}} \in \mathbb{R}^{(n+1)} = (\mathbf{x} \quad 1)$

Output unit

- Input: $\tilde{\mathbf{x}} \cdot \mathbf{w} = \sum_{i=1}^{n} w_i x_i$

- Output: $\sigma(\tilde{\mathbf{x}} \cdot \mathbf{w})$

Optimization problem

Perceptron is the function $f : \mathbb{R}^{n+1} \to \mathbb{R}$ where $f(\mathbf{x}; \mathbf{w}) = \sigma(\tilde{\mathbf{x}} \cdot \mathbf{w})$

Vector $\mathbf{w}$ are the parameters we want to ”learn”

Training data is a set of N examples

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$$

We want to minimize ”the difference” between the true y-s and what our network outputs

$$\min_{\mathbf{w}} \sum_{j=1}^{N} (f(\mathbf{x}; \mathbf{w}) - y_j)^2$$

How do we minimize the loss function?

$$\min_{\mathbf{w}} \sum_{j=1}^{N} (f(\mathbf{x}; \mathbf{w}) - y_j)^2$$

We will employ a general optimization technique called *(stochastic) gradient descend*; more in Lectures 2 & 3

Perceptron for binary classification and decision surface

Perceptron uses a simple threshold function for decisions

$$\sigma(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Example 2-D input; 2 neurons (= features) and bias; $(\mathbf{x} \in \mathbb{R}^2)$

the decision boundary is a straight line $y = az + b$

Linear separation

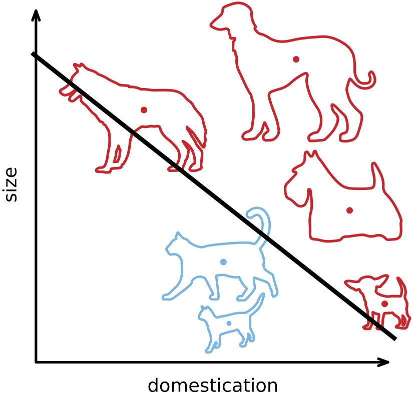


Figure 2: Linear boundary of perceptron[2]

[2] https://en.wikipedia.org/wiki/Perceptron

What a perceptron can do

Linear separability

Let X_0 and X_1 be two sets of points in an n-dimensional Euclidean space. Then X_0 and X_1 are *linearly separable* if there exist $n+1$ real numbers $w_1, \ldots, w_{n+1}$ such that every point $\mathbf{x} \in X_0$ satisfies $\tilde{\mathbf{x}} \cdot \mathbf{w} > 0$ and every point $\mathbf{x} \in X_1$ satisfies $\tilde{\mathbf{x}} \cdot \mathbf{w} \leq 0$

What a perceptron cannot do

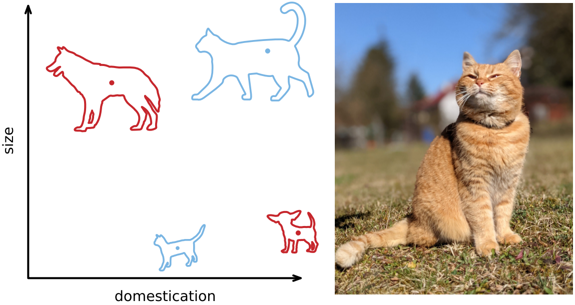
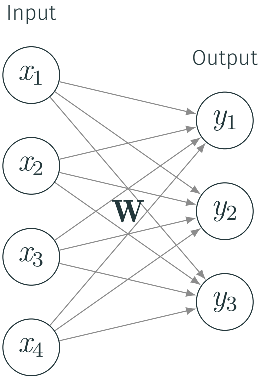


Figure 3: Examples of XOR (exclusive or)[3]

[3] https://en.wikipedia.org/wiki/Perceptron

Neural Network Principles

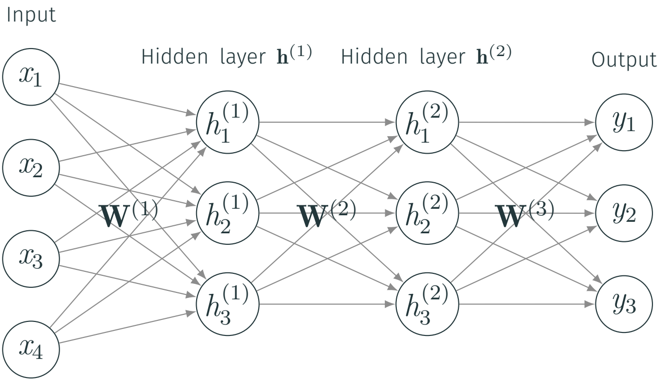
Multiple output nodes



- Previously for 1-D output: a dot product $\mathbf{x} \cdot \mathbf{w}$
- Now matrix multiplication $\mathbf{x} \times \mathbf{W}$
- Weights are in a matrix $\mathbf{W}$
- $\mathbf{x}$ is a row vector
- Each column in $\mathbf{W}$ corresponds to weights of j-th output y_j

Hidden layers

Multiple hidden units/hidden layer(s)



This architecture is known as Multi-Layer Perceptron (MLP).

Computations in detail

$$\mathbf{h}^{(1)} = \sigma(\mathbf{x}\mathbf{W}^{(1)} + b)$$

where $\sigma()$ is applied element-wise, for example

$\mathbf{v} = [v_1, v_2, v_3]$, $\sigma(\mathbf{v}) = [\sigma(v_1), \sigma(v_2), \sigma(v_3)]$

2-nd layer computes

$$\mathbf{h}^{(2)} = \tau(\mathbf{h}^{(1)}\mathbf{W}^{(2)} + c)$$

(where τ is non-linear function which may or might not be the same as σ; $c \in \mathbf{R}$ is bias term)

High-level view of neural networks

Neural networks are mathematical function of the form (ignoring bias terms)

$$\mathrm{NN}(\mathbf{x}; \mathbf{W}, \mathbf{V}) = g(f(\mathbf{x} \cdot \mathbf{W}) \cdot \mathbf{V})$$

where $\mathbf{x}$ is the given input, non-linear functions g and f are also given, but $\mathbf{W}$ and $\mathbf{V}$ are parameters we want to learn

Typically we put all parameters into the theta variable θ, so $\mathrm{NN}(\mathbf{x}; \mathbf{W}, \mathbf{V})$ becomes $\mathrm{NN}(\mathbf{x}; \theta)$

Loss function

We define a **loss function** in the form

$$L(\theta) = \sum_{(\mathbf{x},\mathbf{y})} \left(\mathrm{NN}(\mathbf{x};\theta) - \mathbf{y}\right)^2$$

- $(\mathbf{x}, \mathbf{y})$ is our training data

- θ are the parameters

By minimizing the loss, we optimize our parameters – *training* (learning)

Activation functions

Activation functions σ

Linear activation $\sigma(x) = ax + b$

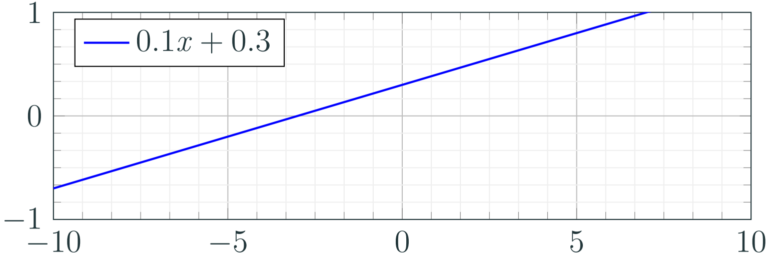


Figure 4: Linear function example

Suppose all neurons have linear activation functions.
What is your neural network computing?

Activation functions σ

Sigmoid (logistic) activation $\sigma(x) = \frac{1}{1+\exp(-ax)}$

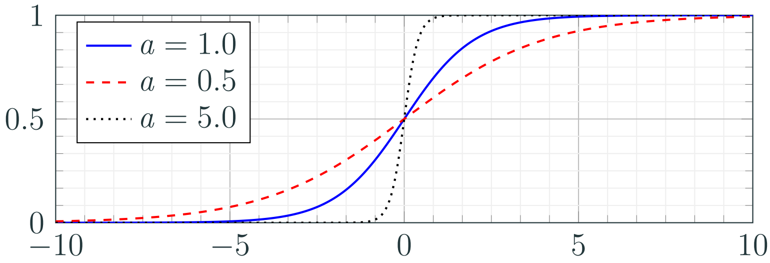


Figure 5: Sigmoid function examples

Output not centered around 0; gradient problematic for large values

Activation functions σ

$\tanh$ activation

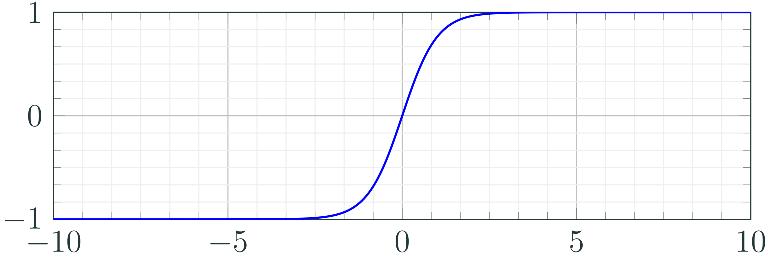


Figure 6: $\tanh$ function

Centered around 0; gradient problematic for large values

Activation functions σ

Rectified linear unit (ReLU) activation

$$\mathrm{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

or $\mathrm{ReLU}(z) = \max(0, x)$

- Not centered around 0; zero gradient problematic for negative values

- Gradient not saturate for positive values; Computationally efficient; Experimentally: converges faster

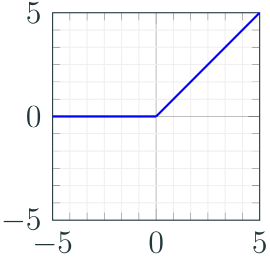


Figure 7: ReLU function

Activation functions σ

Leaky ReLU activation

$$\mathrm{ReLU}(z) = \max(0.01x, x)$$

- Not entered around 0

- Gradient O.K. for entire domain; Computationally efficient; Experimentally: converges fast

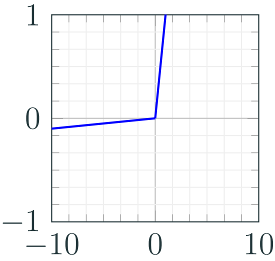
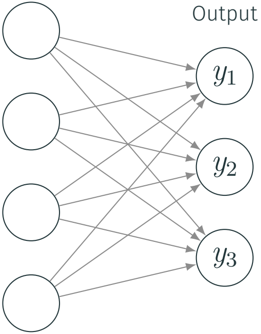


Figure 8: Leaky ReLU function

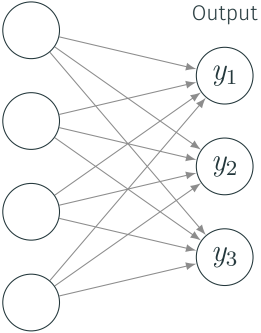
A special activation function: softmax



Classification into multiple output classes (e.g., „Positive“, „Negative“, „Neutral“), we’d often like our outputs to represent a probability distribution over these classes

$$y_1 + y_2 + y_3 = 1; \quad y_j \geq 0$$

A special activation function: softmax



Let z_1 be pre-activation of output node y_1

$$y_1 = \frac{\exp(z_1)}{\exp(z_1) + \exp(z_2) + \exp(z_3)}$$

For m output nodes, **softmax** for each output node y_j is

$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{m} \exp(z_i)}$$

A special activation function: softmax

softmax for each output node y_j in a layer is

$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{m} \exp(z_i)}$$

Unlike other activation functions, softmax depends on all units in the layer

- Global view, probabilistic interpretation

- Softmax name is misleading: It is not a smooth maximum but rather smooth approximation of `arg max` function (for soft maximum see `LogSumExp`)

Wrap up

We started with organizational stuff

Then looked at the history of deep learning

.. and the history of NLP, as well as of problems in NLP (these are also historically changing!)

Afterwards, we presented the perceptron

Derived a learning algorithm

Finally, we looked at some basics of deep learning

References i