



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES



inovex

AI Labor - Sommersemester 2020

Natural **L**anguage **P**rocessing
Einführung

Pascal Fecht, Maximilian Blanck

Karlsruhe, 24. April 2020



Maximilian Blanck

Data Scientist

seit 2018



Pascal Fecht

Software Developer

seit 2016



WHAT IS NATURAL LANGUAGE PROCESSING?



Natural Language Processing

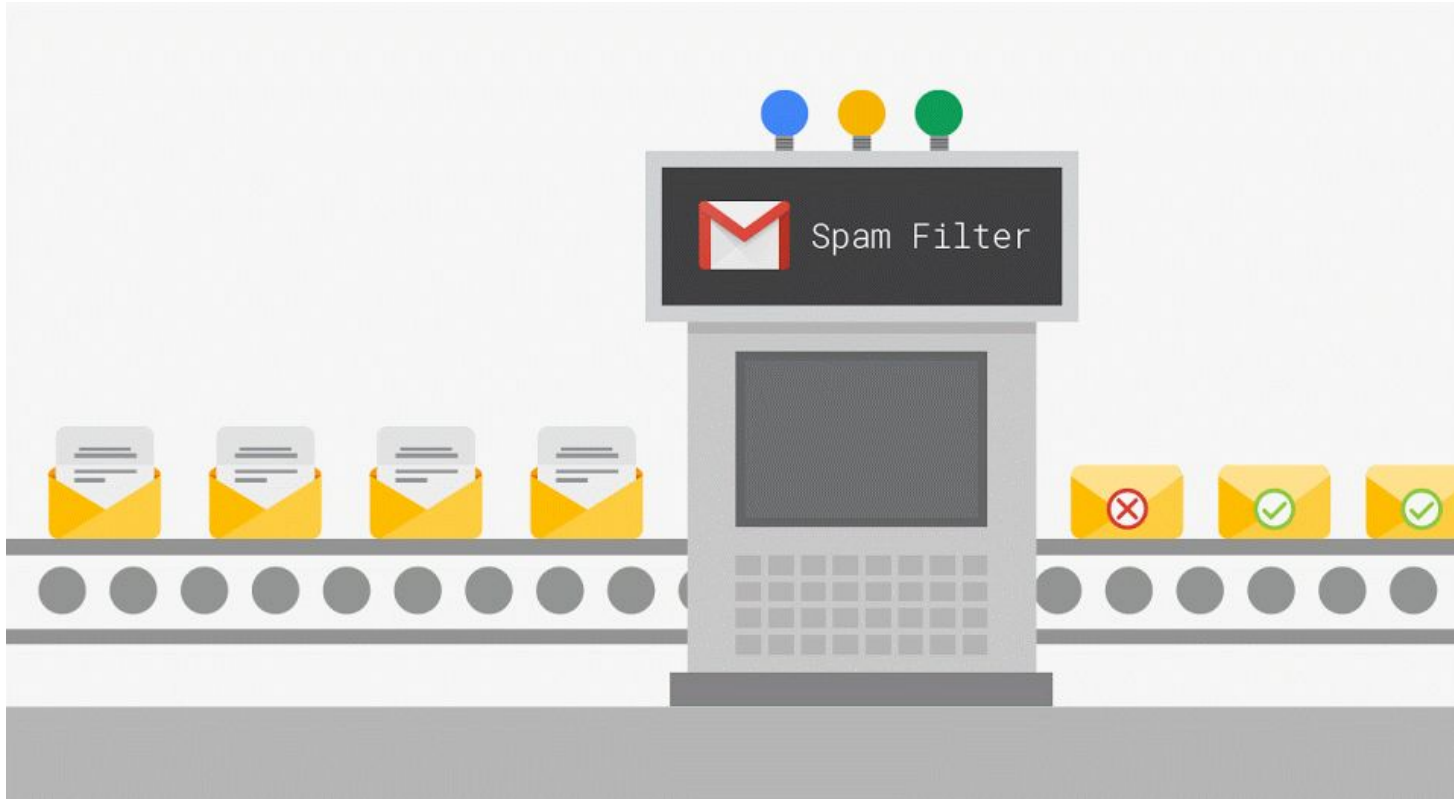
› **Theorie**

- Natural Language Processing
- Vorverarbeitung
- Word Embeddings (word2vec)
- Text Klassifikation → Sentiment Analyse

› **Praxis**

- Word Embeddings in Corona-Domäne (Aufgabe 1)
- Sentiment Analyse für Twitter Posts (Aufgabe 2)

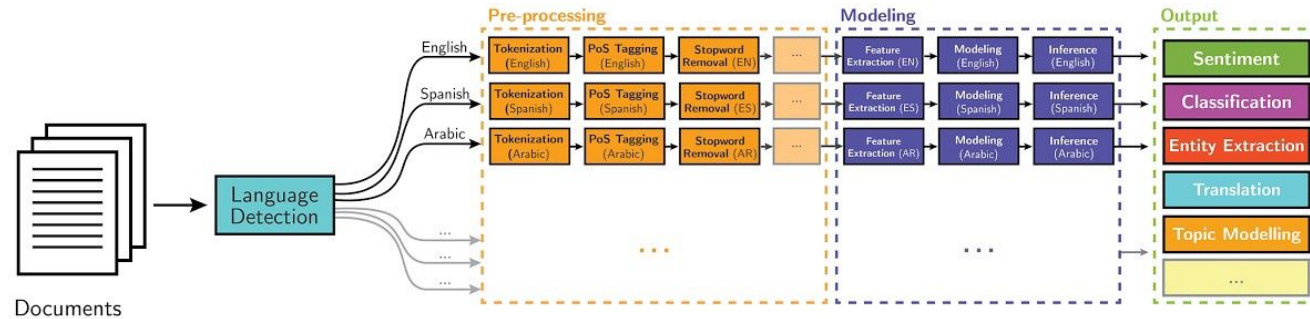
Natural Language Processing



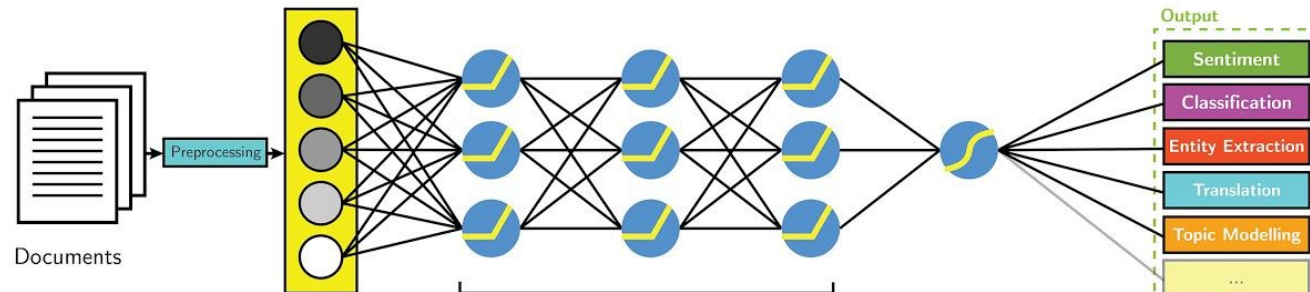
Natural Language Processing...

- › **beinhaltet** Linguistik, Natürliche Sprache, Künstliche Intelligenz, Mathematik, ...
- › **befasst** sich mit der Interaktion zwischen Computern und Menschen
- › **beschäftigt** sich unter anderem mit:
text classification, named entity recognition, machine translation, part of speech tagging, sentiment analysis, question answering, text summarization, text generation, speech recognition, speech to text, text to speech, chatbots, virtual assistants, ...

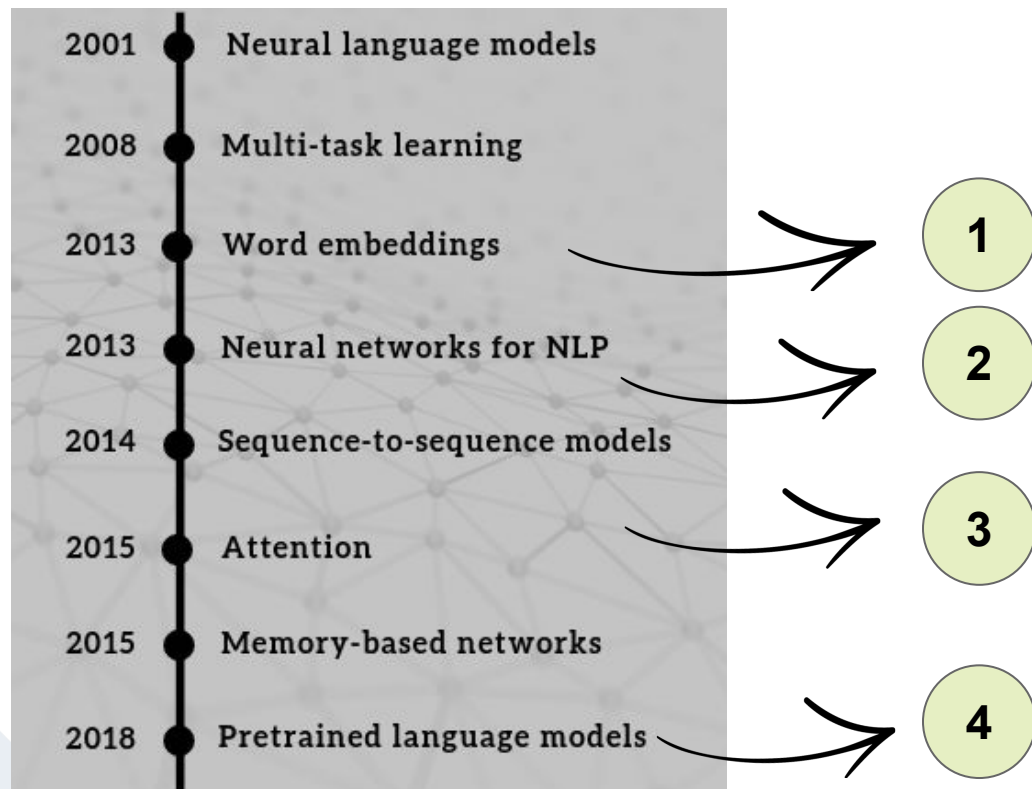
Klassische vs Deep Learning NLP Pipeline



Deep Learning-based NLP



Die letzten 10 Jahre in NLP



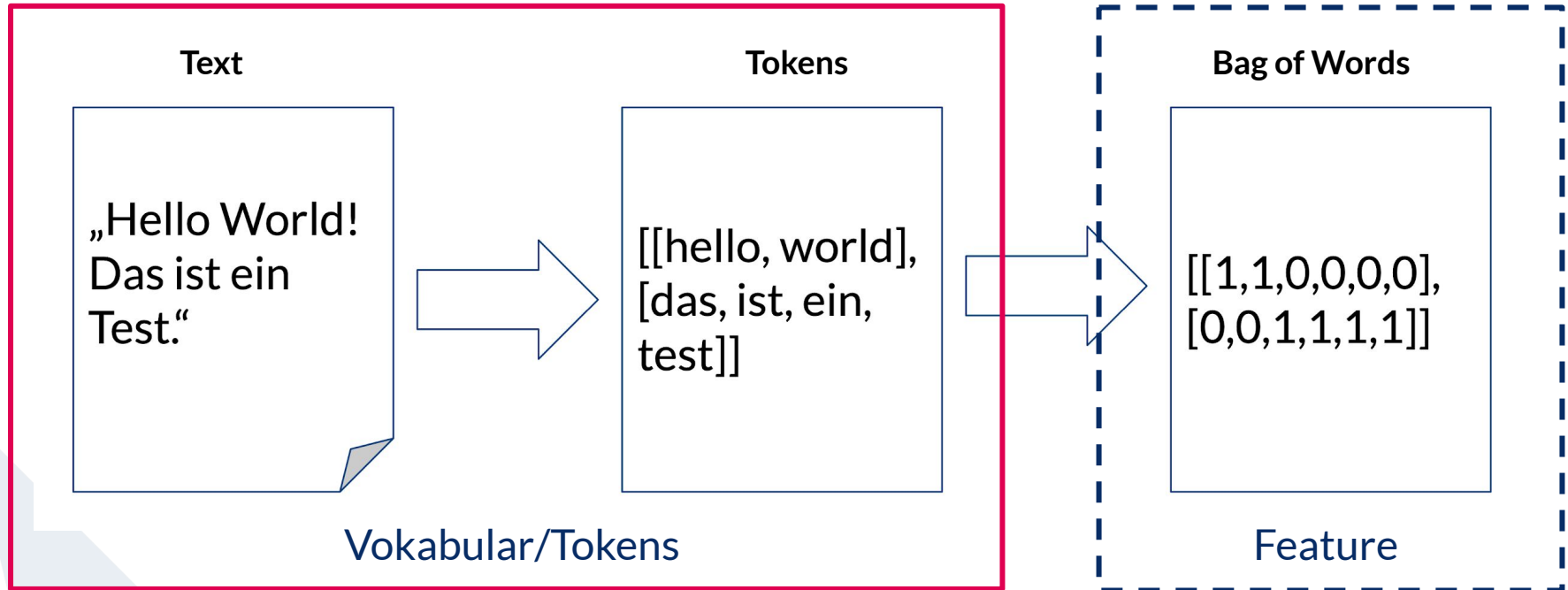
Herausforderungen in NLP

1. Sprache muss irgendwie in den Computer kommen.
 - › Wort / Laut => Zahl(en)
2. Das Vokabular einer Sprache ist in der Regel sehr umfangreich
 - › Dimensionalitätsproblem (?)
3. Einige Wörter sind enger verbunden als andere
 - › Ähnlichkeit, Abstand => Maße?
4. Machine Learning

Wie kommt Sprache in Computer?

Wie kommt die Sprache in den Computer?

Klassischer Ansatz: **Text** → **Tokens** → **Bag of Words**



Vorverarbeitung Vokabular & Features

Vorverarbeitung: Überblick

- › **Vokabular:** Tokenization/Normalization:
 - Erkennung von Satz-/Wortgrenzen
 - Einschränkung des Vokabulars
 - Stop Word Removal
 - Lower Casing
- › Umwandlung von Text in **Features:**
 - Part of Speech Tagging
 - Bag Of Words, **Embeddings, Context Embeddings**
 - Andere Features (Rechtschreibung, numerische Werte, Sprache, etc.)

Vokabular

Tokenisierung

Splitte Texte in kleinere Teile, Vor allem:

- “Wörter” → einfacher Ansatz: Trenne an “ ” (Leerzeichen):

```
re.compile(r'\W*\s+').split("Das Auto fährt")
```

 → [“Das”, “Auto”, “fährt”]
- Aber: was ist mit “H-Milch”, “Auto-\nbahn”,
“Dr. Musterdokter”
- Sätze → Trenne an “.” Punkt?

Vokabular

Sind Wörter die richtige Granularität für Tokens?

Subwords

- › Nicht im Vokabular enthaltene Token werden in Sub-Wörter geteilt.

Characters

- › Jedes Char hat ein Encoding
- › Nachteil: Die Bedeutung von Wörtern geht verloren.

Byte-Pair Encodings

⇒ Wir fangen mit Wörtern, kommen aber darauf in Aufgabe 4 zurück.

Vokabular

Der Eingabevektor von NLP Modellen

Grundfrage: Wie viele unterschiedliche Wörter soll mein ML Modell als Eingabe erhalten?

Typische Probleme:

- › Sehr seltene Wörter gehen unter.
- › Häufige Worte, die wenig Inhalt enthalten (“der”, “ein”, ...) können starke Features werden.
- › Wie gehen wir mit Noise um?

Vokabular

Einfaches Preprocessing

- › Restriktion des Vokabular:
 - Vokabluargröße von x Wörtern
 - Ein Wort muss mindestens y mal vorkommen, damit es “*relevant*” ist.
- › Lowercasing

```
1 TOKEN.lower()
```

Vokabular

Einfaches Preprocessing

- › Stopwörter entfernen

```
1 from nltk.corpus import stopwords
2 stop_words = set(stopwords.words('english'))
3 tokens = [w for w in tokens if not w in stop_words]
```

- › Und vieles mehr (Lemmatisierung, ...)



Vokabular/Features

Tagging

Beispiel: Jede **IBAN** ist ein eigenes Token
⇒ Kein relevantes Feature für Modell

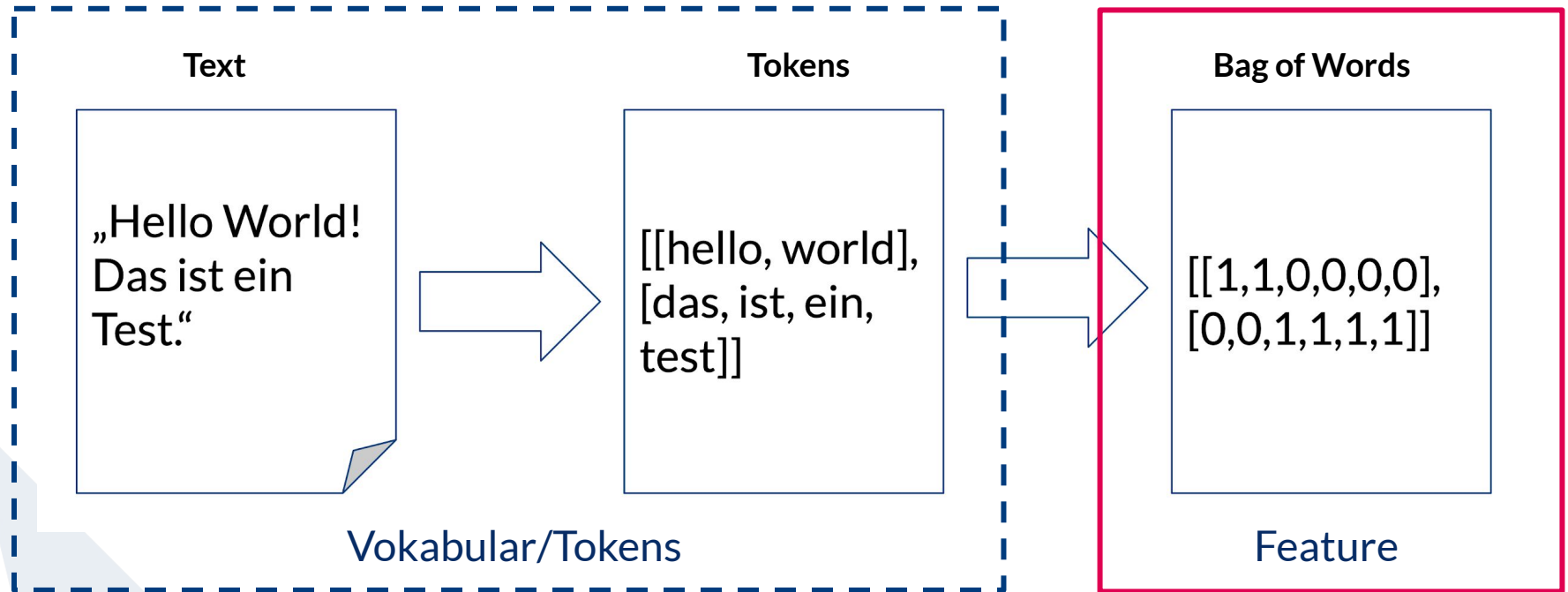
```
1 if re.match('^DE(?:\\s*[0-9a-zA-Z]\\s*){20}$', TOKEN):  
2     return "<IBAN>"
```

⇒ Modell erkennt alle (deutschen) IBANs ein Feature.

Word Embeddings - Die **Bedeutung** von Wörtern

Wie kommt die Sprache in den Computer?

Klassischer Ansatz: **Text** → **Tokens** → **Bag of Words**



Features

Bag of Words

Zähle alle unterschiedlichen Wörter in den Texten, die betrachtet werden:

- › Baue einen Vektor mit Vokabularlänge
- › Jedes Wort hat einen Index im Vektor
- › Merke die Anzahl für jedes Wort und Speichere die Zahl im Vektor
- › Feature für einen Text:
 - $[0,0,2,1,0,0,\dots,0,0,1,3,0,0]$

→ **Problem: Der Vektor beinhaltet sehr viele “0”en**

Word2Vec

- › “You shall know a word by the company it keeps” (Firth, J. R. 1957:11)
- › Word2Vec-Paper: <https://arxiv.org/abs/1301.3781>

Idee:

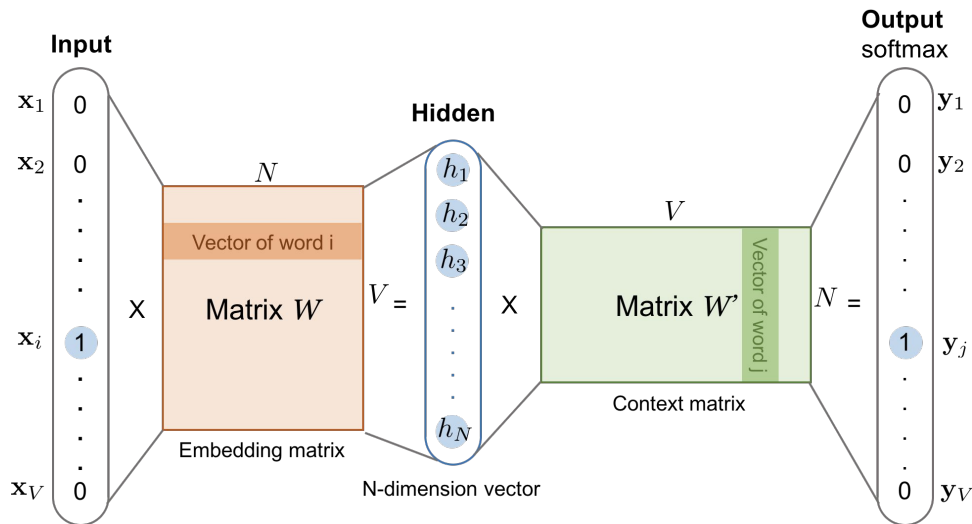
- › Die umliegenden Wörter eines Wortes (in einem beliebigen Text) haben Einfluss auf die Bedeutung dieses Wortes.
- › Beispielsatz: “*Die Ente **quakt** und tanzt.*”

Word2Vec

- › Jedes Wort wird auf einen Vektor der Länge **n** abgebildet:
- › Bsp.
“Auto” → [0.012, 0.981, -0.271,...]
- › **n** ist im Word2Vec-paper 300, andere Werte möglich
- › Wie werden diese Werte erzeugt?
 - 2 Ansätze:
 - Skip-Gram
 - Cbow

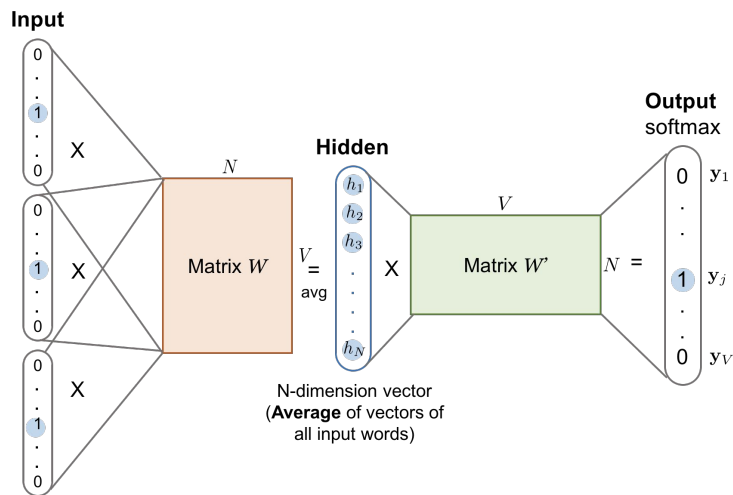
Word2Vec - Skip-Gram (Kontext zum Wort)

- › Gegeben das Wort “**quakt**”, wollen wir den Kontext vorhersagen
 - “Die Ente **quakt** und tanzt.” (Fenstergröße 5)
 - Kontext: [“Die”, “Ente”, “und”, “tanzt”]; Target: [“**quakt**”]
 - Trainingsdaten: $X_1 = [\text{“quakt”}]$, $Y_1 = [\text{“Die”}]$; $X_2 = [\text{“quakt”}]$, $Y_2 = [\text{“Ente”}]$; ...



Word2Vec - Cbow (Wort zum Kontext)

- › Gegeben Kontext “Die”, “Ente”, “und”, “tanzt”, wollen wir “**quakt**” vorhersagen.
 - “Die Ente **quakt** und tanzt.” (Fenstergröße 5)
 - Kontext: [“Die”, “Ente”, “und”, “tanzt”]; Target: [“**quakt**”]
 - Trainingsdaten: $X = [\text{“Die”, “Ente”, “und”, “tanzt”}]$, $Y = [\text{“quakt”}]$



Word Embeddings, Abstände, Gensim

- › Mit Word Embeddings lassen sich nun Abstände berechnen
 - Bsp.: “Ente” und “Gans” haben einen geringeren Abstand als “Ente” und “Auto”, da sie in *Texten* öfters mit den gleichen Kontext Wörtern vorkommen
- › Gensim* ist ein Python package, das das Handling von Wortvektoren vereinfacht.
- › Beispiel:

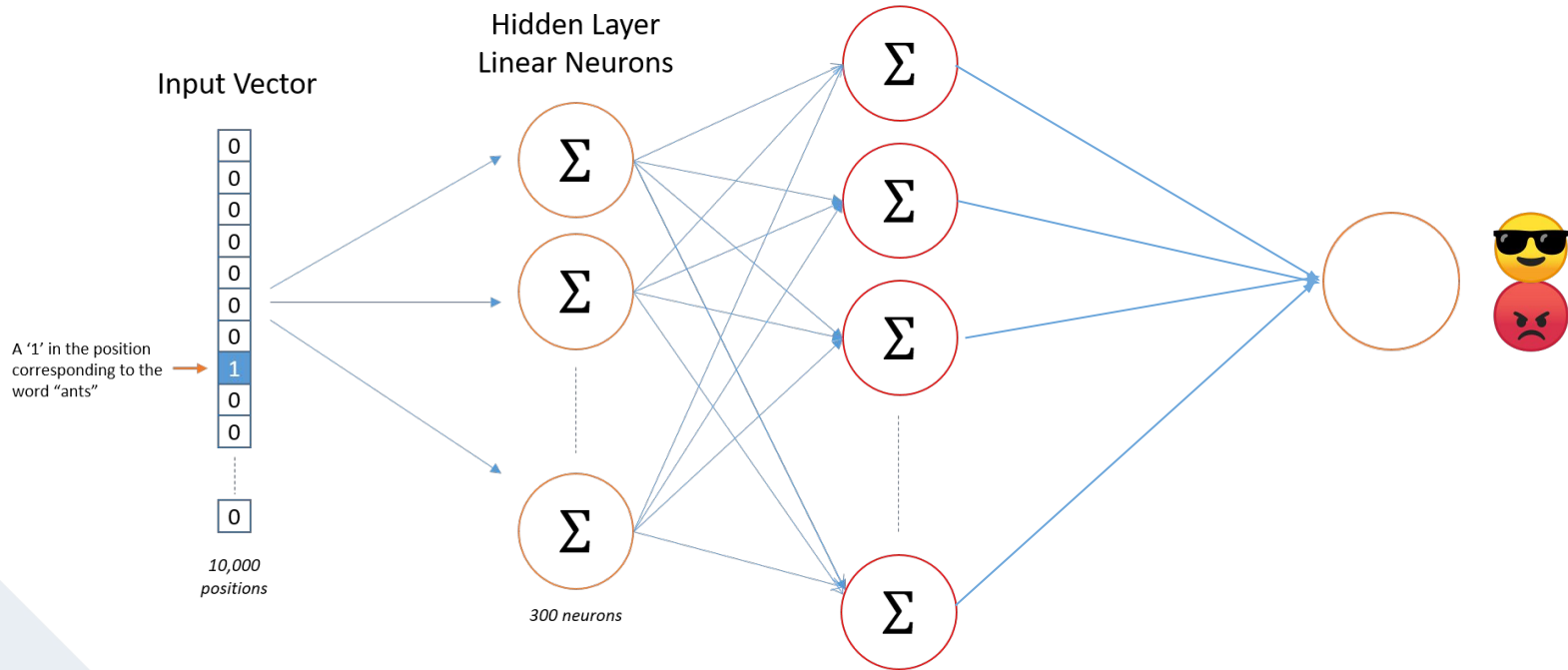
```
from gensim.models import Word2Vec
model = Word2Vec.load("GoogleNews-vectors-negative300.bin")
model.similarity('germany', 'france')
```

Klassifikation

Sentiment Analyse

Sentiment Analysis

Binäre Klassifikation mit BoW



colab

Google Colab

- › In GCP gehosteter Jupyter Notebook Service
- › Kollaborative Arbeit (ähnlich GDoc) möglich
- › Frei nutzbar (inklusive GPU/TPU!)
 - › → Ressourcen sind limitiert
 - › → GPU/TPU Runtime sparsam nutzen!
- › Notebooks und Daten liegen im Google Drive



#TFDevSummit

Making the most of Colab



Feedback



<https://forms.gle/zDjawTGbbs1Z8ryXA>

Vielen Dank

Pascal Fecht

pfecht@inovex.de

Maximilian Blanck

mblanck@inovex.de

Credits an:

Anna Weisshaar

Tilman Berger

