Deep Learning
Teil 3 – Sequence Prediction

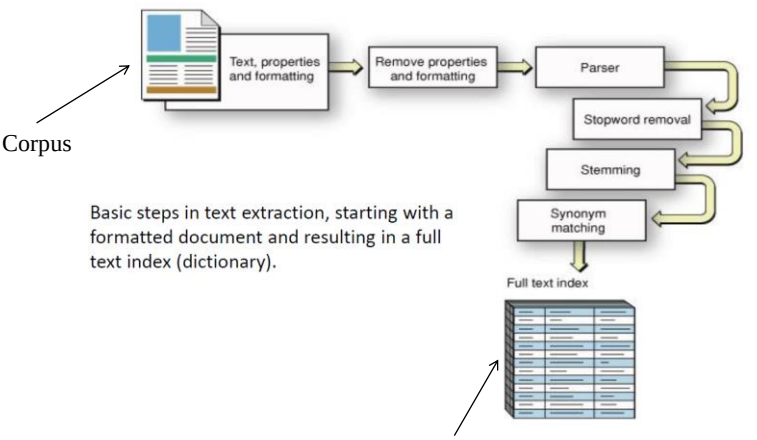
Karin Pröll
DSE

Übersicht

- Einführung
- Sequenzmodelle
- Embedding: word2vec
- Beispielmodell für Sequenzklassifikation
- Beispielmodell für Sequenz-zu-Sequenz

Texte: Wort Sequenzen

Recap Textmining:
Preprocessing Schritte – Basis für die Weiterverarbeitung von Texten



Dictionary / Vocabulary

3

Verarbeitung Text-Sequenzen mit neuronalen Netzwerken (NLP)

- Auf maschinellem Lernen basierende NLP-Systeme arbeiten auf handgefertigten Features, die zeitaufwändig und oft unvollständig sind
- Seit Jahrzehnten wurden Maschine Learning Ansätze im Bereich NLP auf flachen Modellen entwickelt (z. B. SVM und logistische Regression), die auf hoch dimensionalen und spärlichen Features basieren.
- Deep Learning Architekturen und Algorithmen haben beeindruckende Fortschritte in Bereichen wie Computer Vision und Mustererkennung gemacht
- Diesem Trend folgend, konzentriert sich die neueste NLP Forschung immer mehr auf neue Technologien aus dem Bereich Deep Learning.
- In den letzten Jahren haben neurale Netzwerke basierend auf sogenannten „word embeddings“ für Codierung von Worten zu bemerkenswerten Ergebnissen in verschiedensten NLP Aufgabenstellungen geführt.

4

Codierung von Textsequenzen

Vektorraummodelle (VSMs) repräsentieren (eingebettete) Wörter in einem kontinuierlichen Vektorraum, in dem semantisch ähnliche Wörter auf nahegelegene Punkte abgebildet werden ("sind nahe beieinander eingebettet").

VSMs haben eine lange, reiche Geschichte in NLP, aber alle Methoden hängen in irgendeiner Weise von der Verteilungshypothese ab, die besagt, dass Wörter, die in denselben Kontexten auftreten, eine semantische Bedeutung haben.

Die verschiedenen Ansätze, die dieses Prinzip nutzen, können in zwei Kategorien unterteilt werden:

- **zählbasierte Methoden (z. B. latente semantische Analyse):**
  Mit zählbasierten Methoden wird die Statistik berechnet, wie oft ein Wort mit seinen Nachbarwörtern in einem großen Textkorpus zusammen vorkommt. Diese Zählstatistik wird dann für jedes Wort auf einen Vektor abbildet.

- **prädiktive Methoden (z. B. neuronale probabilistische Sprachmodelle):**
  Diese Vorhersagemodelle versuchen direkt, ein Wort abhängig von seinen Nachbarworten vorherzusagen (Kontext).

5

Modelle für Sequenzvorhersage mit neuronalen Netzwerken

Die Sequenzvorhersage unterscheidet sich von anderen Arten von überwachten Lernproblemen: Eine Sequenz hinterlegt bei den Beobachtungen eine Reihenfolge, die beim Trainieren von Modellen und Vorhersagen erhalten bleiben muss.

Sequenzen können Symbole wie Wörter oder Buchstaben in einem Satz oder reale Werte wie jene in einer Zeitreihe sein.

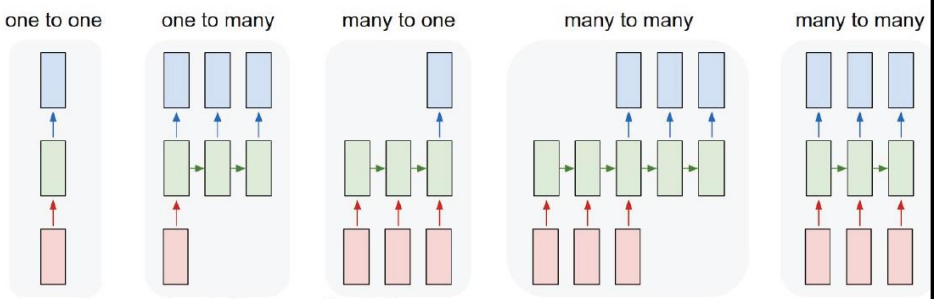
Im Allgemeinen werden Vorhersageprobleme, die Sequenzdaten beinhalten, als Sequenzvorhersage bezeichnet, obwohl es inhaltlich verschiedene Arten von Problemen gibt.

4 verschiedene Arten von Sequenzproblemen:

1. Sequenzvorhersage: Wetter, Börse, Produktempfehlungen
2. Sequenzklassifikation: Sentiment Analyse, DNA Sequenzen
3. Sequenzerzeugung: automatisches Generieren von Texten basierend auf speziellen Korpus - generative Modelle
4. Sequenz-zu-Sequenz-Vorhersage: Übersetzer für Sprachen

6

Modelle für Sequenzvorhersagen mit neuronalen Netzwerken



one to one	one to many	many to one	many to many	many to many
Input: No sequence Output: No sequence Example: "standard" classification / regression problems	Input: No sequence Output: Sequence Example: Im2Caption	Input: Sequence Output: No sequence Example: sentence classification, multiple-choice question answering	Input: Sequence Output: Sequence Example: machine translation, video captioning, open-ended question answering, video question answering	

Codierung von Textsequenzen mit Beibehalten der Reihenfolge der Worte pro Satz

Vocabulary

Index	Wort
1	…..
2	eat
3	I
4	…..
5	…..
6	pizza
7	to
8	want
9	…..

Dimension V
V=9

Beispielsatz: I want to eat pizza

Jedes Wort des Beispielsatzes wird gemäß der Reihenfolge im Satz mit seinem Index ins Vocabulary codiert.

3	8	7	2	6

Alternative Codierung des Beispielsatzes

0	**1**	**1**	0	0	**1**	**1**	**1**	0

V-dim

Beispiel:
Textcodierung mit neuronalem Netz für Sentiment Klassifikation

Standardcodierung Satz:

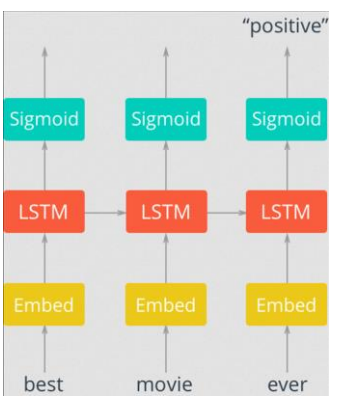
Netzwerk mit unterschiedlichen Layertypen

Recodierung des Inputs erfolgt in einem eigenen Embedding Layer des Netzwerks

Komponenten für Sequenzvorhersagen mit neuronalen Netzwerken

Zwei Hauptbestandteile

- **Word Embeddings**: Abbildung von Worten auf Vektoren reeller Zahlen beliebiger Dimension
- **rekurrentes neuronales Netz** beliebiger Architekturen wie RNN, LSTM



Beispiel Sentiment Analyse Filme

Word Embeddings (automatische Codierung)

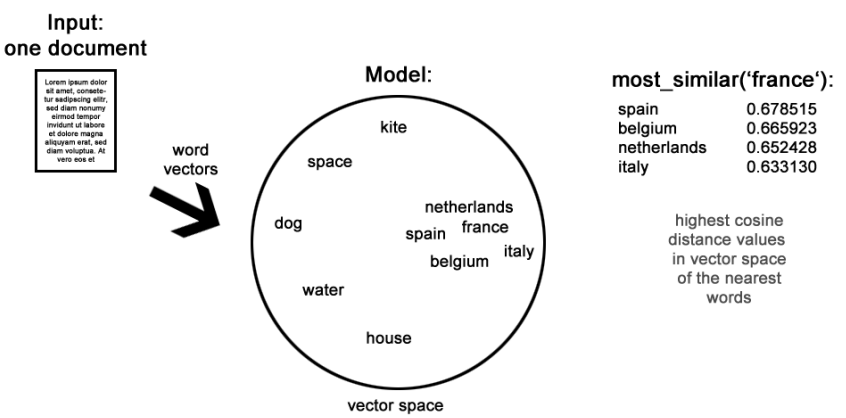
- Word Embedding ist ein Sammelbegriff für eine Reihe von Sprachmodellierungs- und Feature-Lerntechniken in der Verarbeitung natürlicher Sprache, bei denen Wörter oder Wortgruppen aus dem gegebenen Vokabular auf Vektoren reeller Zahlen abgebildet werden.
- Jedes Wort soll auf einen beliebig dimensionalen Vektor abgebildet werden.
- Die Vektoren werden mit neuronalen Netzwerken gelernt, daher wird diese Technik oft dem Gebiet des Deep Learnings zugeschrieben
- Manuelles Feature Engineering nicht notwendig!

Ziel:

Word Embeddings sind eine Vektordarstellung für einzelne Worte, in der Wörter mit ähnlicher Bedeutung auch zu einander ähnliche Vektoren haben.

11

word2vec



2013 erstellte ein Team von Google unter der Leitung von *Tomas Mikolov* **word2vec**, ein Word Embedding-Toolkit, das Vektorraummodelle für Worte schneller trainieren kann als die bisherigen Ansätze. https://arxiv.org/pdf/1301.3781.pdf

12

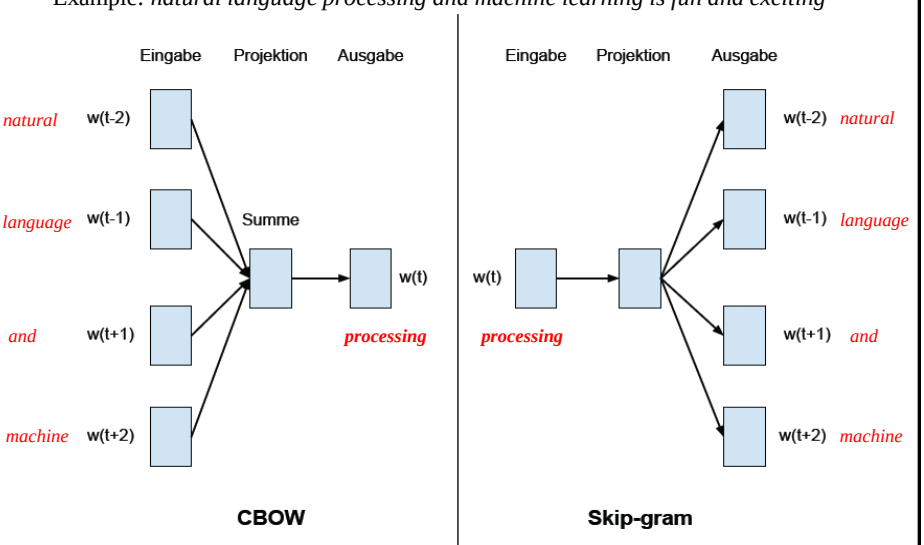
word2vec (automatische Codierung)

- Mit word2vec werden Worte als Vektoren repräsentiert.
- Dabei wird ein n-dimensionaler Vektorraum erzeugt, bei dem jedes Wort aus den Trainingsdaten durch einen Vektor repräsentiert wird, n wird vorgegeben
- Sage für jedes Wort vorher, mit welcher Wahrscheinlichkeit es von anderen Wörtern umgeben ist -> **bestimme Kontext eines Wortes**
- Wort Ähnlichkeit = Vektor Ähnlichkeit
- **word2vec** nimmt als Eingabe einen großen Korpus von Text und trainiert für jedes Wort dessen Kontext
- Realisierung mit flachem, zweischichtigem neuronalem Netz, das darauf trainiert ist, den sprachlichen Kontext von Wörtern zu erlernen
- Zwei Modelle werden verwendet:
  - **Skip-Gram (SG):** In der Skip-Gram-Architektur verwendet das Modell das aktuelle Wort, um das umgebende Fenster von Kontextwörtern vorherzusagen
  - **Continuous Bag of Word (CBOW):** In der CBOW-Architektur sagt das Modell das aktuelle Wort aus einem Fenster umgebender Kontextwörter voraus

13

word2vec : multi word context

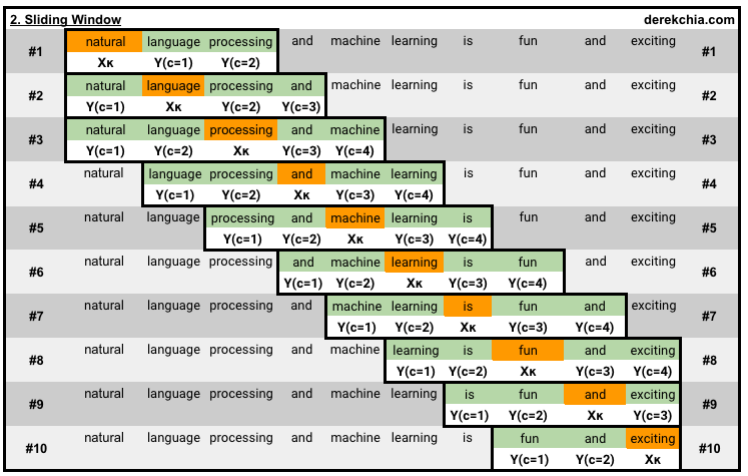
Example: *natural language processing and machine learning is fun and exciting*



14

word2vec: Define sliding window on training data
Example

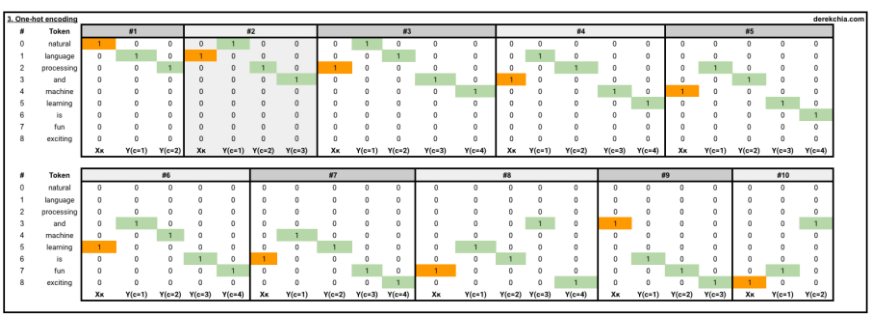
Example: *natural language processing and machine learning is fun and exciting*
Windowsize*:* 2



15

word2vec: One-hot-encode training data
Example

Example Doc: *natural language processing and machine learning is fun and exciting*
Windowsize *C*: 2



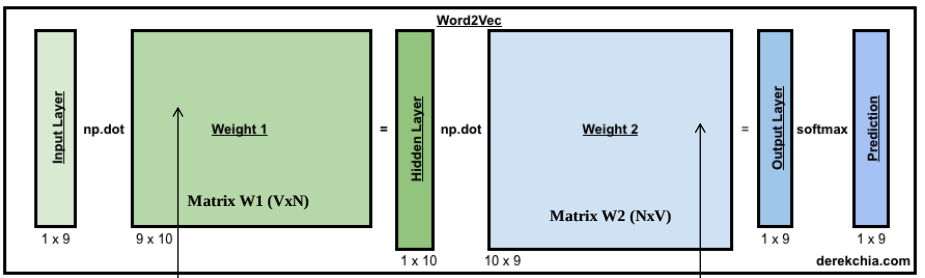
Skip-Gram:
Input: w_i (orange labeled in figure)
Output: context words $\{w_{O,1}\ldots\ldots w_{O,C}\}$ (green labeled in figure)

CBOW:
Input: context words $\{w_{I,1}\ldots\ldots w_{I,C}\}$ (green labeled in figure)
Output: w_o (orange labeled in figure)

16

word2vec: Skip Gram Model Architecture Example

Vocabulary Size V=9
Embedding Vector Size N=10



Weights of W1
randomly initialized

Weights of W2
randomly initialized

17

word2vec: Skip Gram Model Architecture Example

4. Skip-gram Model Architecture — derekchia.com

Forward Pass for #1

Parameters - Embedding size 10

Random initialisation (Range) -1 to 1

input word is at the inputlayer ("natural")
context words are on the output layer
("language","processing")

Calculate Hidden Layer

#	Token	Input - w_t
0	natural	1
1	language	0
2	processing	0
3	and	0
4	machine	0
5	learning	0
6	is	0
7	fun	0
8	exciting	0

1 x 9

np.dot

Weight 1 - W1

0.236	**-0.962**	**0.686**	**0.785**	**-0.454**	**-0.833**	**-0.744**	**0.677**	**-0.427**	**-0.066**
-0.907	0.894	0.225	0.673	-0.579	-0.428	0.685	0.973	-0.070	-0.811
-0.576	0.658	-0.582	-0.112	0.662	0.051	-0.401	-0.921	-0.158	0.529
0.517	0.436	0.092	-0.835	-0.444	-0.905	0.879	0.303	0.332	-0.275
0.859	-0.890	0.651	0.185	-0.511	-0.456	0.377	-0.274	0.182	-0.237
0.368	-0.867	-0.301	-0.222	0.630	0.808	0.088	-0.902	-0.450	-0.408
0.728	0.277	0.439	0.138	-0.943	-0.409	0.687	-0.215	-0.807	0.612
0.593	-0.699	0.020	0.142	-0.638	-0.633	0.344	0.868	0.913	0.429
0.447	-0.810	-0.061	-0.495	0.794	-0.064	-0.817	-0.408	-0.286	0.149

9 x 10

=

Hidden Layer - h

0.236
-0.962
0.686
0.785
-0.454
-0.833
-0.744
0.677
-0.427
-0.066

1 x 10

activation function after W1 linear

Calculate y_pred

Hidden Layer - h

0.236
-0.962
0.686
0.785
-0.454
-0.833
-0.744
0.677
-0.427
-0.066

1 x 10

np.dot

Weight 2 - W2

-0.868	-0.406	-0.288	-0.016	-0.560	0.179	0.099	0.438	-0.551
-0.395	0.890	0.685	-0.329	0.218	-0.852	-0.919	0.665	0.968
-0.128	0.685	-0.828	0.709	-0.420	0.057	-0.212	0.728	-0.690
0.881	0.238	0.018	0.622	0.936	-0.442	0.936	0.586	-0.020
-0.478	0.240	0.820	-0.731	0.260	-0.989	-0.626	0.796	-0.599
0.679	0.721	-0.111	0.083	-0.738	0.227	0.560	0.929	0.017
-0.690	0.907	0.464	-0.022	-0.005	-0.004	-0.425	0.299	0.757
-0.054	0.397	-0.017	-0.563	-0.551	0.465	-0.596	-0.413	-0.395
-0.838	0.053	-0.160	-0.164	-0.671	0.140	-0.149	0.708	0.425
0.096	-0.995	-0.313	0.881	-0.402	-0.631	-0.660	0.184	0.487

10 x 9

=

Output Layer

1.258
-1.369
-1.828
1.196
0.545
1.113
1.333
-1.528
-2.335

1 x 9

Softmax - y_pred

0.218
0.016
0.010
0.205
0.107
0.189
0.235
0.013
0.006

1 x 9

word2vec: Skip Gram Model Architecture Error Calc. Example

Error For #1 — derekchia.com

y_pred Softmax	w_c = 1	Token	Diff	y_pred Softmax	w_c = 2	Token	Diff	EI Sum of Diff
0.218	0	natural	0.218	0.218	0	natural	0.218	0.436
0.016	1	language	-0.984	0.016	0	language	0.016	-0.968
0.010	0	processing	0.010	0.010	1	processing	-0.990	-0.980
0.205	0	and	0.205	0.205	0	and	0.205	0.411
0.107	0	machine	0.107	0.107	0	machine	0.107	0.214
0.189	0	learning	0.189	0.189	0	learning	0.189	0.378
0.235	0	is	0.235	0.235	0	is	0.235	0.471
0.013	0	fun	0.013	0.013	0	fun	0.013	0.027
0.006	0	exciting	0.006	0.006	0	exciting	0.006	0.012

Calculating Error—context words are ‘language’ and ‘processing’

Error Function

$$
\begin{aligned}
E &= -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,C} \mid w_I) \\
&= -\log \prod_{c=1}^{C} \frac{exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} exp(u_{j'})} \\
&= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} exp(u_{j'})
\end{aligned}
$$

For detailed math for error function and backpropgation in word2vec see:
word2vec Parameter Learning Explained

19

word2vec: Coding Example

W2: Look Up Table

#	Token	Input - w_t
0	natural	1
1	language	0
2	processing	0
3	and	0
4	machine	0
5	learning	0
6	is	0
7	fun	0
8	exciting	0

1 x 9

Weight 2 - W2

-0.868	-0.406	-0.288	-0.016	-0.560	0.179	0.099	0.438	-0.551
-0.395	0.890	0.685	-0.329	0.218	-0.852	-0.919	0.665	0.968
-0.128	0.685	-0.828	0.709	-0.420	0.057	-0.212	0.728	-0.690
0.881	0.238	0.018	0.622	0.936	-0.442	0.936	0.586	-0.020
-0.478	0.240	0.820	-0.731	0.260	-0.989	-0.626	0.796	-0.599
0.679	0.721	-0.111	0.083	-0.738	0.227	0.560	0.929	0.017
-0.690	0.907	0.464	-0.022	-0.005	-0.004	-0.425	0.299	0.757
-0.054	0.397	-0.017	-0.563	-0.551	0.465	-0.596	-0.413	-0.395
-0.838	0.053	-0.160	-0.164	-0.671	0.140	-0.149	0.708	0.425
0.096	-0.995	-0.313	0.881	-0.402	-0.631	-0.660	0.184	0.487

10 x 9

- Gewichtsmatrix W2 enthält nach dem Training fertige Wortvektoren
- Vektordarstellung des Wortes entspricht Gewichten, die beim Training gelernt wurden

natural: [-0.868, -0.395, -0.128, ……0.096]
word vector

20

Word Embeddings in Python: Verwendung

Word Embeddings können in Python mit Hilfe:

- Eines bereits vortrainiertes **embedding model** (selbst erzeugt oder frei verfügbar im Internet) verwendet werden, das bereits vordefinierte Vektoren für Worte enthält

  Beispiel:

  - Viele verschiedene **word2vec**-Implementierungen vorhanden
  - **Gensim:** Python-Bibliothek, die viele NLP-Aufgaben erledigt, auch Implementierung von word2vec beinhaltet**.**
  - siehe Beispielcode: **emb_word2vec.py**
- eines eigenen **embedding layers** (von Keras zur Verfügung gestellt) selbst erzeugt werden

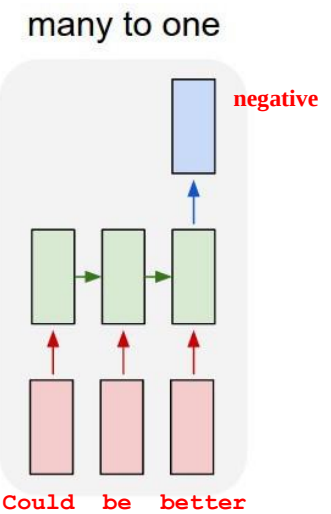
21

Embedding layer

- Alternative zu den mit **word2vec** trainierten Modellen zur Repräsentation von Wörtern in einem Vektor-Raum ist, einen **embedding Layer** zu verwenden.
- Bereits mit einem verhältnismäßig kleinem Korpus lassen sich hiermit insbesondere im Bereich der Text-Klassifizierung sehr gute Ergebnisse erzielen.
- Es wird ein vorklassifizierter Trainings-Datensatz benötigt. Damit kann dann ein sogenannter Embedding Layer eines neuronalen Netzes darauf trainiert werden, den Vektor für jedes Wort möglichst so zu wählen, dass er möglichst nah an den Klassen liegt, in denen das Wort häufig vorkommt und weniger nah an Klassen, in denen es wenig oder gar nicht vorkommt.
- Die so generierten Vektoren können als Input für ein Recurrent Neural Network (RNN) wie z.B. ein LSTM verwendet werden, und so jedes Wort einzeln und in der richtigen Reihenfolge einlesen und das RNN die Klassifizierung lernen lassen.

22

Beispiel Modell Sequenzklassifikation
Sentiment Analyse mit embedding layer und LSTM



23

Beispiel: Sequenzklassifikation - Sentiment Analyse
Textklassifikation mit Keras

```
# define documents
docs = [      'Well done!',
              'Good work',
              'Great effort',
              'nice work',
              'Excellent!',
              'Weak',
              'Poor effort!',
              'not good',
              'poor work',
              'Could have done better.']
# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
```

```
# integer encode the documents, simulate vocabulary
vocab_size = 50 # choose size of vocabulary
encoded_docs = [one_hot(d, vocab_size) for d in docs]
```

24

Beispiel: Sequenzklassifikation - Sentiment Analyse Datenvorbereitung

Die vorher generierten Eingabesequenzen sind so mit Null aufzufüllen, dass sie alle die gleiche Länge haben, hier die Länge 4.
Das kann mit der Funktion **`pad_sequences()`** geschehen.

```
# pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length,
              padding='post')
print(padded_docs)
```

Integer encoded and padded documents:

```
[[ 4  8  0  0]
 [ 6 22  0  0]
 [20 21  0  0]
 [23 22  0  0]
 [24  0  0  0]
 [14  0  0  0]
 [23  7  0  0]
 [20  6  0  0]
 [20  0  0  0]
 [ 9 28  8 17]]
```

25

Beispiel: Sequenzklassifikation - Sentiment Analyse Modell

Verwendung des Embedding Layers:
Input mit jeweils 4 Integerwerten. Pro Wort wird ein 8 dimensionaler Outputvektor erzeugt.
Diese Vektoren werden einem LSTM Layer mit 5 Neuronen übergeben.
Der finale Dense Layer wird für die Klassifikation der Docs verwendet.

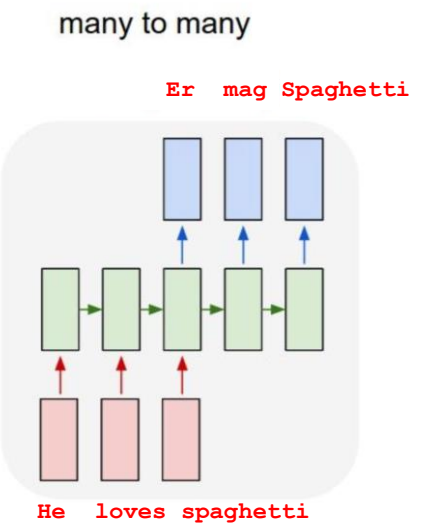
```
# define the model
model = Sequential()
model.add(Embedding (vocab_size, 8, input_length = max_length))
model.add(LSTM(5))
model.add(Dense(1, activation='sigmoid'))

# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
                                          metrics=['acc'])
# fit the model
model.fit(padded_docs, labels, epochs=50, verbose=0)

# evaluate the model
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
```

26

Beispiel Modell Sequenz-zu-Sequenz-Vorhersage
Übersetzer für Sprachen (Natural Language Translation)



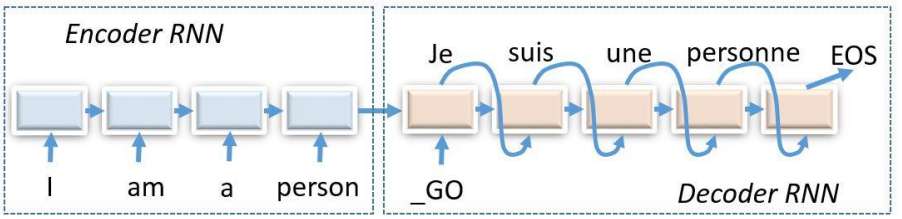
27

Sequenz-zu-Sequenz-Vorhersage
Übersetzer für Sprachen (Neural Machine Translation)

- **Neural Machine Translation** ist der Ansatz die Modellierung des gesamten Übersetzungsprozesses über ein neuronales Netzwerk vorzunehmen, OHNE dabei auf statistisches MT (SMT) zurückzugreifen
- Architekturen beruhen auf einem gemeinsamen Paradigma, das *Codierer-Decodierer* (encoder-decoder) genannt, (auch Kerntechnologie von Google Translate)
- *Codierer* codiert die Eingangssequenz in Sprache A, während der *Decodierer* die Zielsequenz in Sprache B erzeugt
- Sprachübersetzung erreicht heute teilweise menschliche Übersetzungs-leistung (siehe Google Translate)
- **Voraussetzung:** riesiger, identischer Corpus in beiden Sprachen zum Trainieren
  - Beispiel für Englisch / Französisch: 12 Millionen Sätze insgesamt
  - 348 Millionen französiche Worte und 304 Millionen englische Worte

28

Beispiel: RNN encoder-decoder network



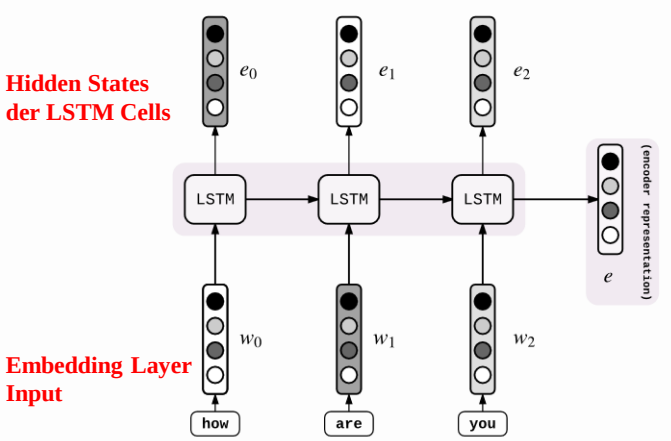
RNN-Codierer-Decodierer besteht aus

- zwei rekurrenten neuronalen Netzen, die als Codierer- und Decodierer fungieren, basierend auf LSTM-Zellen bzw. GRU-Zellen bzw. simplen RNN Neuronen.
- Der Codierer codiert den gesamten Eingabesatz in einen Zustandsvektor, der vom Decodierer als Eingabe verwendet wird. Dieser Zustandsvektor ist eine abstrakte Darstellung des gesamten Eingabesatzes als einzelner Punkt in einem hoch-dimensionalen Raum.
- Der Decoder wird trainiert, diesen Punkt als Ausgangspunkt zum Ausrollen einer übersetzten Version des Eingabesatzes zu verwenden

29

Encoder

Our input sequence is `how are you`. Each word from the input sequence is associated to a vector $w \in \mathbb{R}^d$ (via a lookup table). In our case, we have 3 words, thus our input will be transformed into $[w_0, w_1, w_2] \in \mathbb{R}^{d \times 3}$. Then, we simply run an LSTM over this sequence of vectors and store the last hidden state outputed by the LSTM: this will be our encoder representation e. Let's write the hidden states $[e_0, e_1, e_2]$ (and thus $e = e_2$)



Quelle: https://guillaumegenthial.github.io/sequence-to-sequence.html

30

Decoder

Now that we have a vector e that captures the meaning of the input sequence, we'll use it to generate the target sequence word by word. Feed to another LSTM cell: e as hidden state and a special *start of sentence* vector w_{sos} as input. The LSTM computes the next hidden state $h_0 \in \mathbb{R}^h$. Then, we apply some function $g : \mathbb{R}^h \mapsto \mathbb{R}^V$ so that $s_0 := g(h_0) \in \mathbb{R}^V$ is a vector of the same size as the vocabulary.

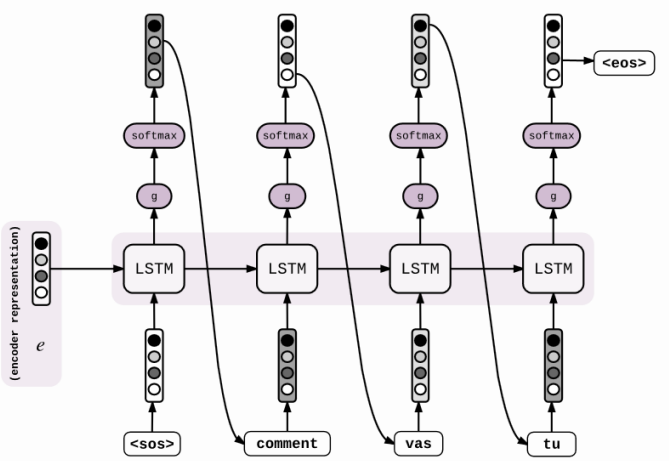
$$
\begin{aligned}
h_0 &= \mathrm{LSTM}(e, w_{sos}) \\
s_0 &= g(h_0) \\
p_0 &= \mathrm{softmax}(s_0) \\
i_0 &= \mathrm{argmax}(p_0)
\end{aligned}
$$

Then, apply a softmax to s_0 to normalize it into a vector of probabilities $p_0 \in \mathbb{R}^V$. Now, each entry of p_0 will measure how likely is each word in the vocabulary. Let's say that the word *"comment"* has the highest probability (and thus $i_0 = \mathrm{argmax}(p_0)$ corresponds to the index of *"comment"*). Get a corresponding vector $w_{i_0} = w_{comment}$ and repeat the procedure: the LSTM will take h_0 as hidden state and $w_{comment}$ as input and will output a probability vector p_1 over the second word, etc.

$$
\begin{aligned}
h_1 &= \mathrm{LSTM}(h_0, w_{i_0}) \\
s_1 &= g(h_1) \\
p_1 &= \mathrm{softmax}(s_1) \\
i_1 &= \mathrm{argmax}(p_1)
\end{aligned}
$$

31

Decoder



The above method aims at modelling the distribution of the next word conditionned on the beginning of the sentence

$$\mathbb{P}\left[y_{t+1} | y_1, \ldots, y_t, x_0, \ldots, x_n\right]$$

by writing

$$\mathbb{P}\left[y_{t+1} | y_t, h_t, e\right]$$

32

Google Translate

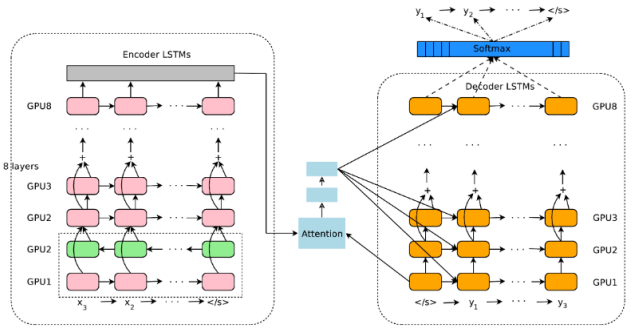


Figure 1: The model architecture of GNMT, Google's Neural Machine Translation system. On the left is the encoder network, on the right is the decoder network, in the middle is the attention module. The bottom encoder layer is bi-directional: the pink nodes gather information from left to right while the green nodes gather information from right to left. The other layers of the encoder are uni-directional. Residual connections start from the layer third from the bottom in the encoder and decoder. The model is partitioned into multiple GPUs to speed up training. In our setup, we have 8 encoder LSTM layers (1 bi-directional layer and 7 uni-directional layers), and 8 decoder layers. With this setting, one model replica is partitioned 8-ways and is placed on 8 different GPUs typically belonging to one host machine. During training, the bottom bi-directional encoder layers compute in parallel first. Once both finish, the uni-directional encoder layers can start computing, each on a separate GPU. To retain as much parallelism as possible during running the decoder layers, we use the bottom decoder layer output only for obtaining recurrent attention context, which is sent directly to all the remaining decoder layers. The softmax layer is also partitioned and placed on multiple GPUs. Depending on the output vocabulary size we either have them run on the same GPUs as the encoder and decoder networks, or have them run on a separate set of dedicated GPUs.

https://arxiv.org/pdf/1609.08144.pdf

33