Computer-Linguistische Anwendungen

CLA | B.Sc. | LMU

Vorlesung: Neuronale Netze

Philipp Wicke, PhD
Centrum für Sprach- und Informationsverarbeitung
Ludwig-Maximilians-Universität München
pwicke@cis.lmu.de

Convolutional Neural Networks

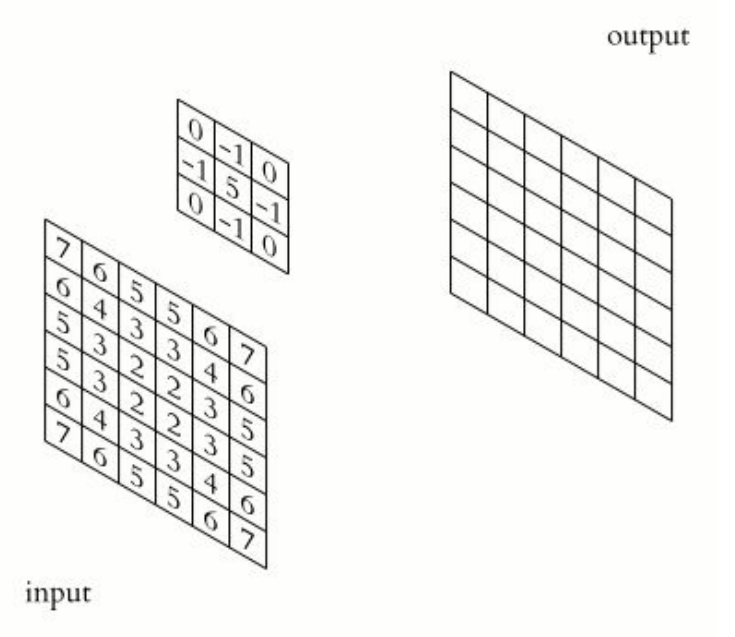
Übersicht

- CNNs - Convolutional Neural Networks
- CNNs - Pooling
- CNNs - Pooling in NLP
- CNNs vs. RNNs

Convolutional Neural Networks

CNNs - Einführung

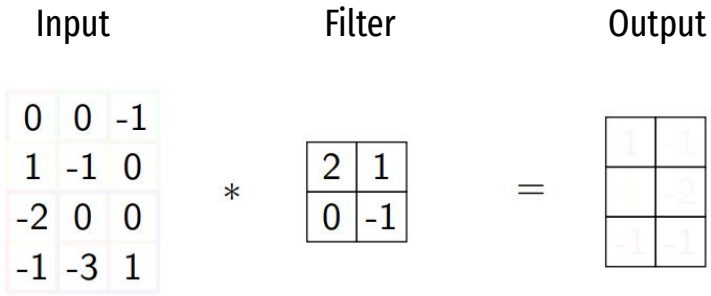
- Convolution: Zwei Funktionen liefern eine dritte Funktion
- Kommt ursprünglich aus der Computer Vision (z.B. Objekterkennung aus Bildern)
- Alternative zu RNNs für viele (nicht alle) NLP-Aufgaben
- Grundidee: Filterbank mit N lernbaren Filtern (Detektoren)
    - Beispiel: angenommen, es gibt einen Filter
    - Bewege den Filter mit einer Schrittweite *s* (hier: 1)
    - An jeder Position werden Filter und Einträge eines Ausschnitts aus Eingabe-Matrix elementweise multipliziert und aufsummiert, was jeweils einen Ergebniswert liefert.



https://commons.wikimedia.org/wiki/File:2D_Convolution_Animation.gif

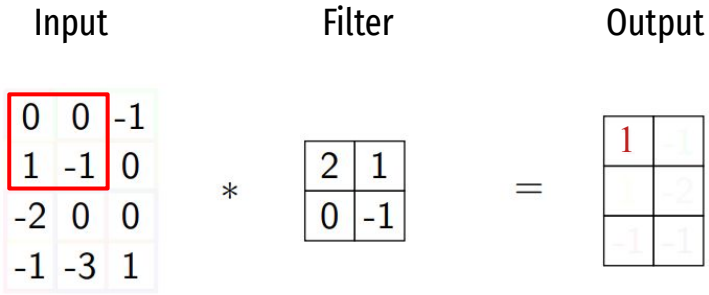
Convolutional Neural Networks

CNNs - Einführung



Convolutional Neural Networks

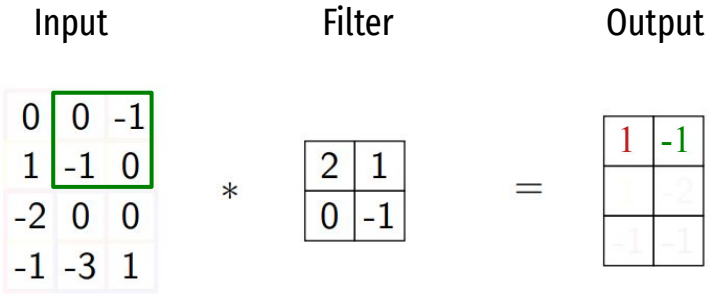
CNNs - Einführung



0*2 + 1*0 + 1*0 + (-1)*(-1) = 1

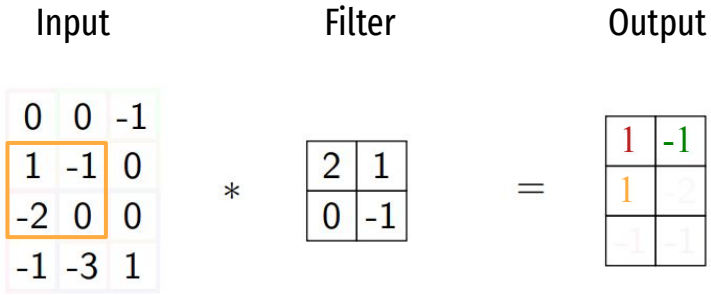
Convolutional Neural Networks

CNNs - Einführung



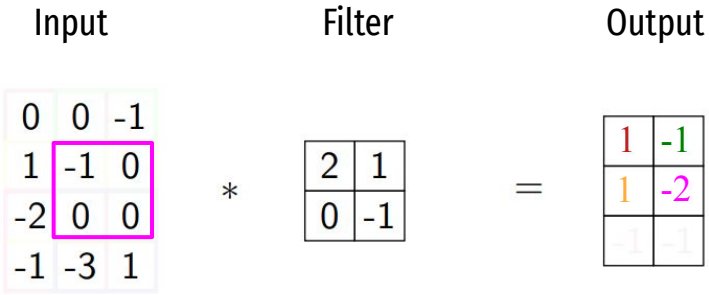
Convolutional Neural Networks

CNNs - Einführung



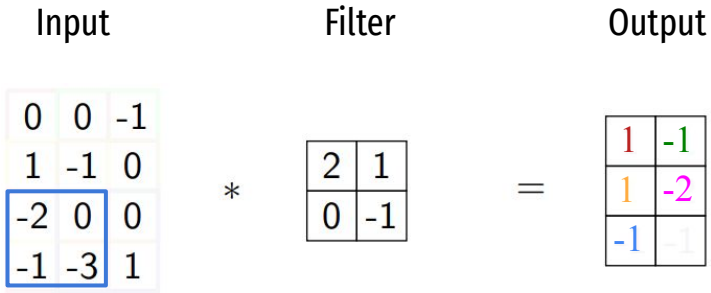
Convolutional Neural Networks

CNNs - Einführung



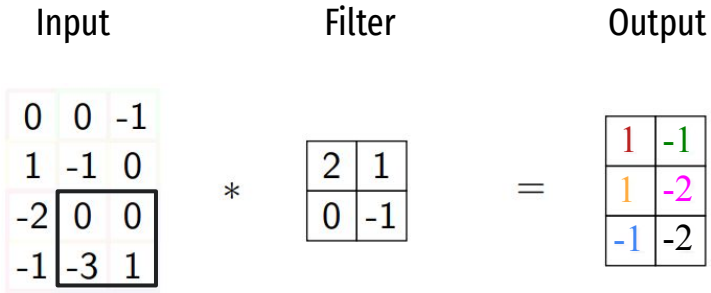
Convolutional Neural Networks

CNNs - Einführung



Convolutional Neural Networks

CNNs - Einführung

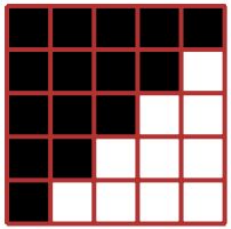


Filter size: 2x2
Input size: 4x3
Output size: 3x2

Convolutional Neural Networks

CNNs - Einführung

- Angenommen -1 steht für schwarz und +1 für weiß
- Wir wollen einen Filter konstruieren, der diagonale Objektgrenzen erkennt, die oben links dunkel sind und unten rechts hell. (Für andere Grenzen gibt es zusätzliche, ähnlich konstruierte Filter)
- = ein Filter der hohe positive Werte auf Bildausschnitten wie diesem berechnet:

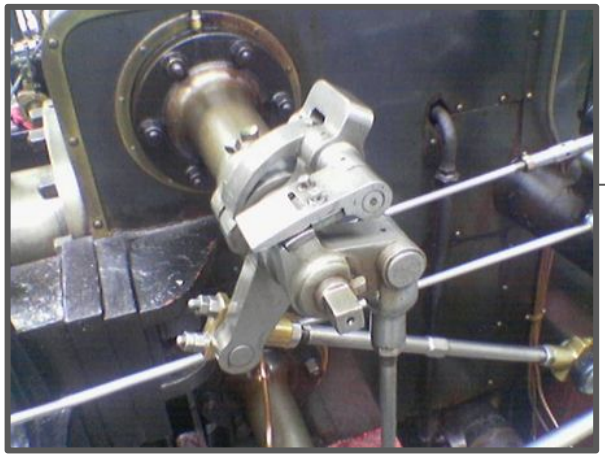


-3	-3	-3	-3	-3
-3	-3	-3	-3	3
-3	-3	-3	3	3
-3	-3	3	3	3
-3	3	3	3	3

Convolutional Neural Networks

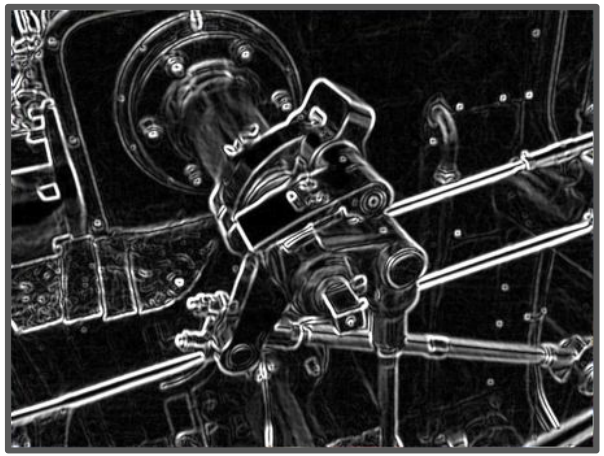
CNNs - Einführung

- Filter um Kanten zu Erkennen werden in der Computer Vision verwendet
- Zum Beispiel der Sobel Filter:



$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and}$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

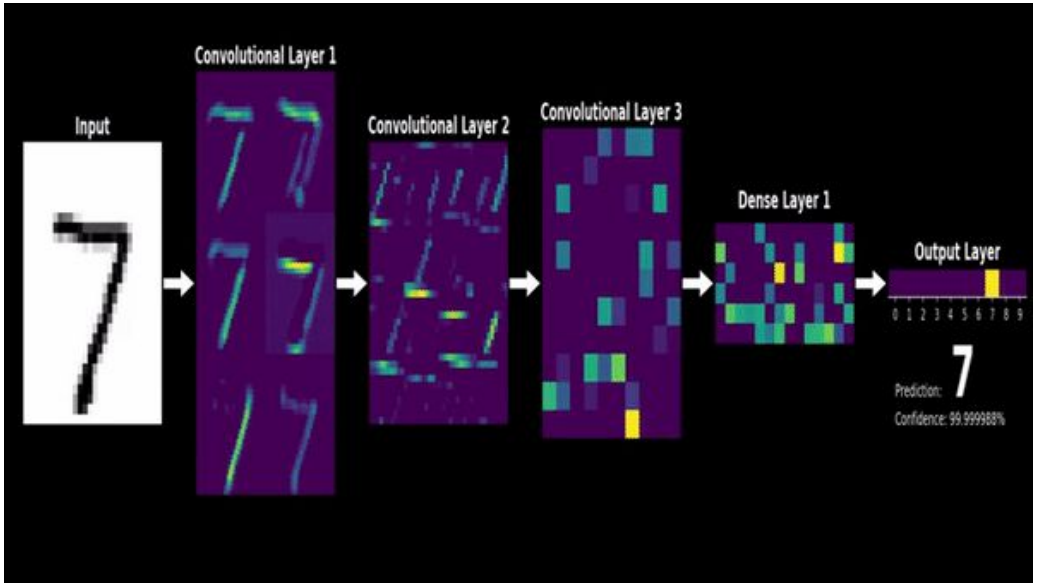


https://en.wikipedia.org/wiki/File:Valve_original_(1).PNG

Convolutional Neural Networks

CNNs - Einführung

- Im CNN werden die Filter (Kernel) nicht konstruiert, sondern mit Gradient Descent gelernt:



https://github.com/charliedavenport/LeNet-MNIST-Demo

Convolutional Neural Networks

CNNs - Einführung

- Bilder sind meist nicht 2D sondern 3D
    - 3te Dimension ist # der Farbkanäle (Channels), z.B., RGB-Werte
    - Höhe × Breite × # Channels

- In diesem Fall ist jeder Filter auch 3D
    - *filter height × filter width × # channels*

- Die Operation ist die gleiche, nur wird die Summation auch über die Channel-Dimension ausgeführt
- Für jeden der N Filter ergibt sich jeweils eine Ausgabe-Matrix von Werten
- Werden die N aufeinander gelegt ergibt sich ein 3D tensor (mit letzter Dimension der Größe N)
- X- und Y-Achsen dieser Ausgabe sind etwas kleiner als die Eingabe? (Warum)

Convolutional Neural Networks

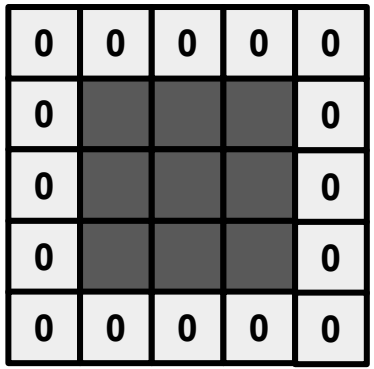
CNNs - Einführung

- X- und Y-Achsen dieser Ausgabe sind etwas kleiner als die Eingabe? (Warum)

  ... weil ein Filter mit Gr¨ oße k in eine Eingabe mit Größe h nur (h - k + 1)-Mal hineinpasst.

  ... außer die Eingabe wird durch 0-Padding etwas vergrößert.

Zero-Padding →



Convolutional Neural Networks

Tensor Größen

Tensor-Größen:

- **Input 3D**: Eingabe Höhe × Eingabe Breite × # Channels (bzw. #Filter aus der vorhergehenden Layer)
- **Parameter tensor 4D**: Filter Höhe × Filter Breite × # Eingabe-Channels × #Filter (Ausgabe-Channels)
- **Output 3D**: Eingabe Höhe* × Eingabe Breite* × #Filter

*Höhe und Breite werden etwas reduziert, außer 0-Padding wird angewendet

Convolutional Neural Networks

Was bewirkt Convolution?

- Kontextualisierung: Repräsentation für Position (i, j) enthält Information von (i, j) bis (i + k, j + k) (k ist die Filtergröße).
- Lokalität wird erhalten: innerhalb einer Convolution-Layer, kann Information nur maximal k Positionen wandern
- Oft werden mehrere Convolutions nacheinander angewandt
- Typische Nichtlinearität nach einer Convolution: **ReLU**
- Pro Layer werden die extrahierten Merkmale immer abstrakter
- Pixels → edges → shapes → small objects → bigger, compositional

https://poloclub.github.io/cnn-explainer/

Convolutional Neural Networks

Pooling

- Wird oft nach Convolution-Layers angewandt
- Bildverarbeitung: Ausgabe Tensor wird in Abschnitte unterteilt (“grid”)
- Werte innerhalb eines Abschnitts werden kombiniert
- Häufig: Durchschnittswert (Average pooling), Maximalwert (Max pooling)
- Max pooling: wähle pro Filter nur den Maximalwert (“maximale Aktivierung”)

2	1	2	0
1	0	2	0
0	4	2	3
0	5	1	0

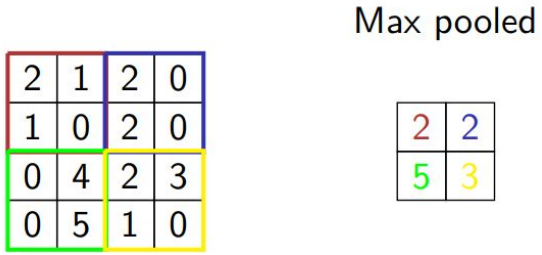
Max pooled

2	2
5	3

Convolutional Neural Networks

Pooling

- Wird oft nach Convolution-Layers angewandt
- Bildverarbeitung: Ausgabe Tensor wird in Abschnitte unterteilt (“grid”)
- Werte innerhalb eines Abschnitts werden kombiniert
- Häufig: Durchschnittswert (Average pooling), Maximalwert (Max pooling)
- Max pooling: wähle pro Filter nur den Maximalwert (“maximale Aktivierung”)



Convolutional Neural Networks

CNNs für NLP

- Images have width and height, but text only has “width” (length)
  → We can discard the “height” dimension from our filters

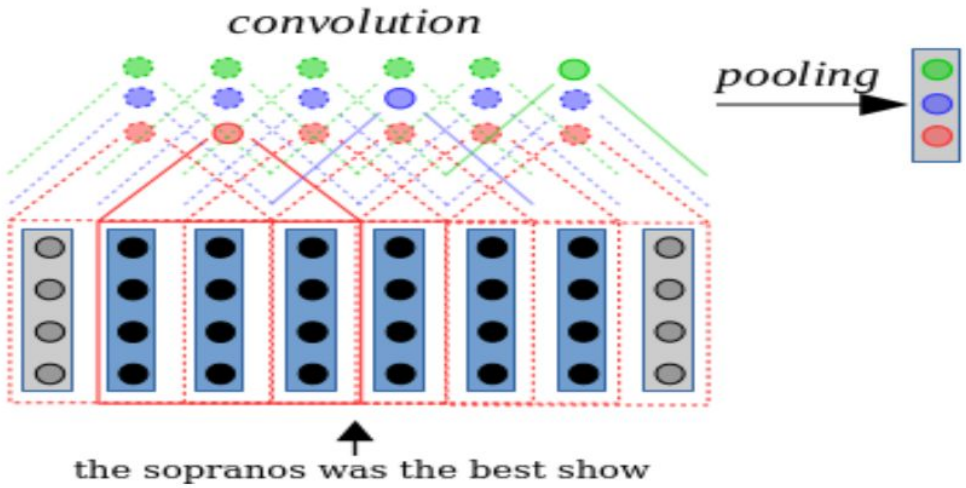
- Tensor sizes (in NLP):
  - **Input 2D**: sentence length × # channels (word embedding size, or # filters of previous convolution)
  - **Parameter tensor 3D**: filter length × # channels × #filters
  - **Output 2D**: sentence length × #filters (length slightly reduced unless we do padding)

- Computer vision: 2D convolution (over height and width)
- NLP: 1D convolution (over length)
- Typically fewer convolutional layers than Computer Vision

Convolutional Neural Networks

Pooling für NLP

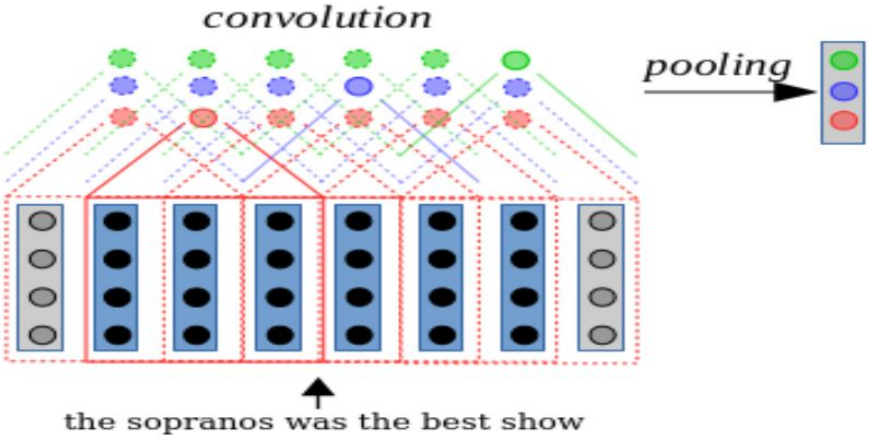
- Pooling between convolutional layers less frequently used than in Computer Vision
- After last convolutional layer: “global” pooling step
- Calculate max/average over the entire sequence (“pooling over time”)



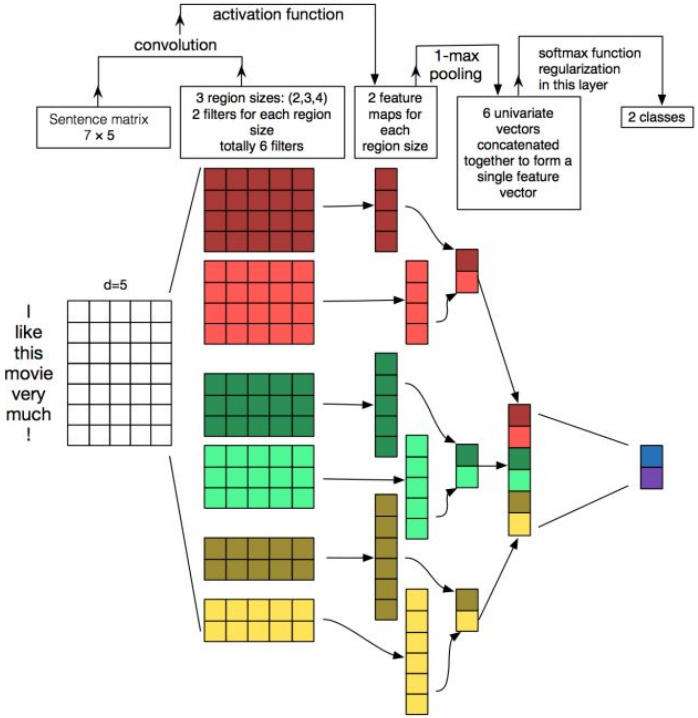
Convolutional Neural Networks

Pooling für NLP

- What is the unpadded input size (=length)? 6
- What is the padded input size? 8
- How many filters? **3**
- How many input channels (=word vector dimensions)? **4**
- What is the filter size (=filter width)? **3**
- What stride (=step size)? 1
- What is the output size of the convolution operation? 6 × 3
- What is the output size of the pooling operation? 3
- How many parameters have to be learned? 3 × 3 × 4 = 36

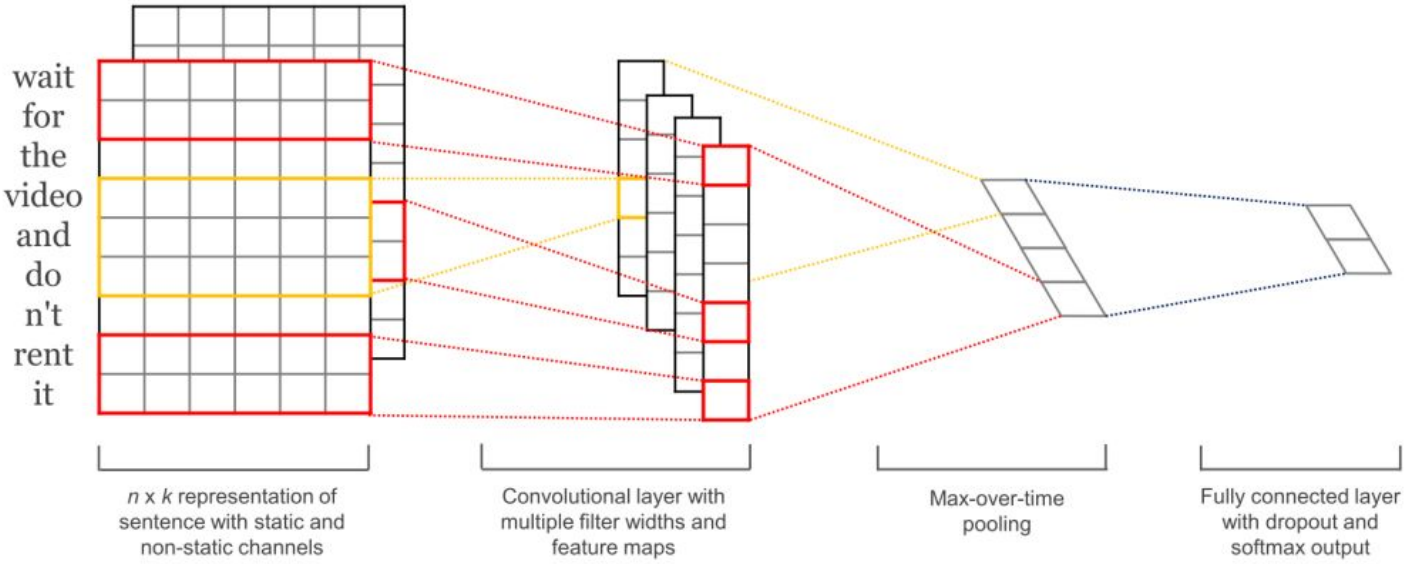


Convolutional Neural Networks



Source: Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of ConvNets for Sentence Classification.

Convolutional Neural Networks



Source: Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification.

Convolutional Neural Networks

- CNN können ähnlich einfach verwendet werden wie RNN's.
- Um z.B. für **Sentiment Vorhersage** ein CNN mit 50 Filtern (Ausgabe-Dimensionen) und Filterweite 3 Wörtern zu erzeugen ...

  ... muss statt der Zeile `model.add(LSTM(...))` ein CNN mit max-Pooling verwendet werden:

```
...
model.add(Conv1D(filters=50, kernel_size=3, \
activation='relu', padding='same'))
model.add(GlobalMaxPooling1D())
...
```

Convolutional Neural Networks

```
# binary classifier, e.g., sentiment polarity

from keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from keras.models import Sequential

embedding = Embedding(input_dim = VOCAB_SIZE, output_dim = EMB_DIM)
conv_layer = Conv1D(filters = NUM_FILTERS, kernel_size = FILTER_WIDTH,
        activation = "relu")
pool_layer = GlobalMaxPooling1D()
dense_layer = Dense(units = 1, activation = "sigmoid")

model = Sequential(layers = [emb_layer, conv_layer, pool_layer, dense_layer])
model.compile(loss = "binary_crossentropy", optimizer = "sgd")
X, Y = # load_data()
model.fit(X, Y)
```

Convolutional Neural Networks

CNN vs RNN

Reichweite

- **CNN:** Kann Abhängigkeiten mit einer Reichweite über k × L nicht erfassen (wobei k die Filterbreite und L die Anzahl der Schichten ist).
- **RNN:** Kann weitreichende Abhängigkeiten erfassen.

Informationsübertragung

- **CNN:** Keine Informationsübertragung über die Zeit, herausragende Informationen werden durch globales Max-Pooling "beschleunigt".
- **RNN:** Muss lernen, herausragende Informationen über viele Zeitschritte hinweg zu "transportieren".

Effizienz

- **CNN:** Eingabefenster sind voneinander unabhängig → stark parallelisierbar über die Zeit.
- **RNN:b**

Convolutional Neural Networks

CNN vs RNN: Quiz

- Given a task description, choose appropriate architecture!

  Task: predict the number of the main verb (sleep or sleeps)
    - *The cats, who were sitting on the map inside the house,* [sleep/sleeps?]
  - Which architecture should we use?

  Task: predict the polarity of the review:
    - *[... many useless sentences ...] best book ever [... many useless sentences ...]*
  - Which architecture should we use?

- Task: Machine Translation
  - Which architecture should we use?

Convolutional Neural Networks

CNN vs RNN: Quiz

- Given a task description, choose appropriate architecture!

  Task: predict the number of the main verb (sleep or sleeps)
    - *The cats, who were sitting on the map inside the house,* [sleep/sleeps?]
    - Which architecture should we use? RNN

  Task: predict the polarity of the review:
    - *[... many useless sentences ...] best book ever [... many useless sentences ...]*
    - Which architecture should we use? CNN

- Task: Machine Translation
    - Which architecture should we use?
      Intuitively RNN (because MT is all about long-range dependencies), but ...
    - Attention gives CNNs the ability to capture long-range dependencies, while maintaining parallel processing (Gehring et al.)

Convolutional Neural Networks

CNN vs RNN: Quiz

Rekurrente Neuronale Netzwerke

- Erzeugen eine Sequenz von Vektoren (hidden states).
- Jeder hidden Vektor wird rekursiv aus dem vorhergehenden hidden State, und dem Wort-Embedding der aktuellen Position berechnet.
- Eine Sequenz kann z.B. durch den letzten hidden State dargestellt werden.

Layers: Funktionen, die Arrays (Vektoren, Matrizen, ...) auf Arrays abbilden

Keras:
Um mit Keras produktiv arbeiten zu können ist es wichtig, sich mit der API/Dokumentation vertraut zu machen!
Keras erwartet als Eingaben (inputs) Numpy arrays. Listen verschiedener Länge (z.B. Satzrepräsentationen)