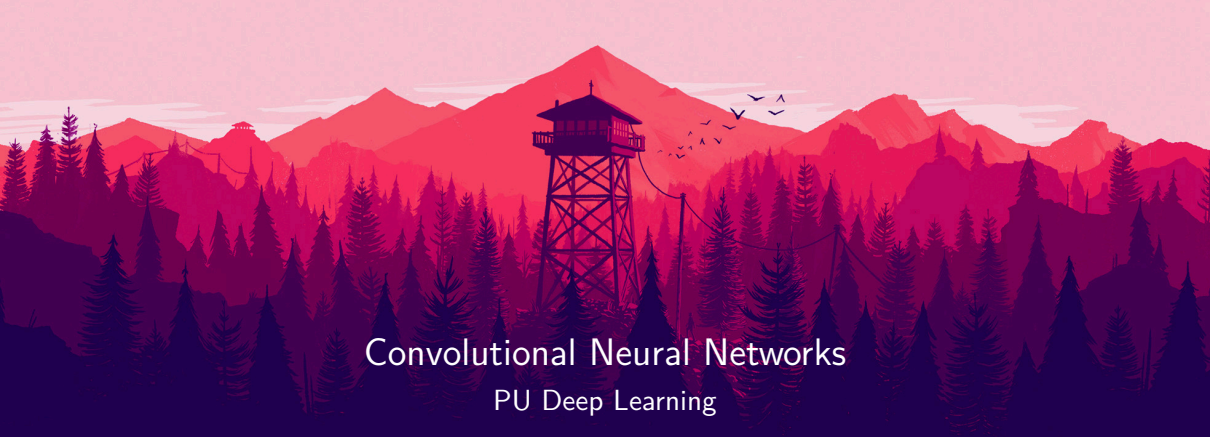


Convolutional Neural Networks

PU Deep Learning

Nils Reiter, nils.reiter@uni-koeln.de

June 15, 2021 (Summer term 2021)

Recap

Embeddings Preparations

Recap

Embeddings Preparations Exercise 8

Introduction

- So far: ›bag of words‹
- We count the number of times a word appears in a text

Introduction

- So far: ›bag of words‹
- We count the number of times a word appears in a text
- This is not how language works
  - Example
    - After the bad predecessor, this movie was very good.
    - After the good predecessor, this movie was very bad.
  - Both sentences have the same feature vector, but different meanings
- Convolution: Take multi-word structures into account

Introduction

- So far: ›bag of words‹
- We count the number of times a word appears in a text
- This is not how language works
  - Example
    - After the bad predecessor, this movie was very good.
    - After the good predecessor, this movie was very bad.
  - Both sentences have the same feature vector, but different meanings
- Convolution: Take multi-word structures into account
- CNNs mostly used for image recognition
  - Text: RNNs are better ($\rightarrow$ next week!)

Intuition

- Moving window over the input
- For each window, we apply logistic regression
- And continue with a slightly shorter vector

Intuition

- Moving window over the input
- For each window, we apply logistic regression
- And continue with a slightly shorter vector

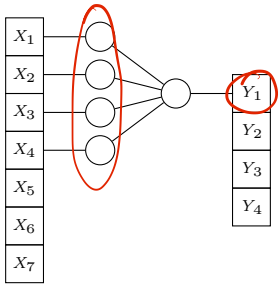


Figure: A 1D convolutional layer of size 4 (strides = 1)

Intuition

- Moving window over the input
- For each window, we apply logistic regression
- And continue with a slightly shorter vector

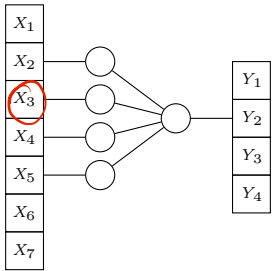


Figure: A 1D convolutional layer of size 4 (strides = 1)

Intuition

- Moving window over the input
- For each window, we apply logistic regression
- And continue with a slightly shorter vector

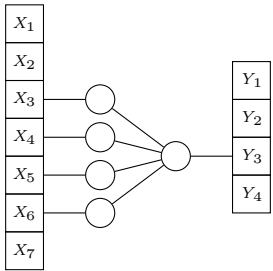


Figure: A 1D convolutional layer of size 4 (strides = 1)

Intuition

- Moving window over the input
- For each window, we apply logistic regression
- And continue with a slightly shorter vector

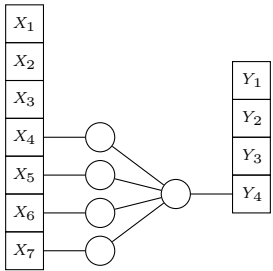


Figure: A 1D convolutional layer of size 4 (strides = 1)

1D Convolution

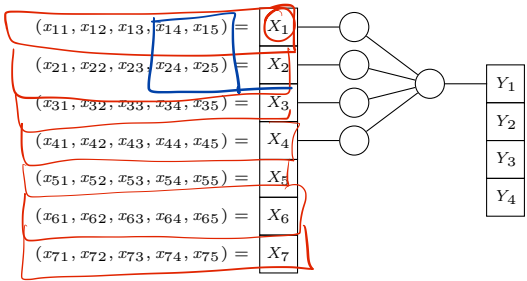


Figure: A 1D convolutional layer of size 4 (strides = 1)

- Requires a 2D input shape – because of embeddings!
  - (if you're not using embeddings, each token can be coded as a vector of length 1)

Convolution in Two Dimensions (e.g., images)

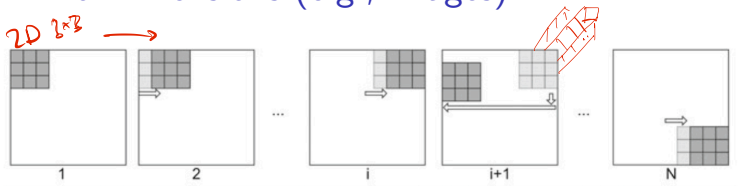


Figure: 2D convolutional layer (Skansi, 2018, p. 124)

Convolution

Parameters

- Natural language: 1D
  - Images: 2D/3D
- Size: How many vector cells are ›merged‹
- Rectified linear unit (ReLU): Common choice for activation function
- Output size
  - Above: Output is smaller than input
  - Padding the input allows enforcing same output size

Implementation

```
model.add(layers.Conv1D(filters=1,
  kernel_size=2, strides=1,
  activation="relu"))
```

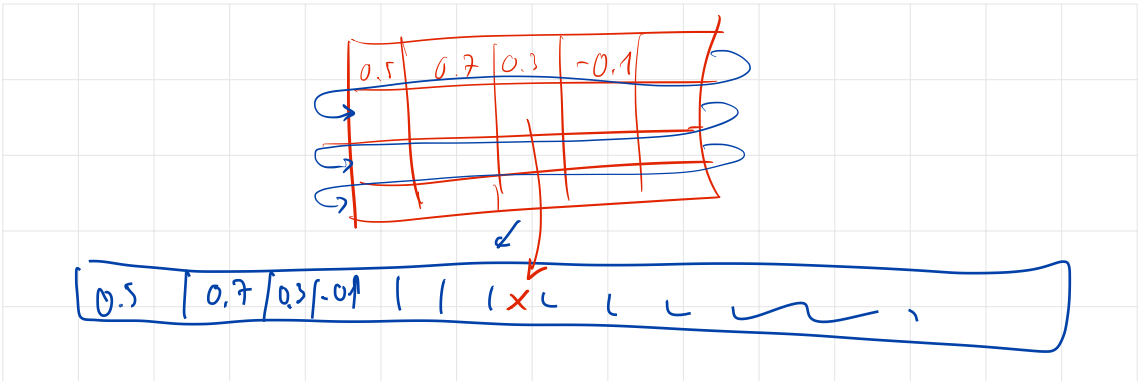
- (Some) arguments
  - `filters` Number of (differently initialized) logistic regressions to run
  - `kernel_size` Length of the window
  - `strides` The number of elements to shift the window
- Docs: `https://keras.io/api/layers/convolution_layers/convolution1d/`

Subsection 1

Flattening and Pooling

Flattening

- Remember that fully connected layers (`layers.Dense()`) work on vectors
- If the output of a layer is a matrix, it needs to be flattened, with `layers.Flatten()`



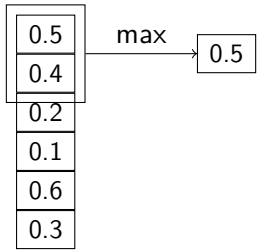
Pooling

- Convolutional layers are often combined with pooling layers
- A pooling layer looks at a data window and returns the maximum or average
  - For images, this corresponds to reducing the resolution
  - No clear interpretation for text

0.5
0.4
0.2
0.1
0.6
0.3

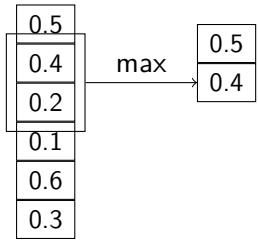
Pooling

- Convolutional layers are often combined with pooling layers
- A pooling layer looks at a data window and returns the maximum or average
  - For images, this corresponds to reducing the resolution
  - No clear interpretation for text



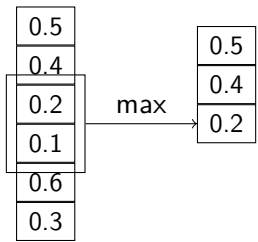
Pooling

- Convolutional layers are often combined with pooling layers
- A pooling layer looks at a data window and returns the maximum or average
  - For images, this corresponds to reducing the resolution
  - No clear interpretation for text



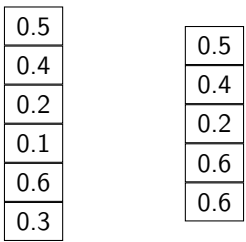
Pooling

- Convolutional layers are often combined with pooling layers
- A pooling layer looks at a data window and returns the maximum or average
  - For images, this corresponds to reducing the resolution
  - No clear interpretation for text



Pooling

- Convolutional layers are often combined with pooling layers
- A pooling layer looks at a data window and returns the maximum or average
  - For images, this corresponds to reducing the resolution
  - No clear interpretation for text



Implementation in Keras

Flattening (`layers.Flatten()`)

- Docs: `https://keras.io/api/layers/reshaping_layers/flatten/`
- `data_format` can be specified

Implementation in Keras

Flattening (`layers.Flatten()`)

- Docs: `https://keras.io/api/layers/reshaping_layers/flatten/`
- `data_format` can be specified

Pooling (`layers.MaxPooling1D()`, …)

- Docs: `https://keras.io/api/layers/pooling_layers/`
- `pool_size`: Width of the window
- `strides`: As for convolutional layers, the step size

Multi-Class Classification

So far: Binary classification

- Binary output value (survived/drowned, positive/negative, ...)
- Final network layer: `layers.Dense(1, activation="sigmoid")`
- Interpretation as a robability of the positive class
- Loss: `binary_crossentropy`

Multi-Class Classification

So far: Binary classification

- Binary output value (survived/drowned, positive/negative, ...)
- Final network layer: `layers.Dense(1, activation="sigmoid")`
- Interpretation as a robability of the positive class
- Loss: `binary_crossentropy`

Multi-class classification: More than two classes

- Output value: One-hot vector (e.g., $\langle 0, 0, 0, 1, 0, 0 \rangle$ for the fourth class)
  - `tensorflow.keras.utils.to_categorical(...)`
- Loss: `categorical_crossentropy`
- Fully connected output layer with n values (one for each class)
  - Activation function: `softmax`

Full Example

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_texts)
vocab_size = len(tokenizer.word_index) + 1
train_texts = tokenizer.texts_to_sequences(train_texts)

MAX_LENGTH = max(len(train_ex) for train_ex in train_texts)

train_texts = pad_sequences(train_texts, maxlen=MAX_LENGTH, padding="post")

model = models.Sequential()
model.add(layers.Input(shape=(MAX_LENGTH,)))
model.add(layers.Embedding(input_dim=vocab_size, output_dim=64))
model.add(layers.Conv1D(filters=10, kernel_size=3, activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])


model.summary()
```

Section 3

Exercise

Exercise 09

https://github.com/idh-cologne-deep-learning/exercise-09