Applied Deep Learning for NLP

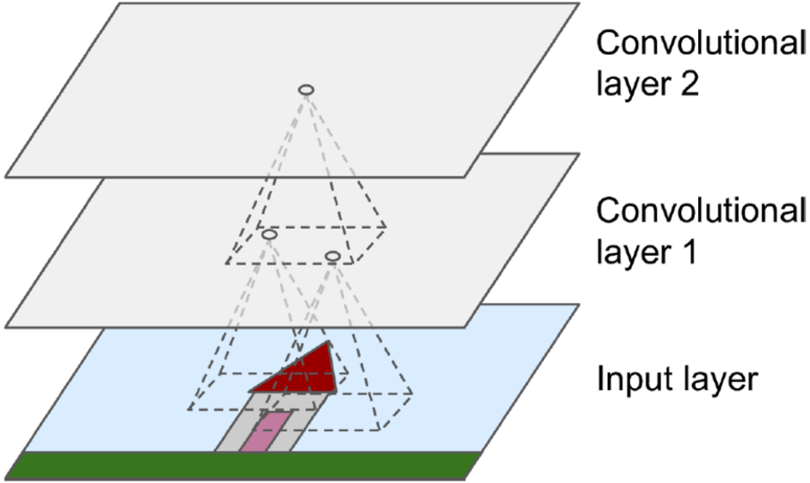
Week 10 - CNN & Autoencoders

Juan Carlos Medina Serrano

Technische Universität München
Hochschule für Politik
Political Data Science

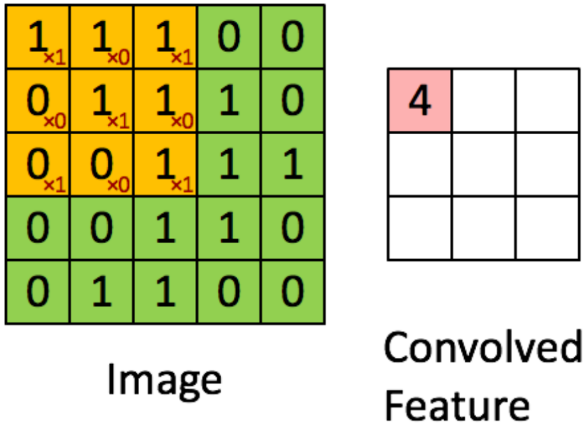
Munich, 07. January 2021

CNNs



CNNs

Discrete convolutions using kernel filters of size k



Main idea for text: Compute vectors for every possible word subsequence of a certain length. And then group them.

1D Convolution for Text

the	0.2	0.1	-0.3	0.4
red	0.5	0.2	-0.3	-0.1
brown	-0.1	-0.3	-0.2	0.4
fox	0.3	-0.3	0.1	0.1
jumped	0.2	-0.3	0.4	0.2
over	0.1	0.2	-0.1	-0.1

3	1	2	-3
-1	2	1	-3
1	1	-1	1

t,r,b	-1
r,b,f	-0.5
b,f,j	-3.6
f,j,o	-0.2

Padding

0	0.0	0.0	0.0	0.0
the	0.2	0.1	-0.3	0.4
red	0.5	0.2	-0.3	-0.1
brown	-0.1	-0.3	-0.2	0.4
fox	0.3	-0.3	0.1	0.1
jumped	0.2	-0.3	0.4	0.2
over	0.1	0.2	-0.1	-0.1
0	0.0	0.0	0.0	0.0

0,t,r	-0.6
t,r,b	-1
r,b,f	-0.5
b,f,j	-3.6
f,j,o	-0.2
j,o,0	-0.3

3 Channel 1D Convolution

3	1	2	-3
-1	2	1	-3
1	1	-1	1

1	0	0	1
1	0	-1	-1
0	1	0	1

1	-1	2	-1
1	0	-1	3
0	2	2	1

0,t,r	-0.6	0.2	1.4
t,r,b	-1	1.6	-1.0
r,b,f	-0.5	-0.1	0.8
b,f,j	-3.6	0.3	0.3
f,j,o	-0.2	0.1	1.2
j,o,0	0.3	0.6	0.7

Max Pooling

3	1	2	-3
-1	2	1	-3
1	1	-1	1

1	0	0	1
1	0	-1	-1
0	1	0	1

1	-1	2	-1
1	0	-1	3
0	2	2	1

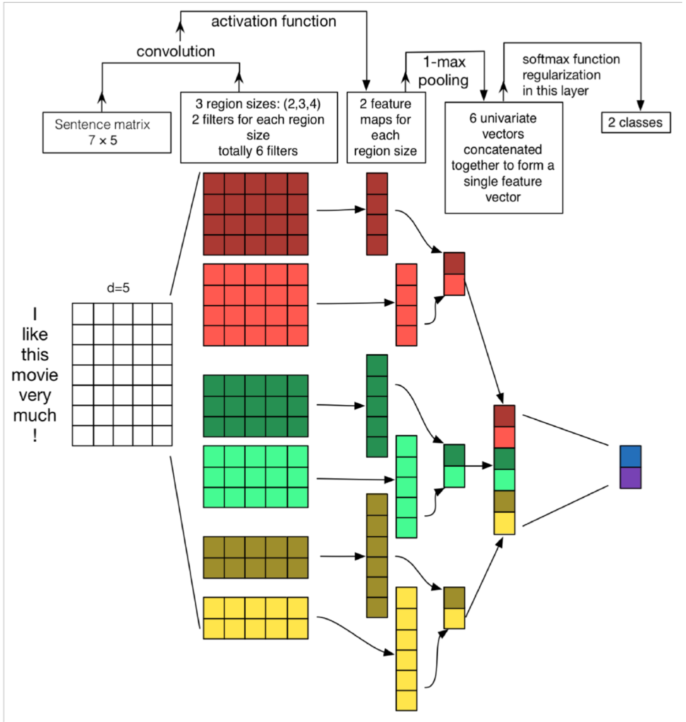
0,t,r	-0.6	0.2	1.4
t,r,b	-1	1.6	-1.0
r,b,f	-0.5	-0.1	0.8
b,f,j	-3.6	0.3	0.3
f,j,o	-0.2	0.1	1.2
j,o,0	0.3	0.6	0.7

max pooling	0.3	1.6	1.4

Dilation

0,t,r	-0.6	0.2	1.4
t,r,b	-1	1.6	-1.0
r,b,f	-0.5	-0.1	0.8
b,f,j	-3.6	0.3	0.3
f,j,o	-0.2	0.1	1.2
j,o,0	0.3	0.6	0.7

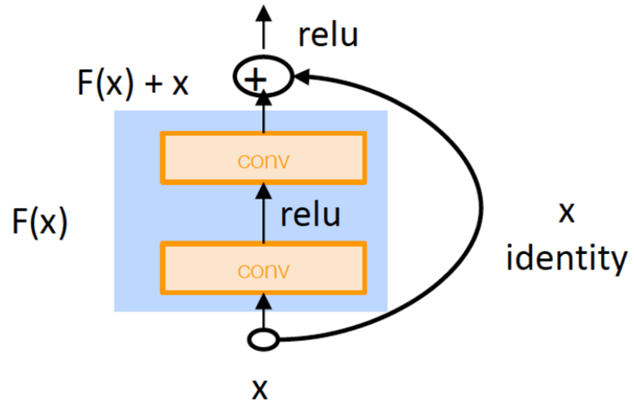
Example CNN for Classification



https://arxiv.org/pdf/1510.03820.pdf

Residual Blocks

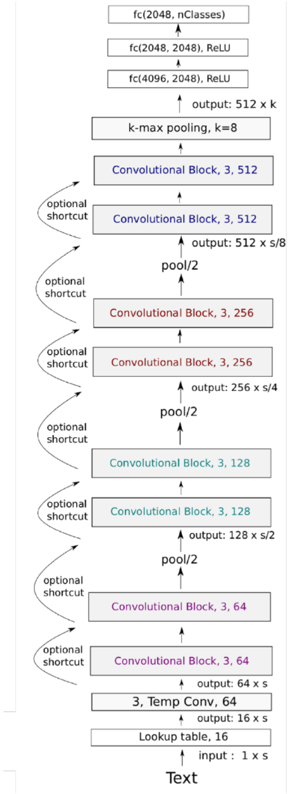
Residual blocks were crucial for deep deep CNNs

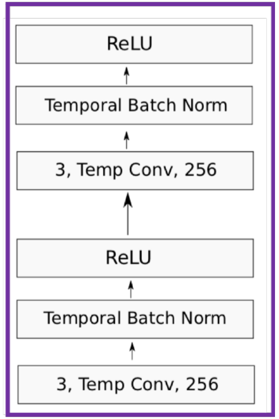


Character Level CNNs

Common in practice to use character embeddings, and use CNNs to generate word embeddings

Deep Text CNN



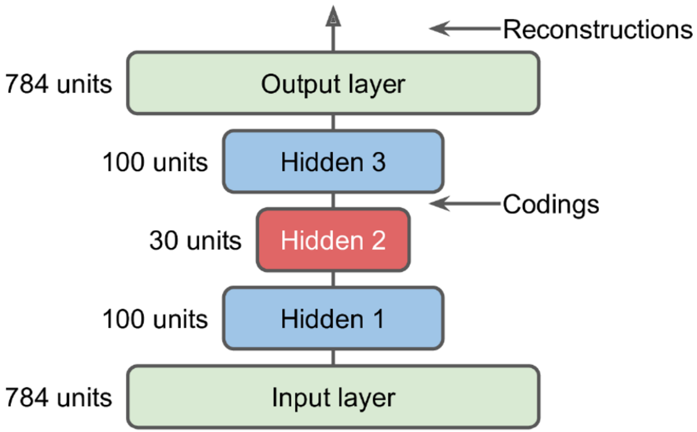


CNN on Tensorflow

```
cnnmodel = Sequential()
cnnmodel.add(embedding_layer)
cnnmodel.add(Conv1D(128, 5, activation='relu'))
cnnmodel.add(MaxPooling1D(5))
cnnmodel.add(Conv1D(128, 5, activation='relu'))
cnnmodel.add(MaxPooling1D(5))
cnnmodel.add(Conv1D(128, 5, activation='relu'))
cnnmodel.add(GlobalMaxPooling1D())
cnnmodel.add(Dense(128, activation='relu'))
cnnmodel.add(Dense(len(labels_index), activation='softmax'))
cnnmodel.compile(loss='categorical_crossentropy',
                 optimizer='rmsprop',
                 metrics=['acc'])
cnnmodel.fit(x_train, y_train,
```

Stacked Autoencoders

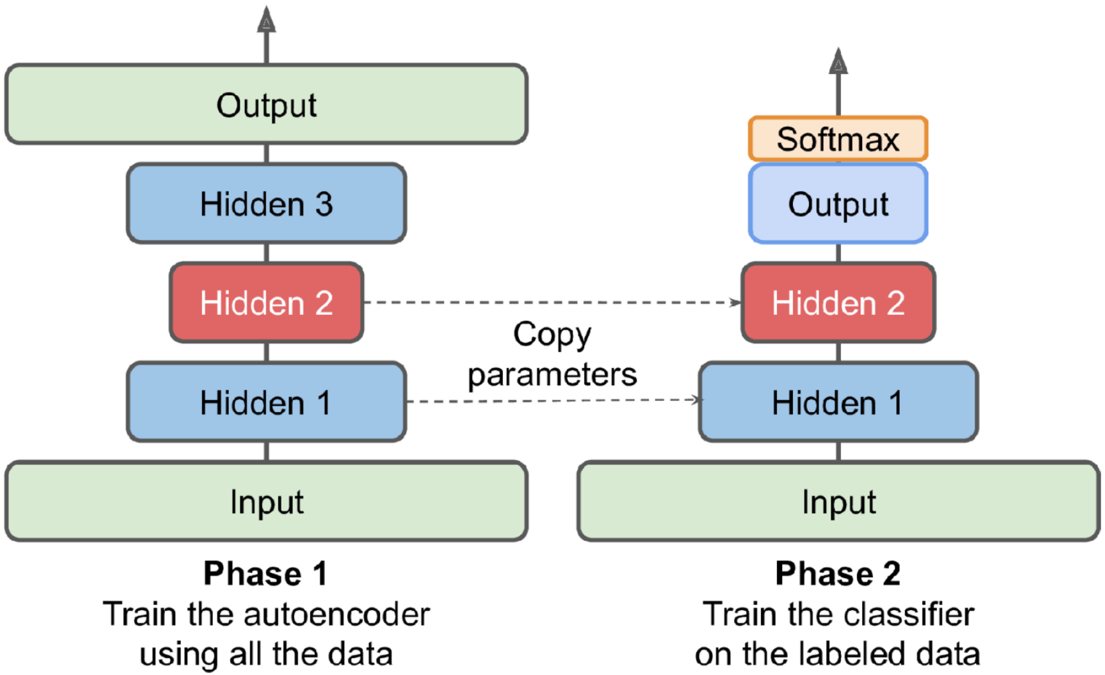
The **Encoder** learn a compressed representation of the input. The **Decoder** reconstructs the input.



Learning a low-dimensional representation of the data similar to PCA

Unsupervised Pre-training

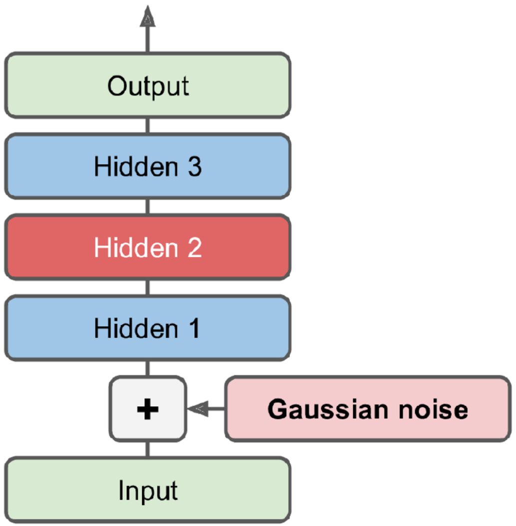
After training, the Decoder is discarded.



Recurrent Autoencoder on Tensorflow

```
recurrent_encoder = keras.models.Sequential([
    keras.layers.LSTM(100, return_sequences=True, input_shape=[None, 28]),
    keras.layers.LSTM(30)
])
recurrent_decoder = keras.models.Sequential([
    keras.layers.RepeatVector(28, input_shape=[30]),
    keras.layers.LSTM(100, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(28, activation="sigmoid"))
])
recurrent_ae = keras.models.Sequential([recurrent_encoder, recurrent_decoder])
```

Denoising Autoencoders



BERT and BART are considered to have a Denoising Autoencoder architecture. The noise is introduced by the [MASK] tokens.

Sparse Autoencoders

Simple reconstruction does not ensure that the network will learn abstract features from the dataset. We need to add further constraints.

Ensure that fewer units in the inner layer are activated -> **Sparsity**

Option 1: Using L1 regularization in the inner layer.

Option 2: Add a **sparsity loss** to the cost function. Which one? **Kullback-Leibler** divergence

$$D_{KL}(P||Q) = \sum_i P(i) * log \frac{P(i)}{Q(i)}$$

In our case the divergence between target probability **p** that a neuron will activate and the actual mean probability **q**.

$$D_{KL}(p||q) = p * log \frac{p}{q} + (1-p) log \frac{1-p}{1-q}$$

Overcomplete Autoencoders

Denoising and Sparse Autoencoders work better as **overcomplete** autoencoders: The inner layer has larger dimension than the input/output layer

Remember the pointwise FNN layer from the Transformer? It was an overcomplete autoencoder!