# Statistical Natural Language Processing

## Recurrent and convolutional networks

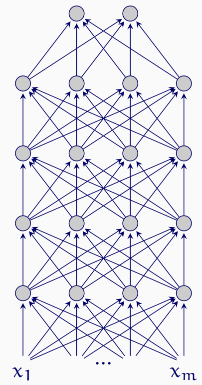
Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2019



## Deep neural networks




- Deep neural networks (>2 hidden layers) have recently been successful in many tasks
- They often use sparse connectivity and shared weights
- We will focus on two important architectures: recurrent and convolutional networks





## Why deep networks?

- We saw that a neural network with a single hidden layer is a *universal approximator*
- However, this is a theoretical result – it is not clear how many units one may need for the approximation
- Successive layer may learn different representations
- Deeper architectures have been found to be useful in many tasks





## Why now?

- Increased computational power, especially advances in graphical processing unit (GPU) hardware
- Availability of large amounts of data
  - mainly unlabeled data (more on this later)
  - but also labeled data through 'crowd sourcing' and other sources
- Some new developments in theory and applications





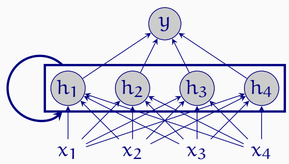
## Recurrent neural networks

- Feed forward networks
  - can only learn associations
  - do not have memory of earlier inputs: they cannot handle sequences
- Recurrent neural networks are ANN solution for sequence learning
- This is achieved by recursive loops in the network





## Recurrent neural networks



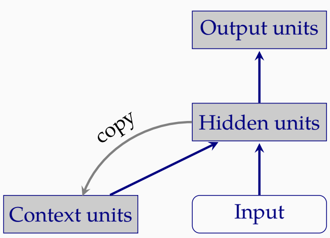

- Recurrent neural networks are similar to the standard feed-forward networks
- They include loops that use previous output (of the hidden layers) as well as the input
- Forward calculation is straightforward, learning becomes somewhat tricky





## A simple version: SRNs

Elman (1990)



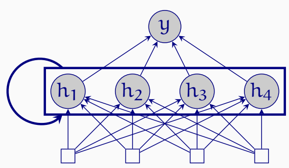

- The network keeps previous hidden states (context units)
- The rest is just like a feed-forward network
- Training is simple, but cannot learn long-distance dependencies



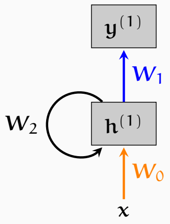


## Processing sequences with RNNs

- RNNs process sequences one unit at a time
- The earlier input affects the output through the recurrent links



## Learning in recurrent networks



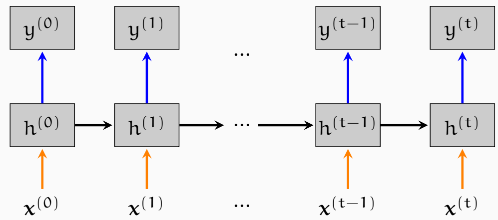

- We need to learn three sets of weights: $W_0$, $W_1$ and $W_2$
- Backpropagation in RNNs are at first not that obvious
- The main difficulty is in propagating the error through the recurrent connections





## Unrolling a recurrent network

Back propagation through time (BPTT)




Note: the weights with the same color are shared.





## Unstable gradients

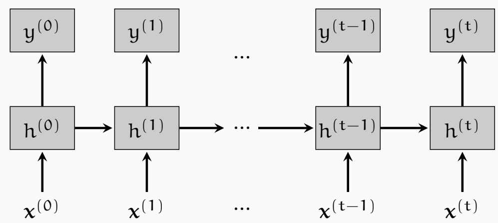
- A common problem in deep networks is *unstable gradients*
- The patial derivatives with respect to weights in the early layers calculated using the chain rule
- A long chain of multiplications may result in
  - *vanishing gradients* if the values are in range $(-1, 1)$
  - *exploding gradients* if absolute values larger than 1
- A practical solution for exploding gradients is called *gradient clipping*
- Solution to vanishing gradients is more involved (coming soon)





## RNN architectures

Many-to-many (e.g., POS tagging)

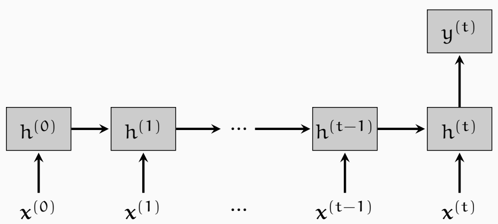







## RNN architectures

Many-to-one (e.g., document classification)

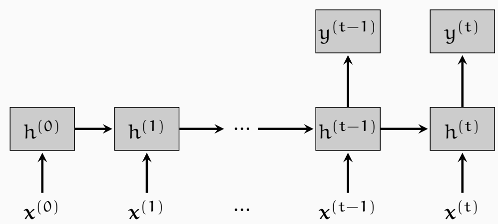







## RNN architectures

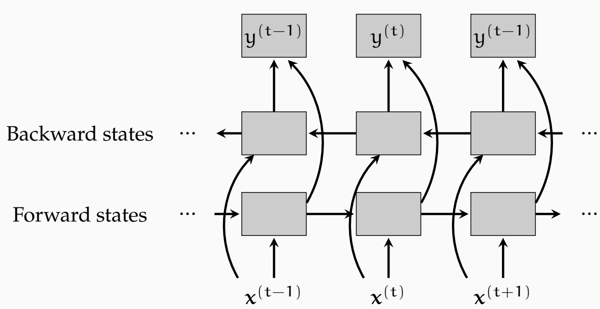
Many-to-one with a delay (e.g., machine translation)








## Bidirectional RNNs



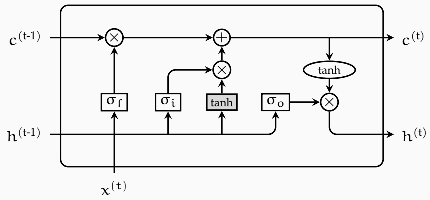





## Unstable gradients revisited

- We noted earlier that the gradients may *vanish* or *explode* during backpropagation in deep networks
- This is especially problematic for RNNs since the effective dept of the network can be extremely large
- Although RNNs can theoretically learn long-distance dependencies, this is affected by unstable gradients problem
- The most popular solution is to use *gated* recurrent networks

## Gated recurrent networks




- Most modern RNN architectures are 'gated'
- The main idea is learning a mask that controls what to remember (or forget) from previous hidden layers
- Two popular architectures are
  - Long short term memory (LSTM) networks (above)
  - Gated recurrent units (GRU)





## Convolutional networks

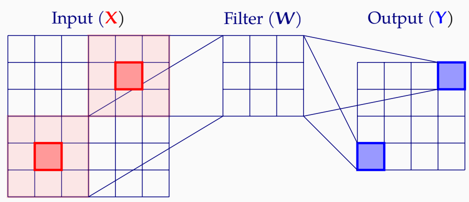
- Convolutional networks are particularly popular in image processing applications
- They have also been used with success some NLP tasks
- Unlike feed-forward networks we have discussed so far,
  - CNNs are not fully connected
  - The hidden layer(s) receive input from only a set of neighboring units
  - Some weights are shared
- A CNN learns features that are *location invariant*
- CNNs are also computationally less expensive compared to fully connected networks





## Convolution in image processing

- Convolution is a common operation in image processing for effects like edge detection, blurring, sharpening, ...
- The idea is to transform each pixel with a function of the local neighborhood




$$y = \sum_i w_i x_i$$





## Example convolutions

- Blurring

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$





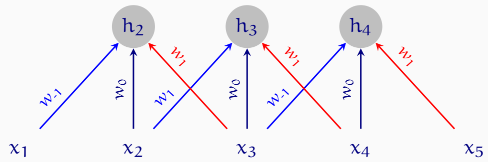
## Learning convolutions

- Some filters produce features that are useful for classification (e.g., of images, or sentences)
- In machine learning we want to *learn* the convolutions
- Typically, we learn multiple convolutions, each resulting in a different feature map
- Repeated application of convolutions allow learning higher level features
- The last layer is typically a standard fully-connected classifier





## Convolution in neural networks

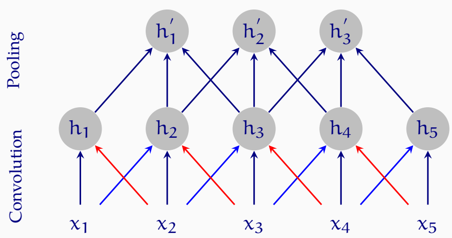



- Each hidden layer corresponds to a local window in the input
- Weights are shared: each convolution detects the same type of features





## Pooling

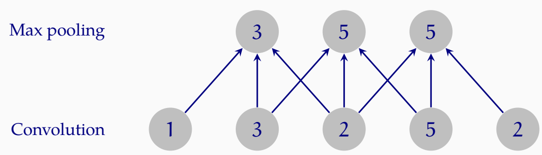



- Convolution is combined with *pooling*
- Pooling 'layer' simply calculates a statistic (e.g., max) over the convolution layer
- Location invariance comes from pooling





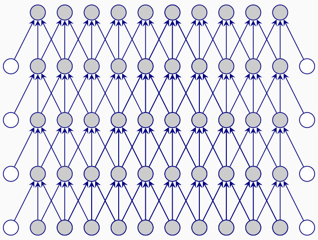
## Pooling and location invariance




- Note that the numbers at the pooling layer are stable in comparison to the convolution layer

## Padding in CNNs

- With successive layers of convolution and pooling, the size of the later layers shrinks
- One way to avoid this is *padding* the input and hidden layers with enough number of zeros

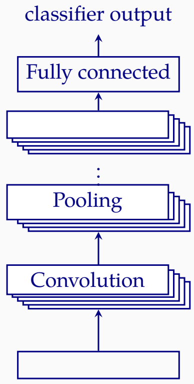






## CNNs: the bigger picture

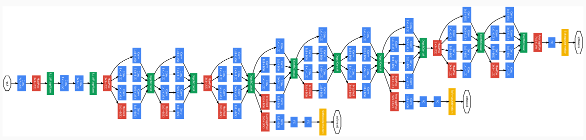
- At each convolution/pooling step, we often want to learn multiple feature maps
- After a (long) chain of hierarchical features maps, the final layer is typically a fully-connected layer (e.g., softmax for classification)








## Real-world examples are complex



The real-world ANNs tend to be complex

- Many layers (sometimes with repetition)
- Large amount of branching

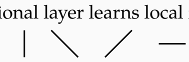
* Diagram describes an image classification network, GoogLeNet (Szegedy et al. 2014).



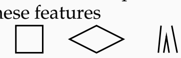


## CNNs in natural language processing

- The use of CNNs in image applications is rather intiutive
  - the first convolutional layer learns local features, e.g., edges



  - successive layers learn more complex features that are combinations of these features

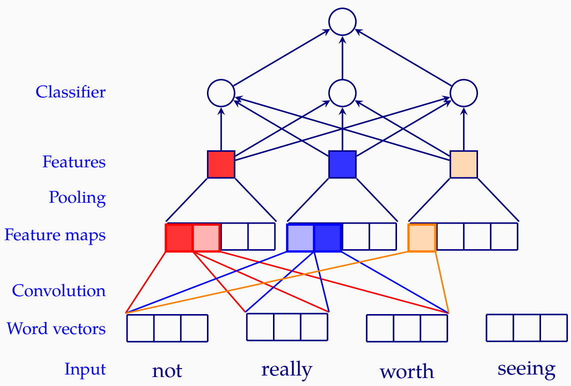


- In NLP, it is a bit less straight-forward
  - CNNs are typically used in combination with word vectors
  - The convolutions of different sizes correspond to (word) n-grams of different sizes
  - Pooling picks important 'n-grams' as features for classification





## An example: sentiment analysis








## Summary

- Deep networks use more than one hidden layer
- Common (deep) ANN architectures include:
  - CNN location invariance
  - RNN sequence learning

Next:

Wed lab

Fri N-gram language models