Universität Potsdam

Deep Learning for NLP

Convolution & Recurrence

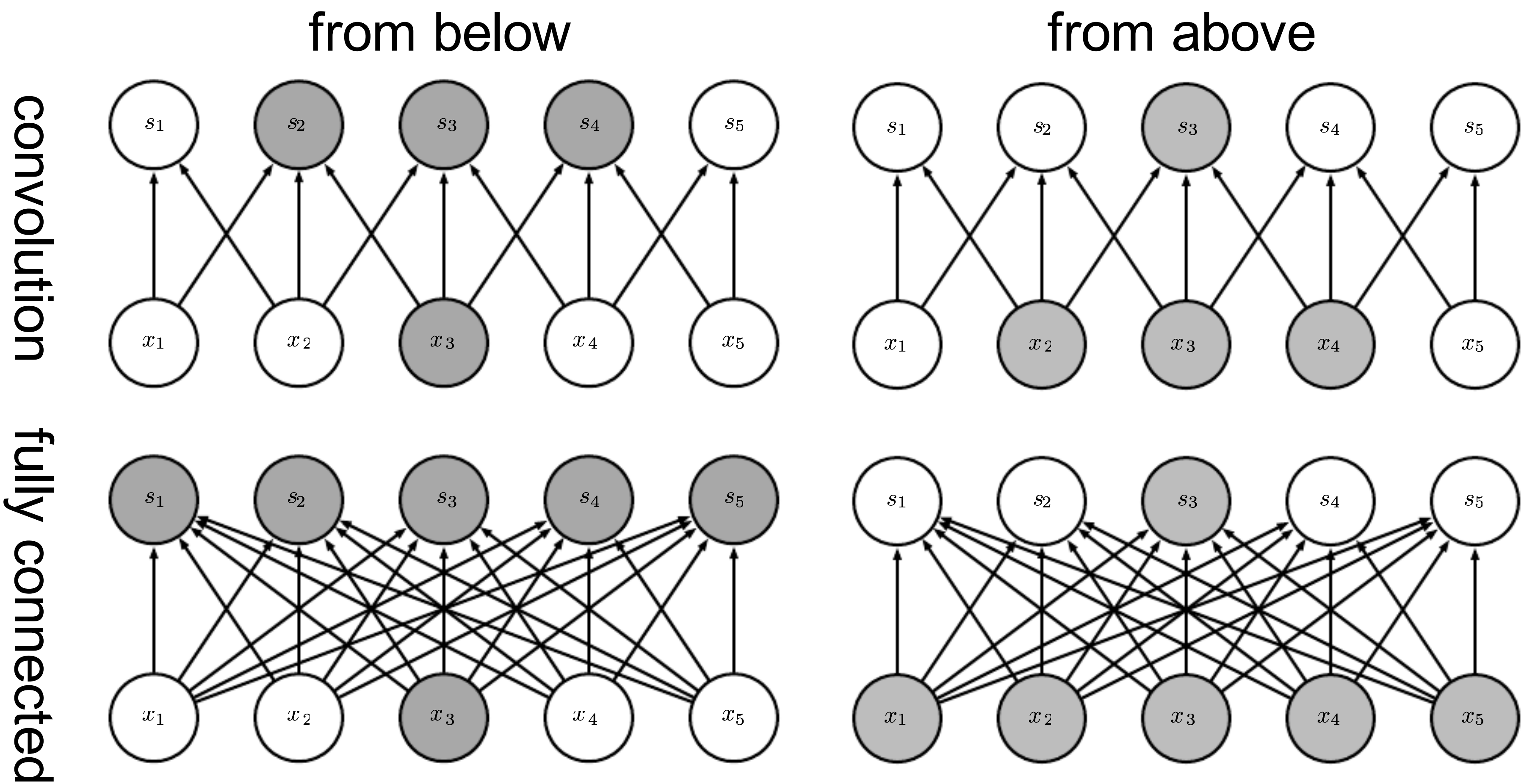
April 29, 2016

Sebastian Stober <sstober@uni-potsdam.de>

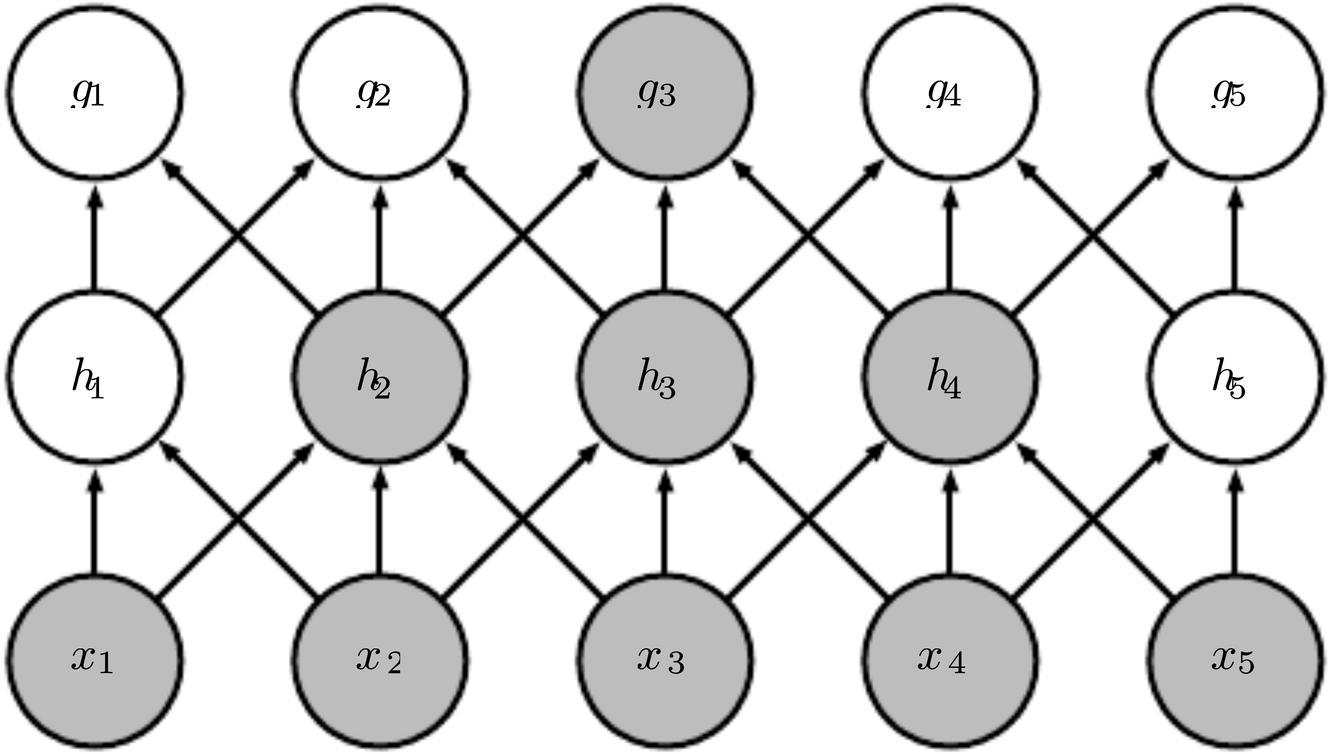
*with figures from deeplearningbook.org

Convolution

Sparse Connectivity

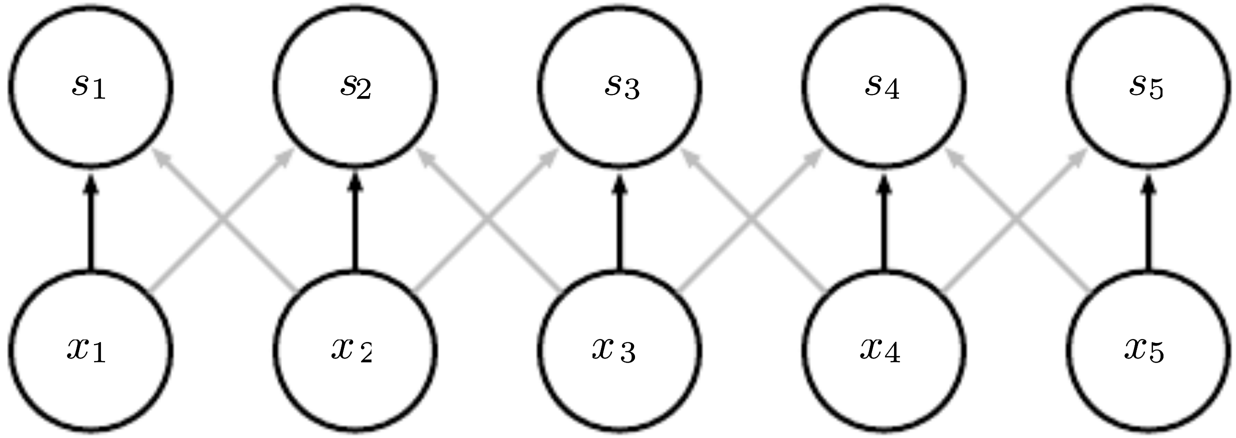


Receptive Field

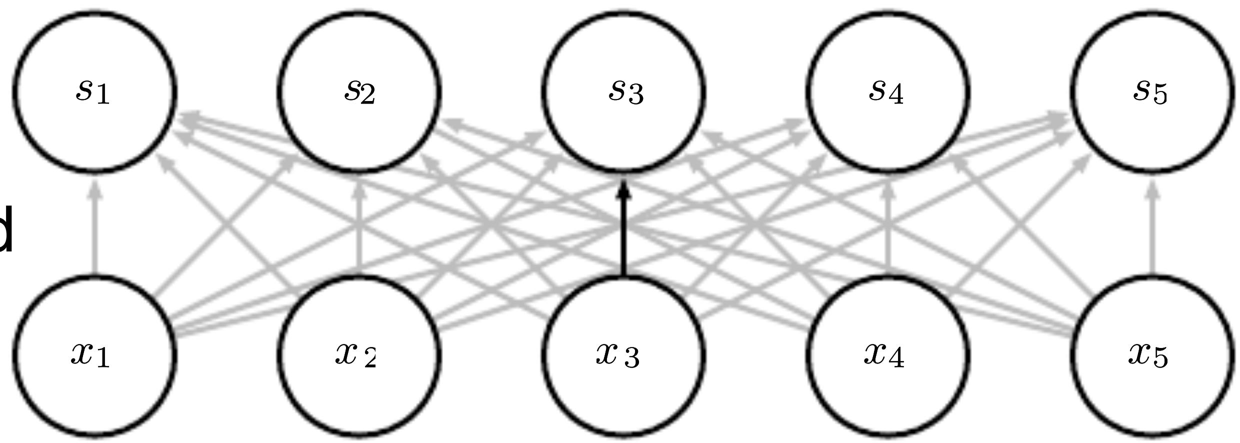


Parameter Sharing

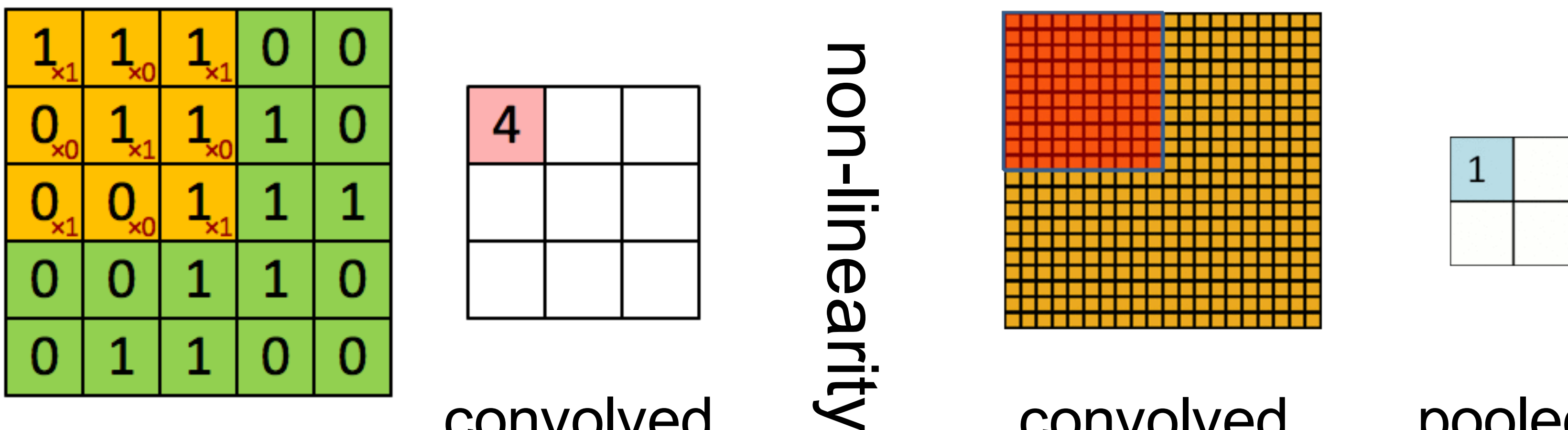
convolution



fully connected



Convolution & Pooling



2D input

convolved feature

convolved feature

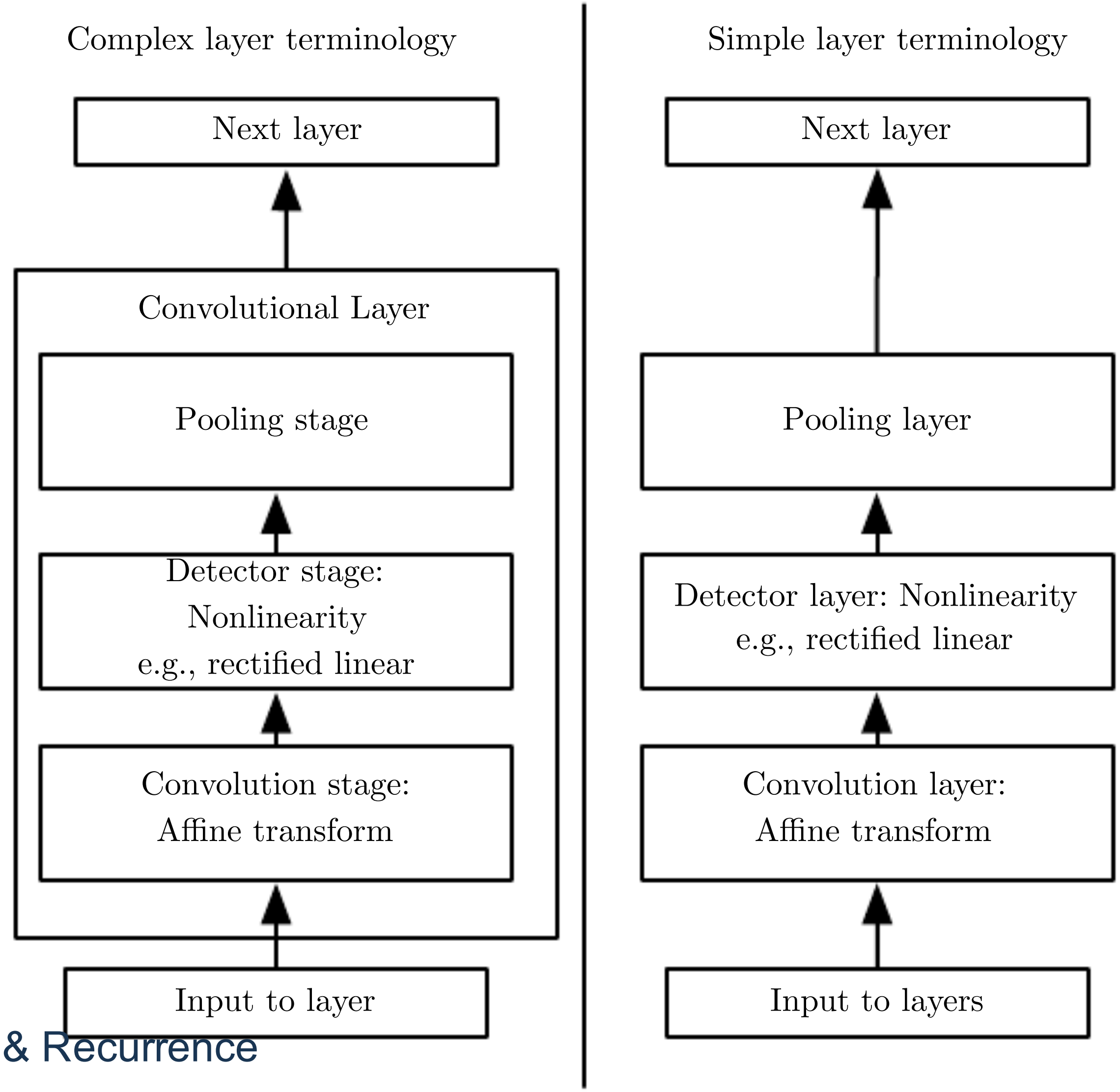
pooled feature

[http://ufldl.stanford.edu/wiki/i]

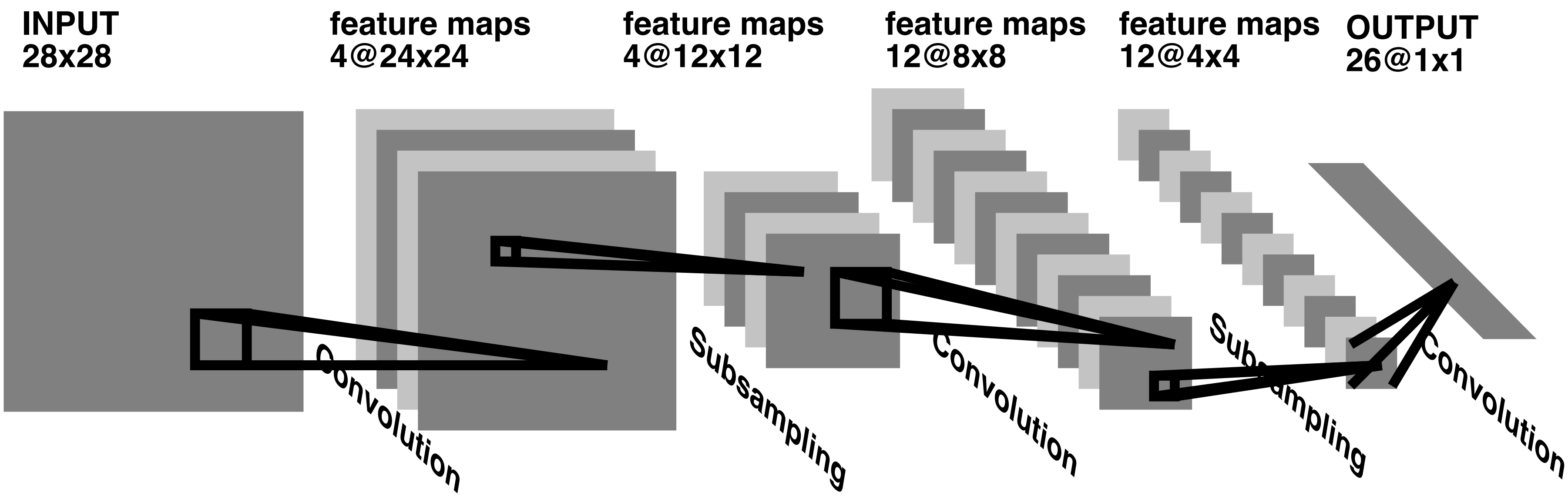
Convolution & Pooling

- convolution
  - equivariance: if the input changes, the output changes in the same way
- pooling
  - approximate invariance to small translations
  - trade-off: whether? vs. where?
  - special case: maxout-pooling (pooling over several filters => learn invariance)

Complex vs. Simple Layer Structure

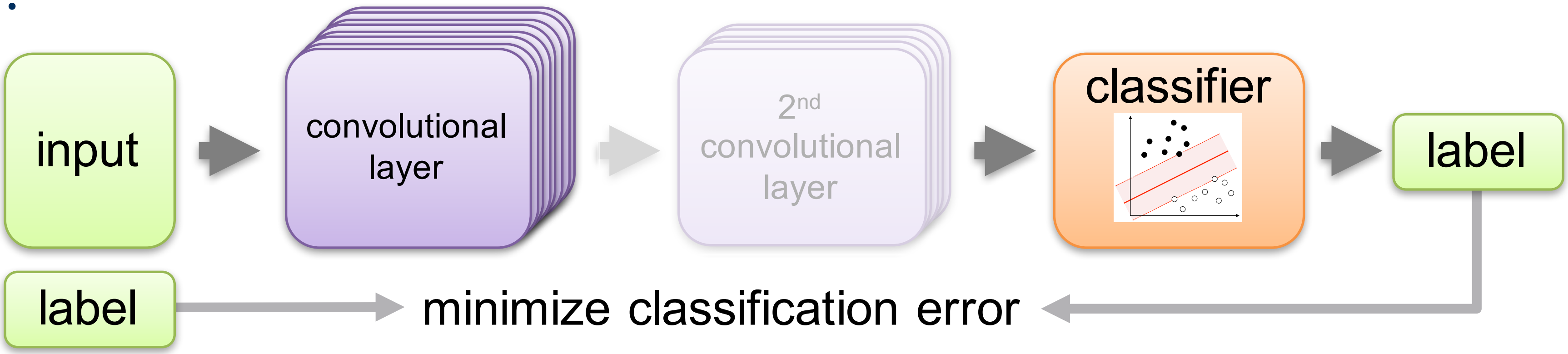


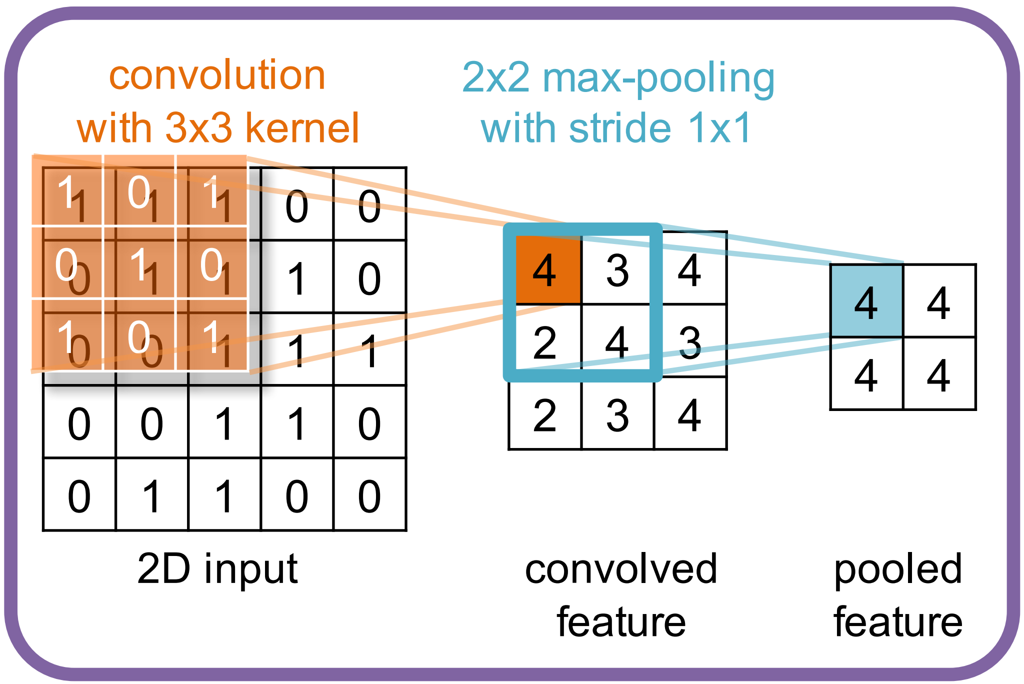
CNN (simple layers)



[Y. Bengio and Y. Lecun, 1995]

CNN (complex layers)



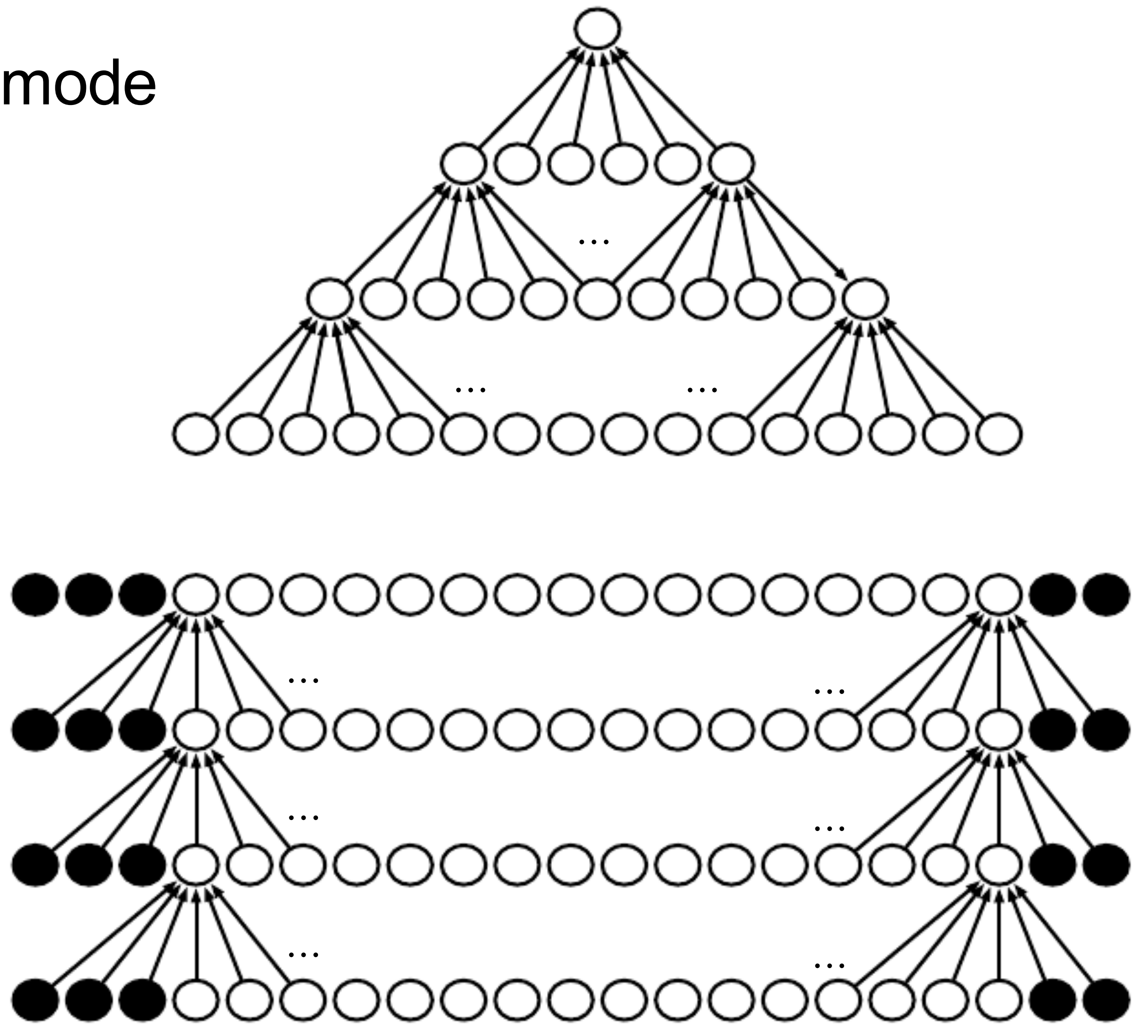


- involves non-linear transform (activation function) after conv.
- pool size controls amount of invariance to input translations
- pool stride (step size) controls non-linear sub-sampling

To Pad or Not to Pad?

see convolution mode in Blocks:

- valid
- same
- full



Down-Sampling/Stride

- reduces dimensionality

- stride > 1 for convolution
- down-sampling in combination with pooling

Strong Priors

- CNN = “fully connected net with an infinitely strong prior [on weights]”
- only useful when the assumptions made by the prior are reasonably accurate
- convolution+pooling can cause underfitting

Recurrence

Motivation

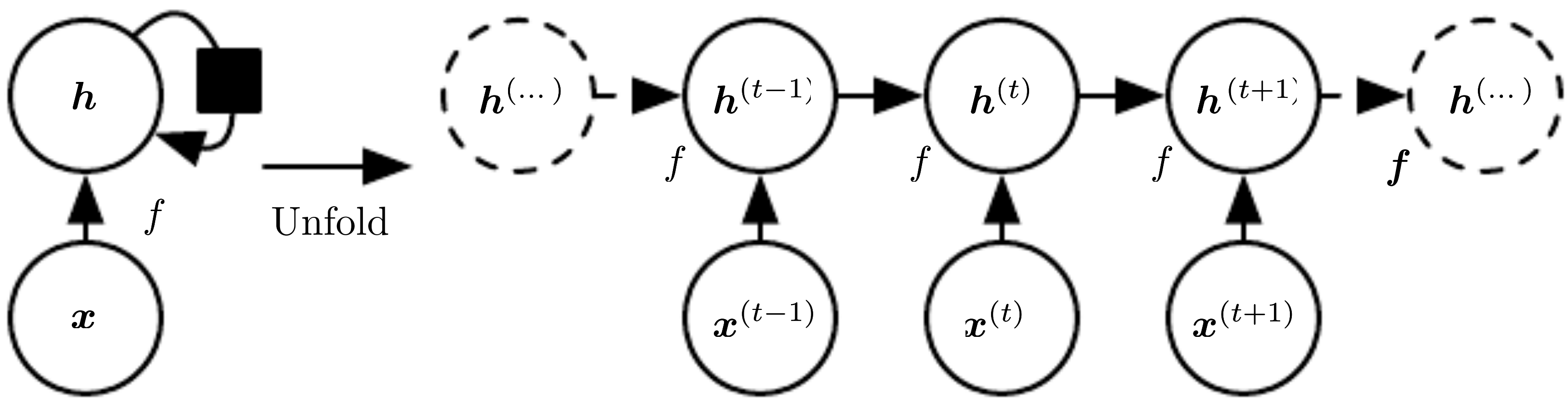
- process sequential data
- capture history of inputs/states
- share parameters through a very deep computational graph
  - output is a function of the previous output
  - produced using the same update rule applied to the previous outputs.
- different from convolution across time steps

Cyclic Connections

- computational graph includes cycles (recursion)
- represent influence of the present value of a variable on its own value at future time steps

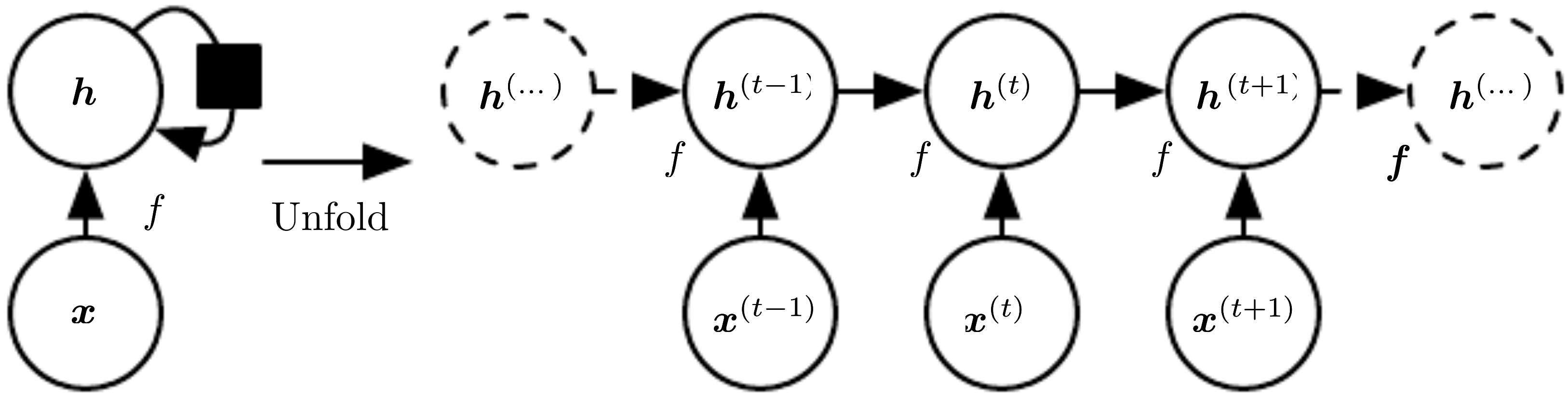
- unfolding to yield a graph that does not involve recurrence
  => gets very deep very quickly

Unfolding



1. regardless of the sequence length, the learned model always has the same input dimensionality
2. can use the same transition function f with the same parameters at every time step

Back-Propagation Through Time (BPTT)



- gradient computation for unfolded loss function w.r.t parameters very expensive
- O(T) where T is history length
- no parallelization (sequential dependence)

Dynamic System

- network now contains information about the whole past sequence:
  - inputs,
  - states,
  - outputs

Hidden State

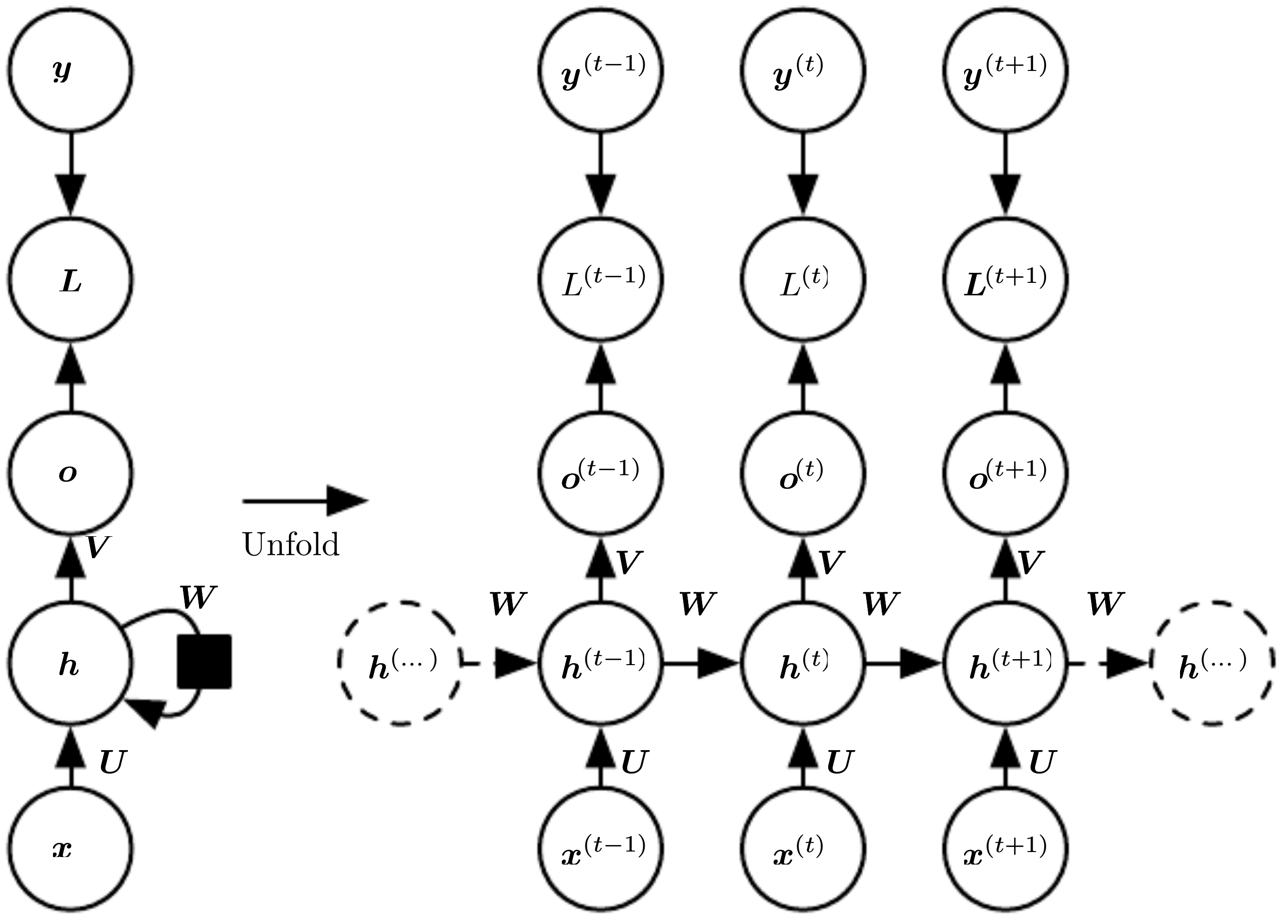
- h(t) as a kind of “lossy summary” of the task-relevant aspects of the history up to t
  - lossy compression necessary
  - selectivity based on training criterion (cost)

- most demanding situation: rich enough representation h(t) to allow approximate recovery of input sequences (autoencoder)

Design Patterns

1. output at each time step,
   recurrent connections between hidden units
2. output at each time step
   recurrent connections only from output at one time step to hidden units at next step
3. single output for entire input sequence,
   recurrent connections between hidden units

1. output at each time step, rec. conn. between hidden units



Such a recurrent network of with finite size can compute any function computable by a Turing machine. Training???

Gradient Clipping