Neural Networks Part II

STAT 37710 / CMSC 35400
Rebecca Willett and Yuxin Chen

Overfitting

Overfitting

Two approaches are often used to help avoid overfitting in neural networks:

a) **Early stopping**: train the model for a while, but stop iterating *before* convergence
Since we initialize *near* a linear model, the effect of this is to prefer learned models that are closer to being linear

b) **Weight decay**: (analogous to ridge regression for linear models) Instead of minimizing $\frac{1}{n}\sum_{i=1}^{n} L_i(\theta)$, we

minimize $\frac{1}{n}\sum_{i=1}^{n} L_i(\theta) + \lambda \|\theta\|_2^2$

$\|\theta\|_2^2$ ← sum of squares of all weights

(Many other forms of regularization or penalization have been proposed, but currently weight decay is the most widely adopted.)

Under some conditions, it can be shown that early stopping is a form of weight decay

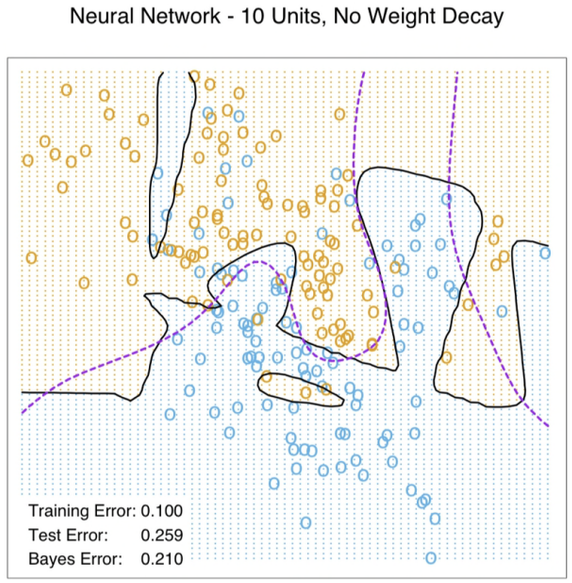
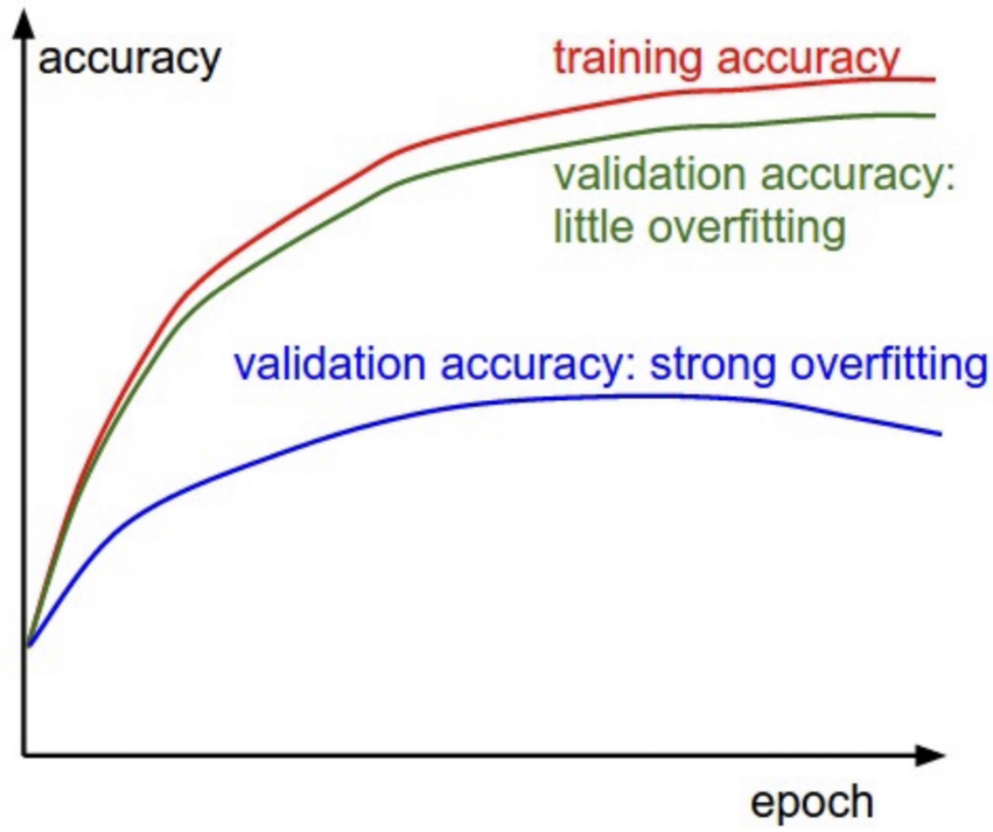


FIGURE 11.4. *A neural network on the mixture example of Chapter 2. The upper panel uses no weight decay, and overfits the training data. The lower panel uses weight decay, and achieves close to the Bayes error rate (broken purple boundary). Both use the softmax activation function and cross-entropy error.*

Overfitting Part II

Monitor Accuracy on Train/Validation During Training

- Check how your desired performance metrics behaves during training



[http://cs231n.github.io/neural-networks-3/]

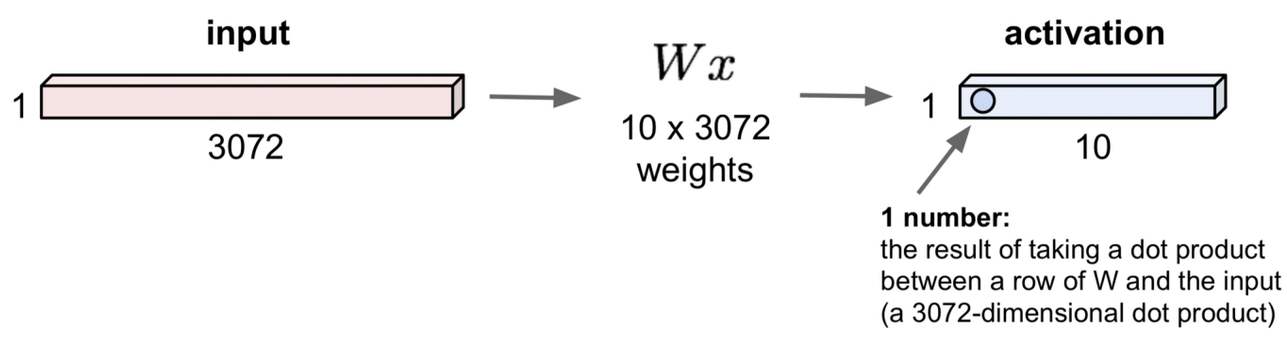
Convolutional Neural Nets

Convolutional Neural Networks

(images from http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf)

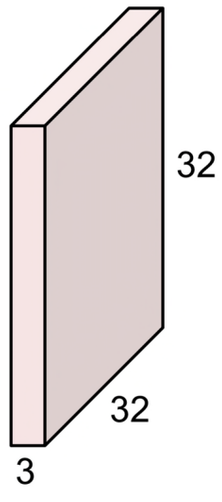
So far, we have only discussed fully-connected layers:

32x32x3 image -> stretch to 3072 x 1

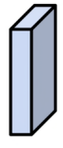


Now we will consider convolutional layers

32x32x3 image



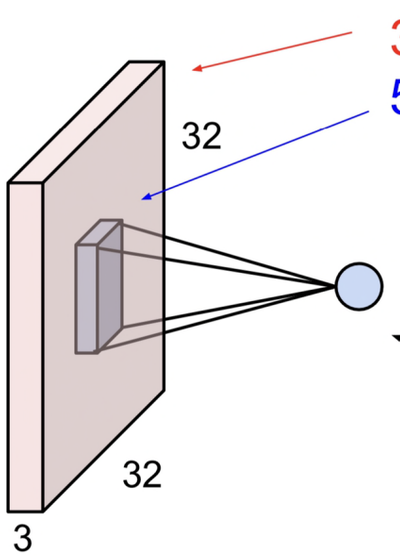
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution and activation

Convolution operation & activation maps:



32x32x3 image

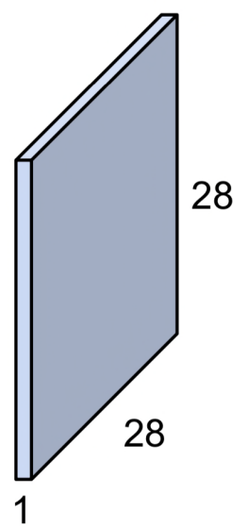
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

convolve (slide) over all spatial locations

activation map



Note: the mapping from the $32 \times 32 \times 3$ image to the $28 \times 28 \times 1$ activation map can be represented using convolution, as described above. Here we have $5 \times 5 \times 3 = 75$ weights to learn.

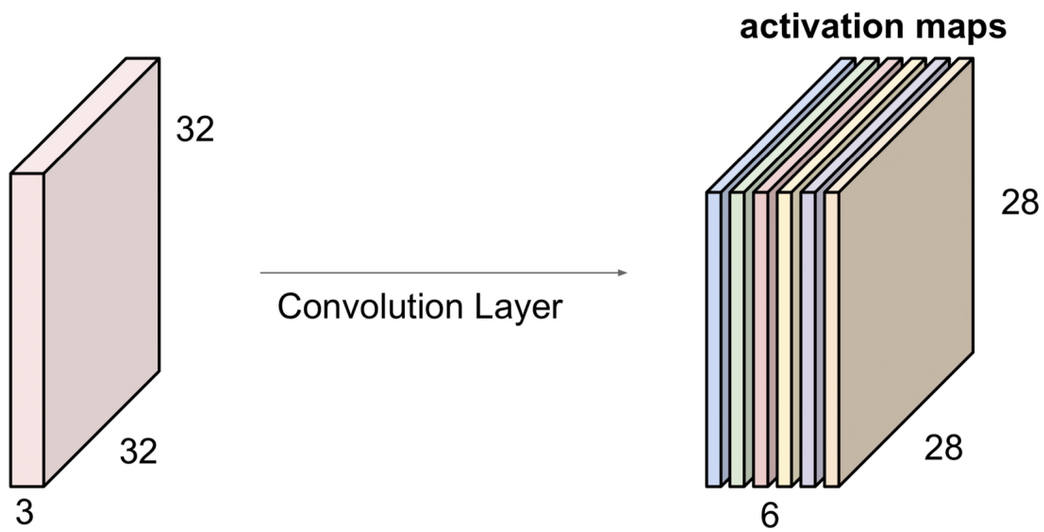
Equivalently, we could represent this mapping with a fully connected layer corresponding to a weight matrix W of size $\underbrace{784}_{28^2} \times \underbrace{3072}_{32^2 \cdot 3} = 2.4\text{M}$ weights

So essentially, we are learning a fully connected layer with 2.4M weights, but the convolutional structure constrains many of the weights to be zero and many of the non-zero weights to have the same value. These constraints ensure we don't have too many degrees of freedom and hence can learn good values from training data

Multiple Filters

We can repeat this for different filters:

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

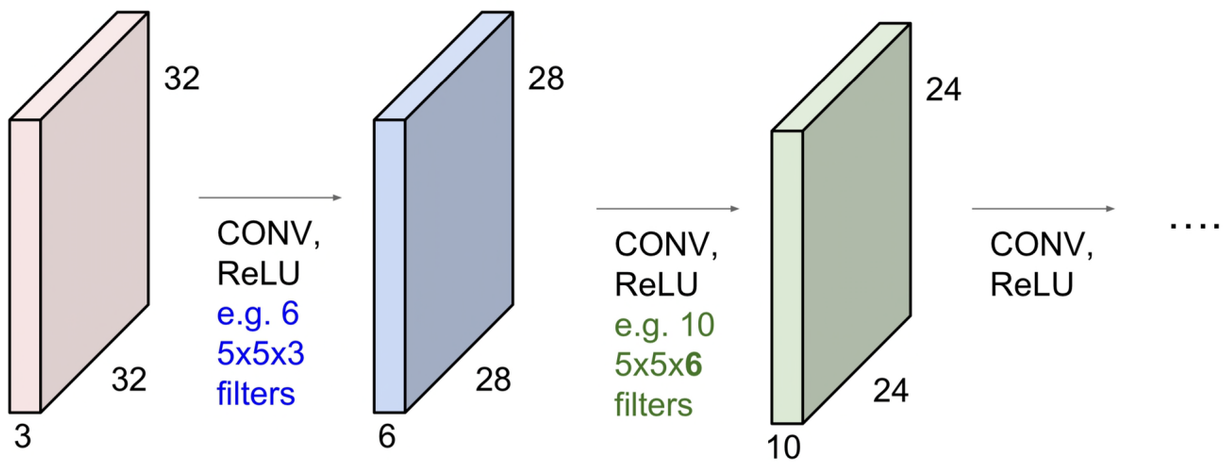


We stack these up to get a "new image" of size 28x28x6!

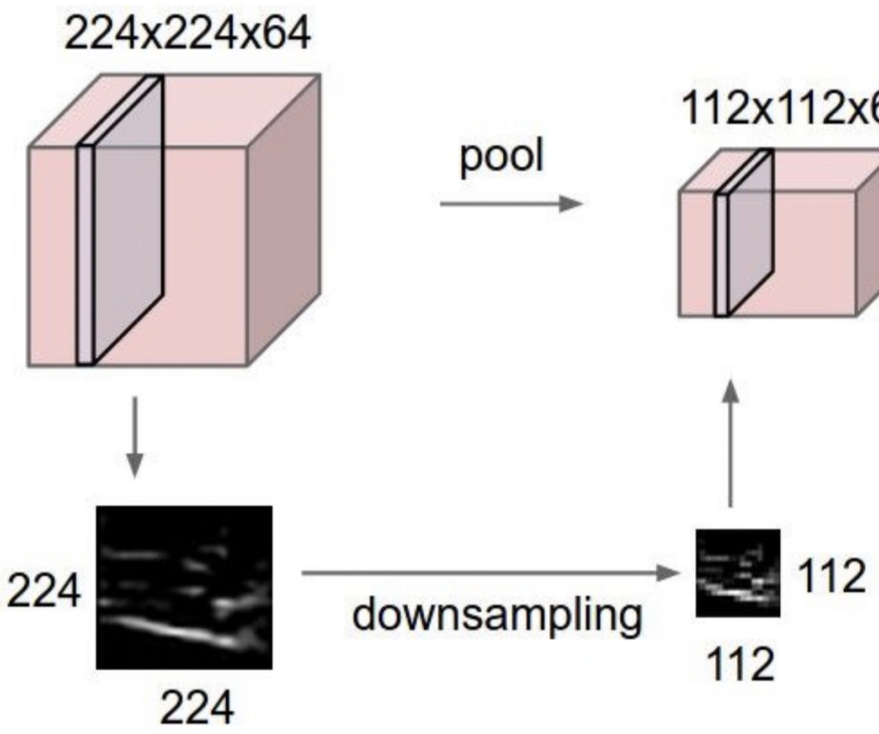
ConvNets and Pooling

ConvNets:

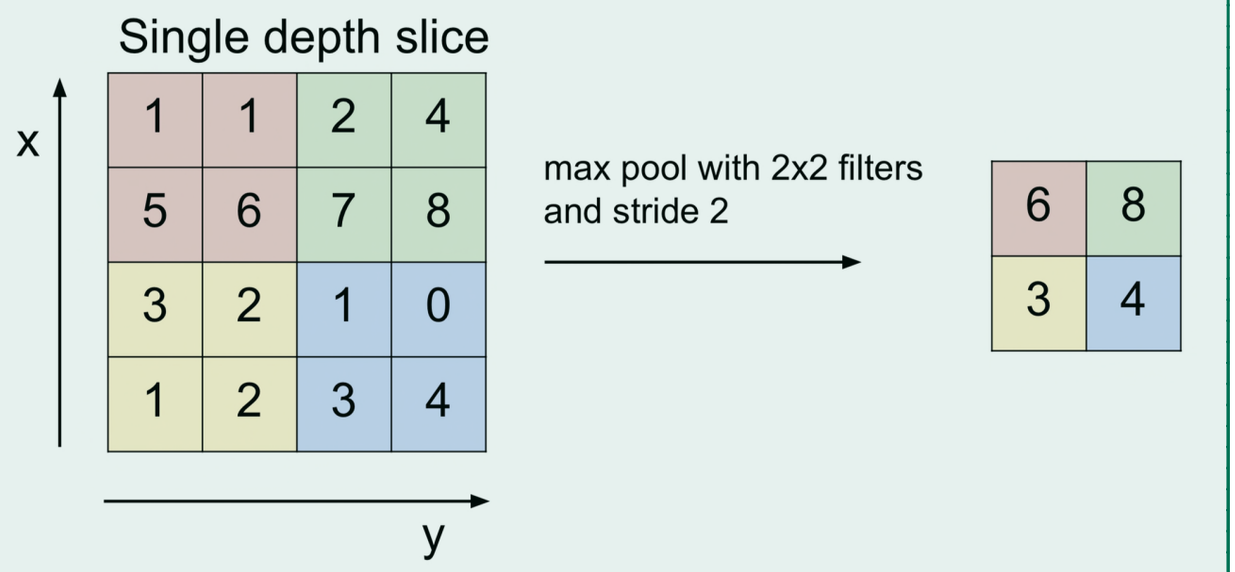
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



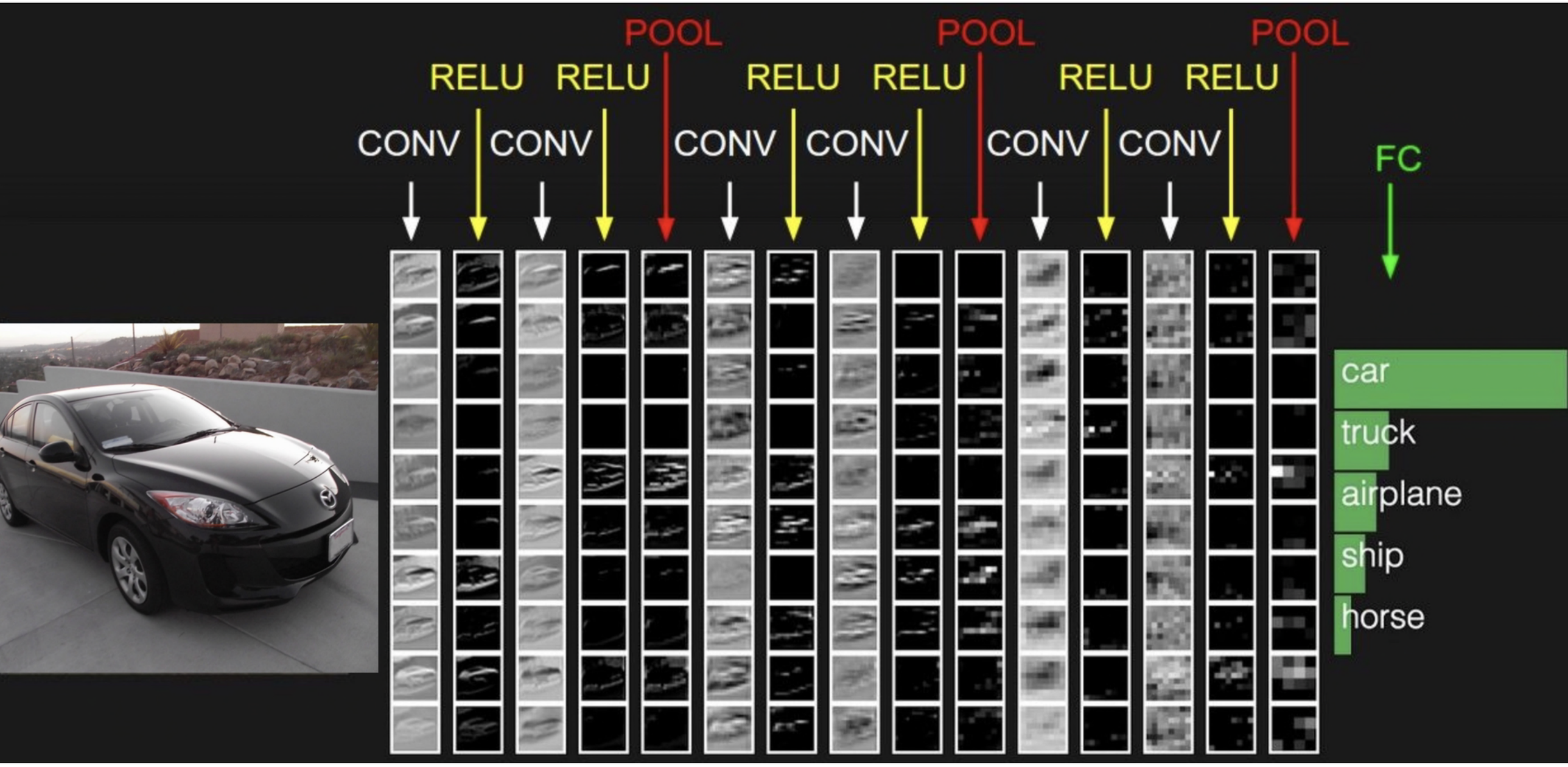
Modern CNNs also use a pooling step to keep the size of the networks manageable:



Example: MaxPooling



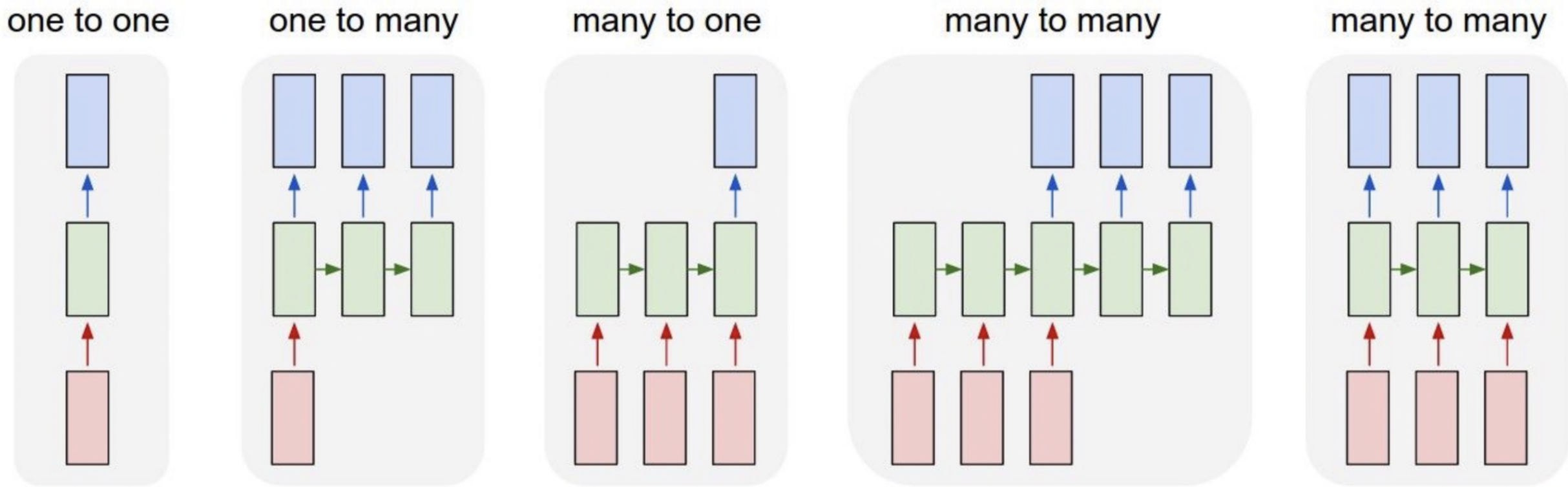
Example


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Neural networks for sequences

Recurrent neural networks for processing sequences

(images from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf)



our focus so far. e.g. 1 image → 1 label

e.g. image captioning: 1 image → sequence of words in caption

e.g. sentiment classification: sequence of words in sentence → 1 sentiment

e.g. Machine Translation: sequence of words to sequence of words

e.g. classification of each frame of a video

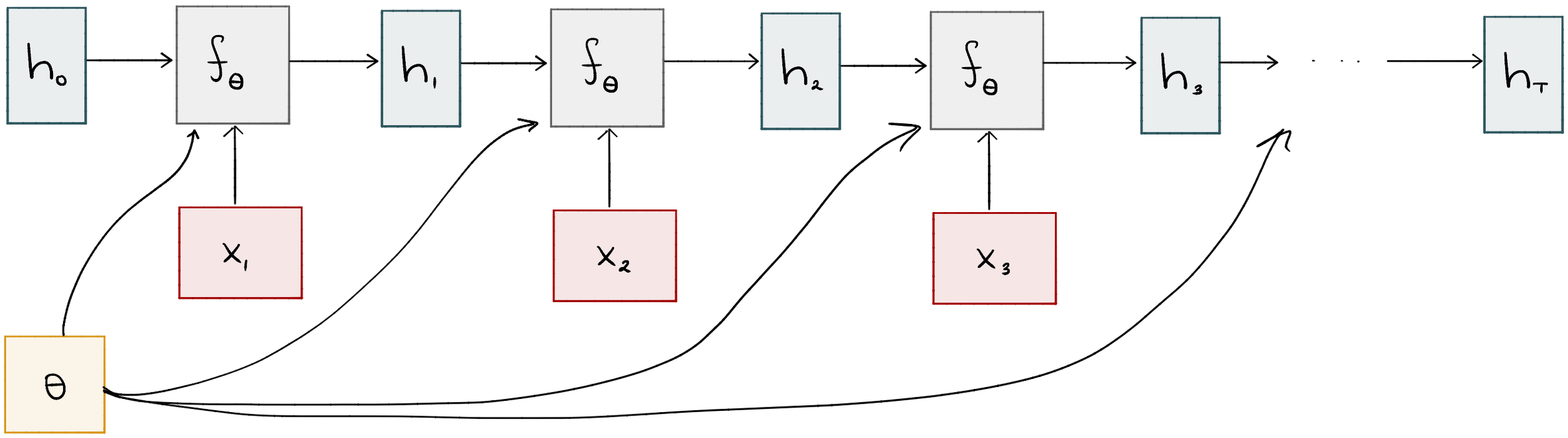
Hidden states

When processing sequence data, it's insufficient to only consider the current data or activations — we also need memory to capture everything that happened previously in the sequence.

This memory is represented by a hidden state, which we denote h_t for the t^{th} element of a sequence

Basic model for RNNs: $h_t = f_\theta(h_{t-1}, x_t)$ — the same f and θ are used at every time t

- h_t: next state
- f_θ: network weights
- h_{t-1}: previous state
- x_t: input vector at time t

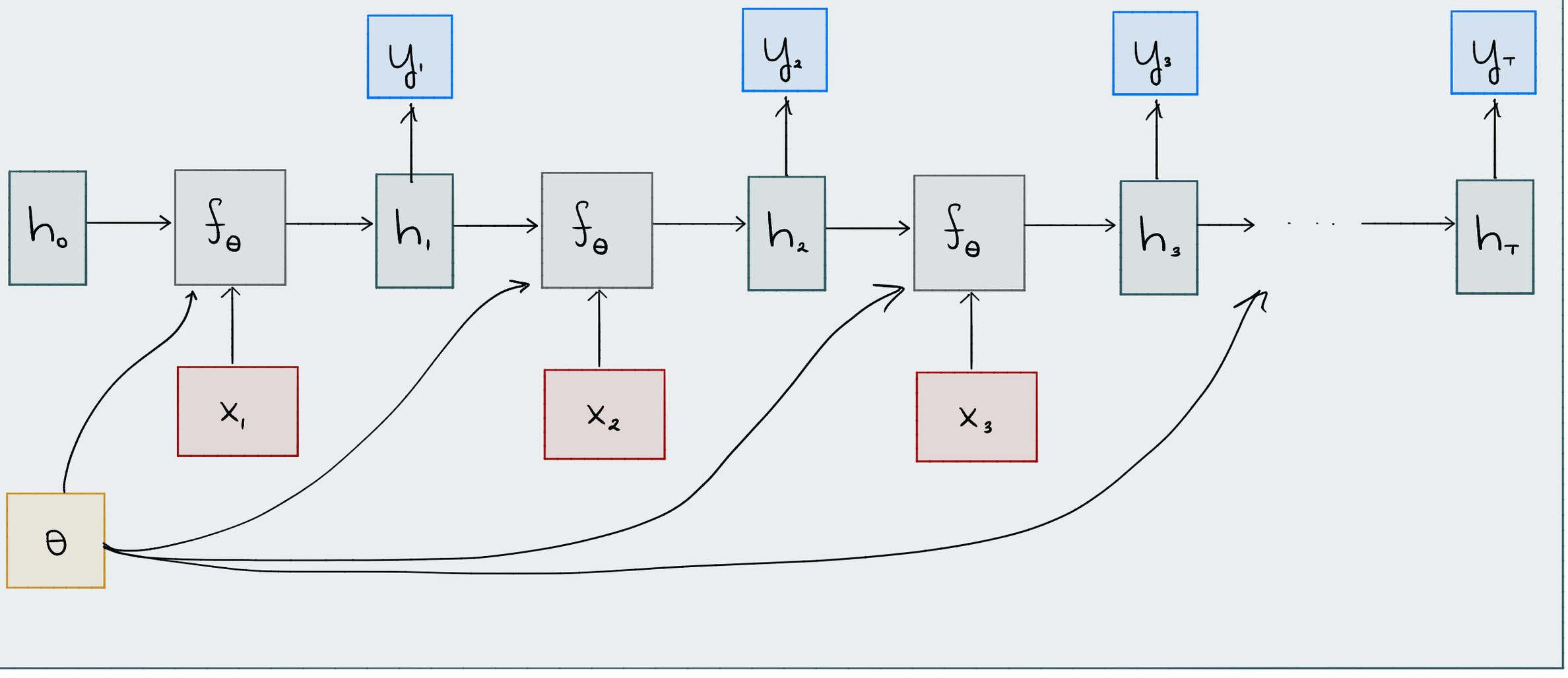


Examples

Example:

$$h_t = \tanh\left(W_h h_t + W_x x_t\right), \qquad y_t = W_y h_t$$

Example: Many-to-Many Network



Typical Schematic: