Computational Systems Biology
Deep Learning in the Life Sciences

6.802 6.874 20.390 20.490 HST.506

David Gifford
Lecture 3
February 16, 2017

Deep Networks, Convolutional Networks, and Recurrent Networks

http://mit6874.github.io

Overall goal for today

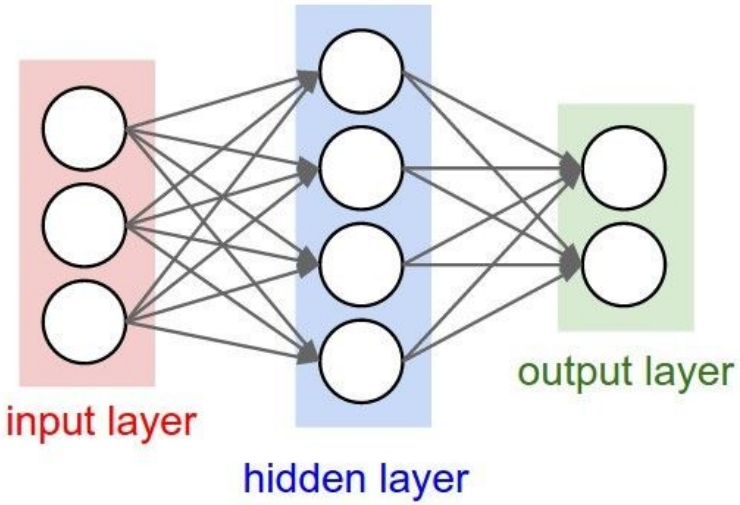
- Understand the key importance of parameter sharing in network design
- Understand how to design and learn deep convolutional neural networks (CNNs) that share parameters
- Understand the essential idea of recurrent neural networks (RNNs) that share parameters

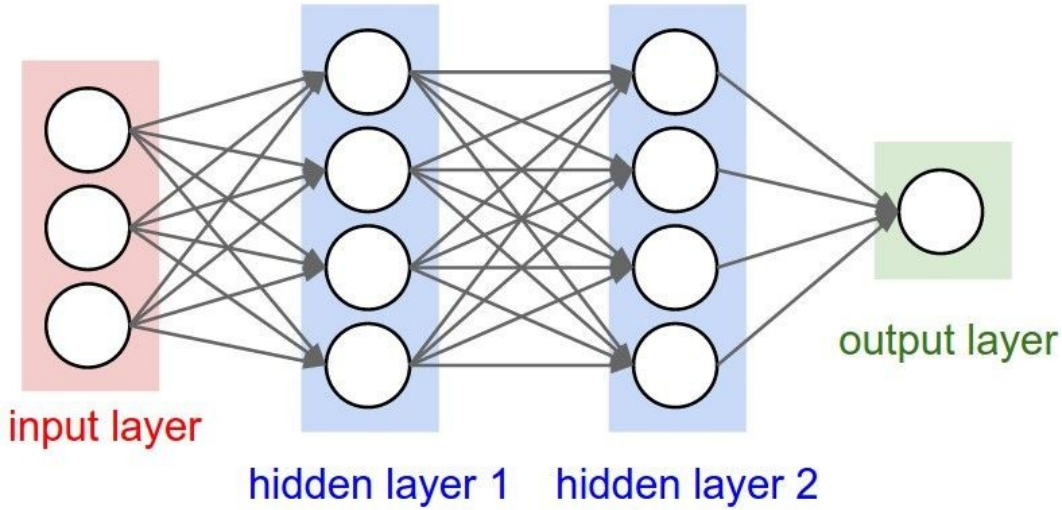
Today's lecture

- Convolution and pooling permit parameter sharing
  - Network architecture is key
  - Backprop works with convolution and pooling
  - Need to consider network capacity for stable, generalizable results
- Recurrent networks unroll a network to permit the processing of variable length sequential data
  - Text is typically processed by an RNN
  - LSTMs are a form of RNN that is optimized for the communication of information through time

Convolutional Networks (Part I)

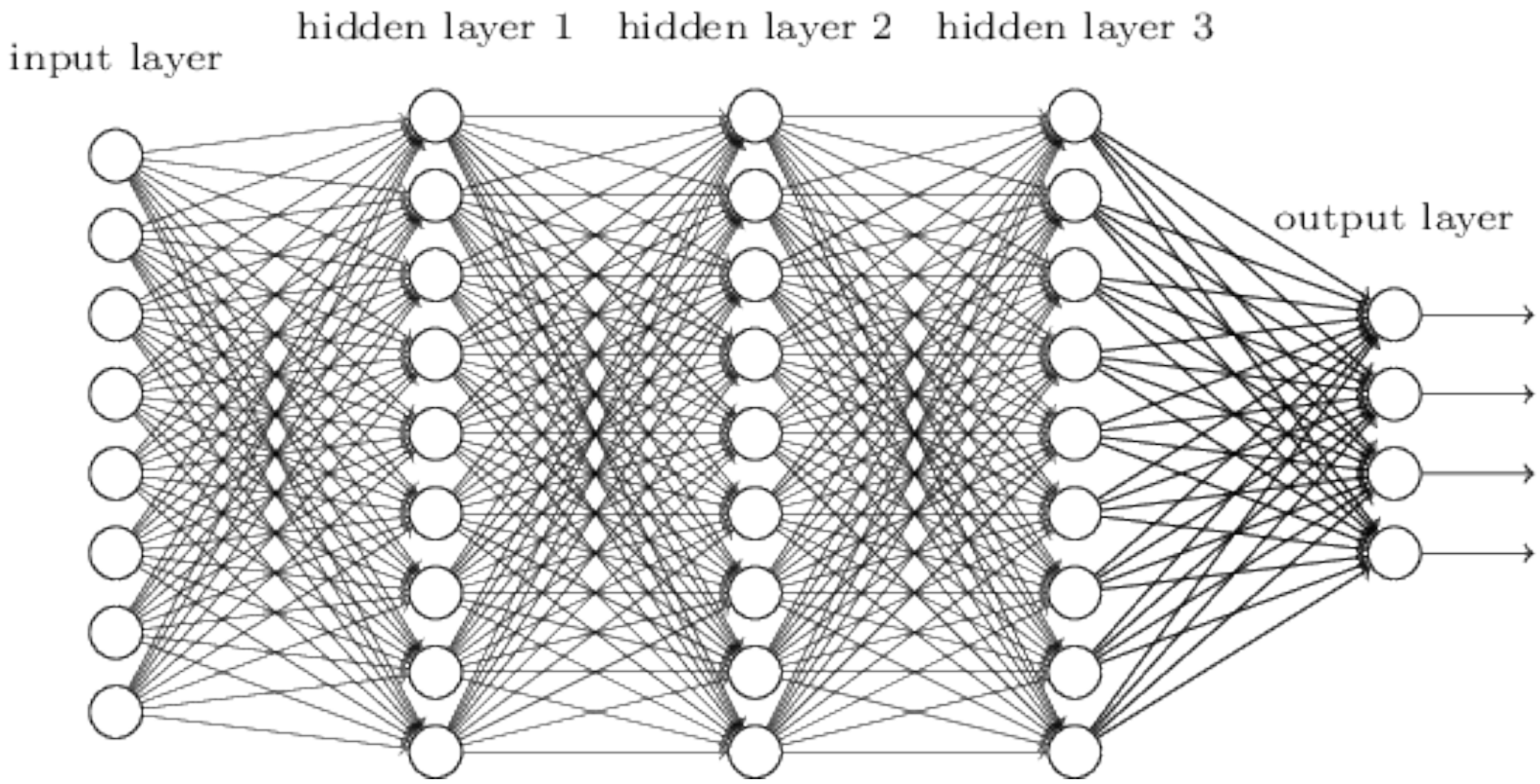
Deep neural networks vary in their hidden layer architecture



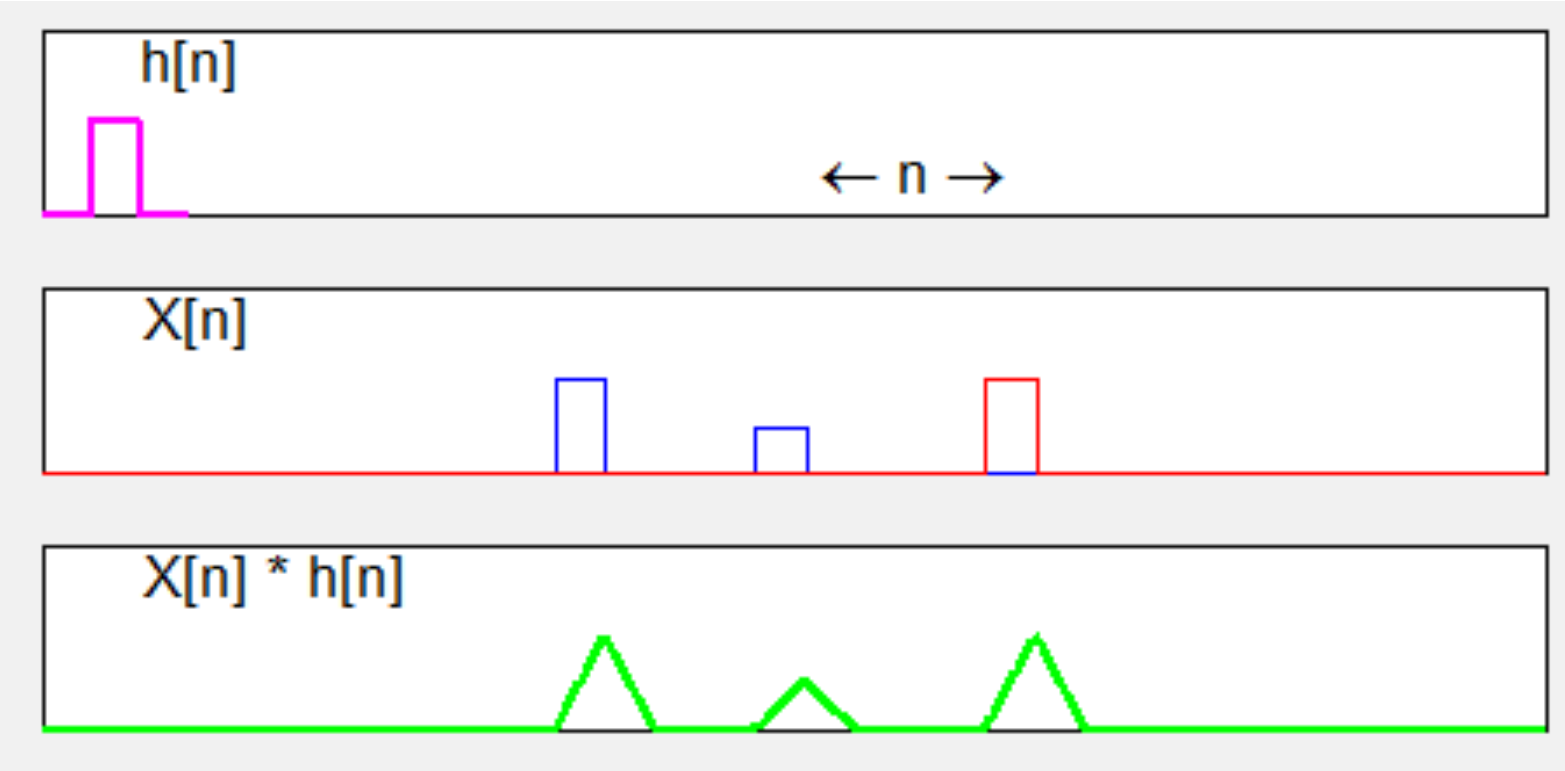


- Solve for parameters using training data and backprop to minimize a cost function (negative log likelihood) by gradient descent

As networks grow in complexity they have a large number of parameters to learn

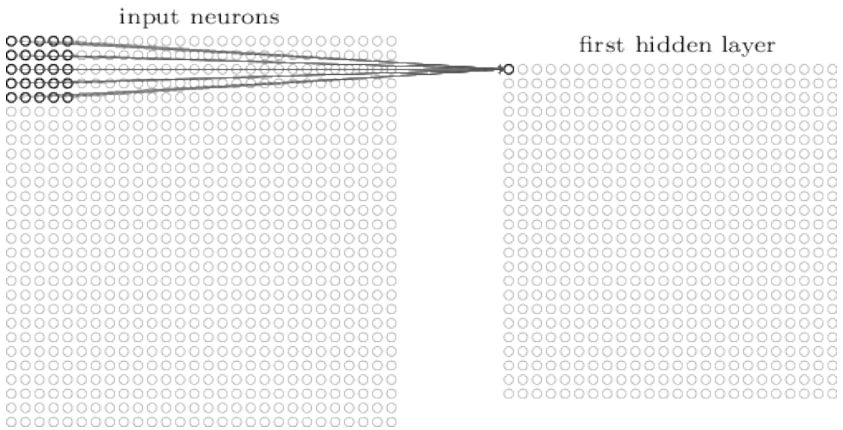


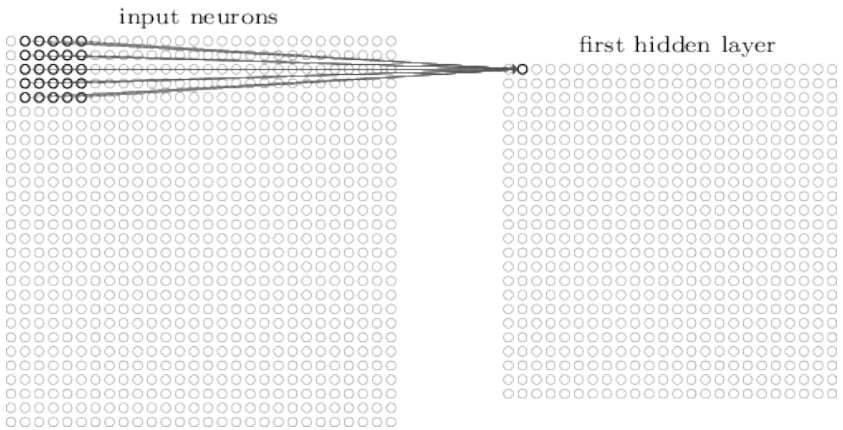
Convolution provides us with a way to share parameters for pattern identification



Imagine we wish to find steps in X[n]
We convolve X[n] with the step kernel h[n]

Convolution generalizes to arbitrary dimensions (2D images, 3D voxels, 4D color video)





$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

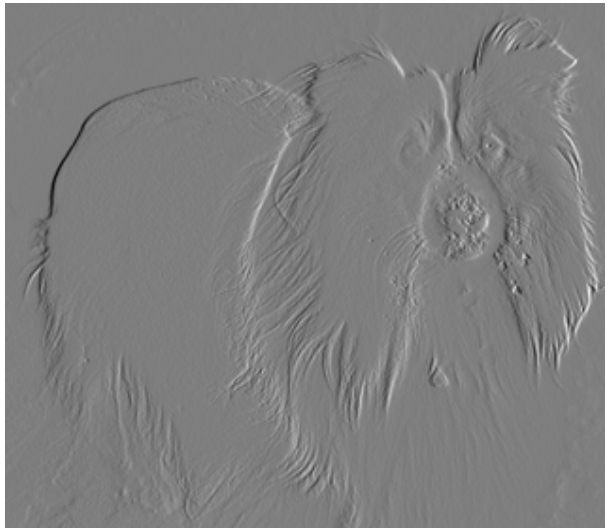
Convolution

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

Cross-correlation

Convolution can detect features (here edges)

Input



Output

1	-1

Kernel

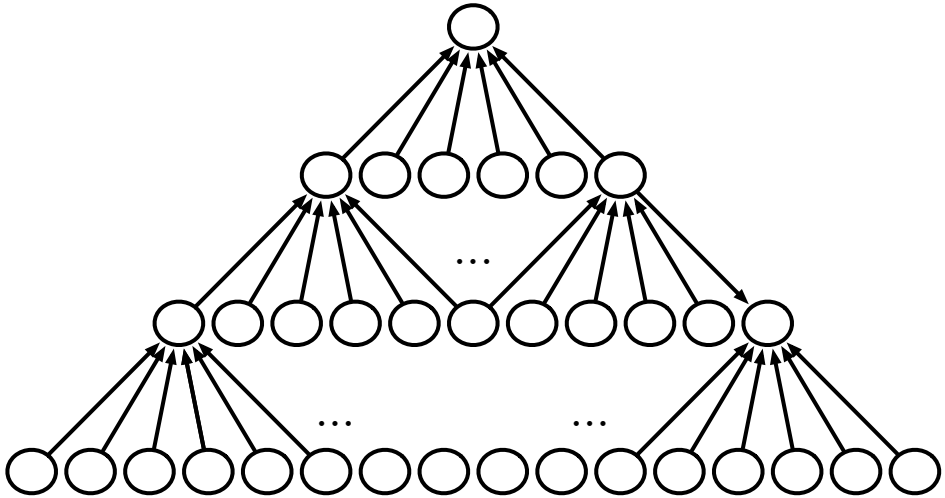
There are three approaches to edge cases in convolution

- **Valid convolution**: output only when entire kernel is contained in input (shrinks output)
- **Same convolution**: zero pad input so output is same size as input dimensions
- **Full convolution**: zero pad input so output is produced whenever an output value contains at least one input value (expands output)

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

Zero Padding Controls Output Size

Valid -
no padding



Same -
add zeros

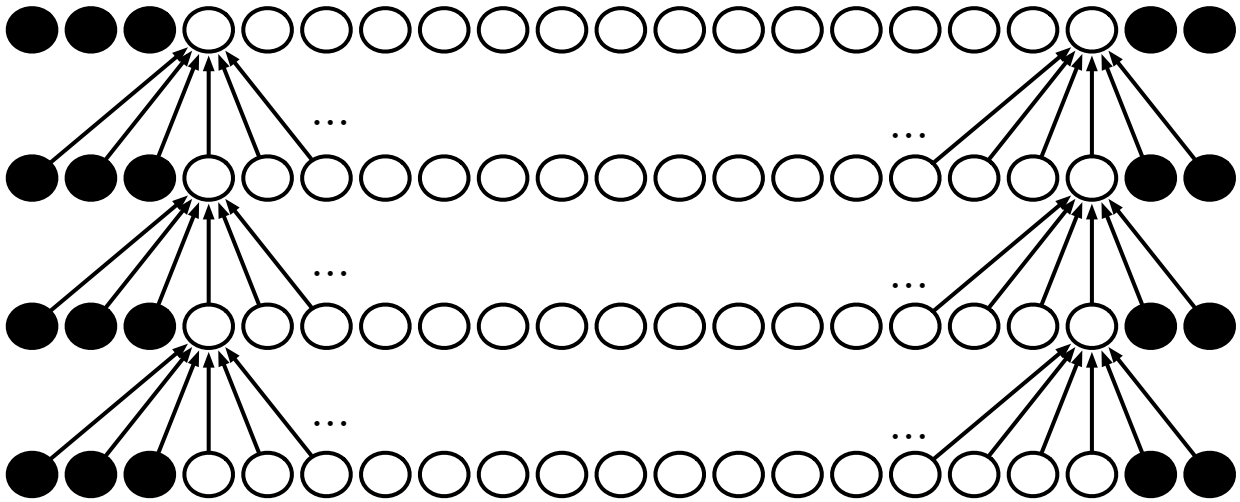
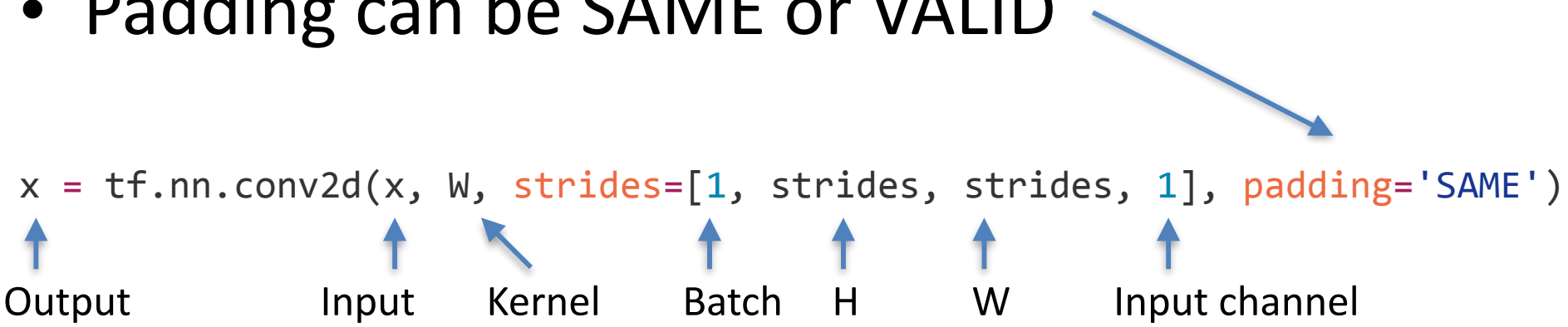


Figure 9.13

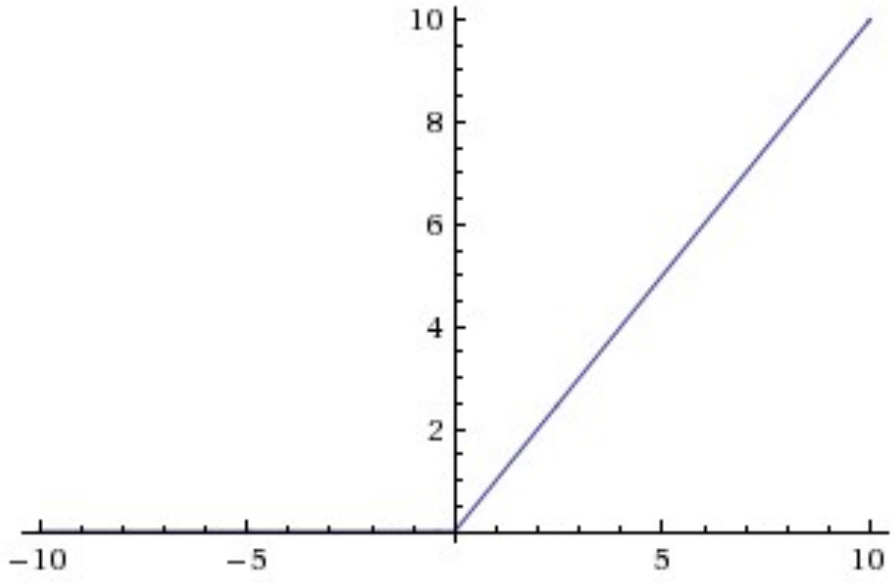
TF convolution operator takes stride and zero fill option as parameters

- Stride is distance between kernel applications in each dimension
- Padding can be SAME or VALID



The REctified Linear Unit (RELU) is a common non-linear **detector** stage after convolution

```
x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
x = tf.nn.bias_add(x, b)
x= tf.nn.relu(x)
```



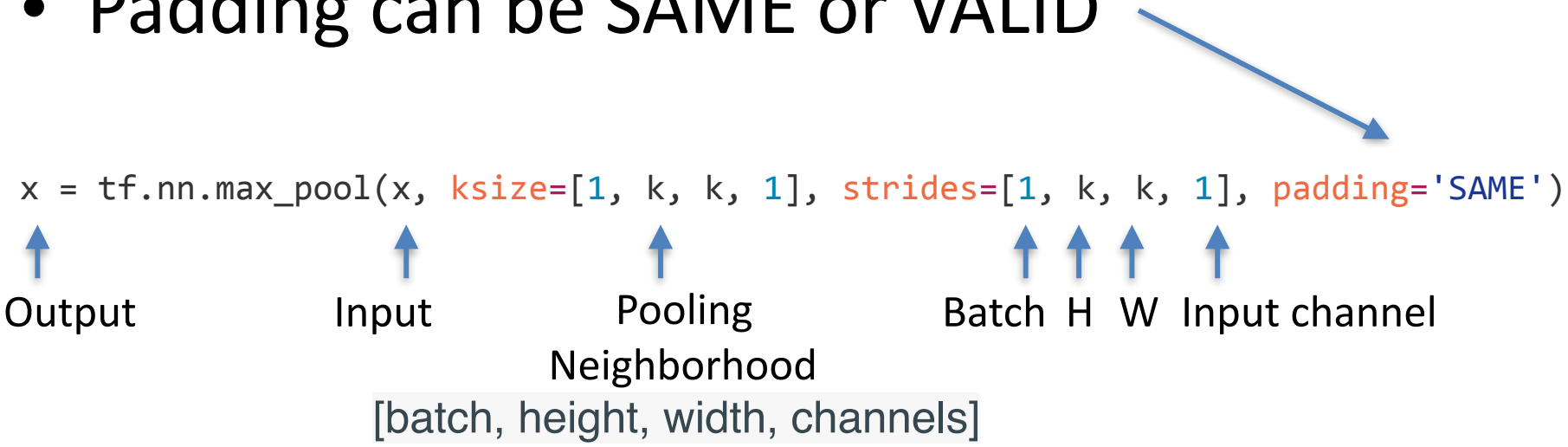
f(x) = max(0, x)

When will we backpropagate through this?

Once it “dies” what happens to it?

Pooling reduces dimensionality by giving up spatial location

- **max pooling** reports the maximum output within a defined neighborhood
- Padding can be SAME or VALID



A convolutional layer has convolution, detection, and pooling stages

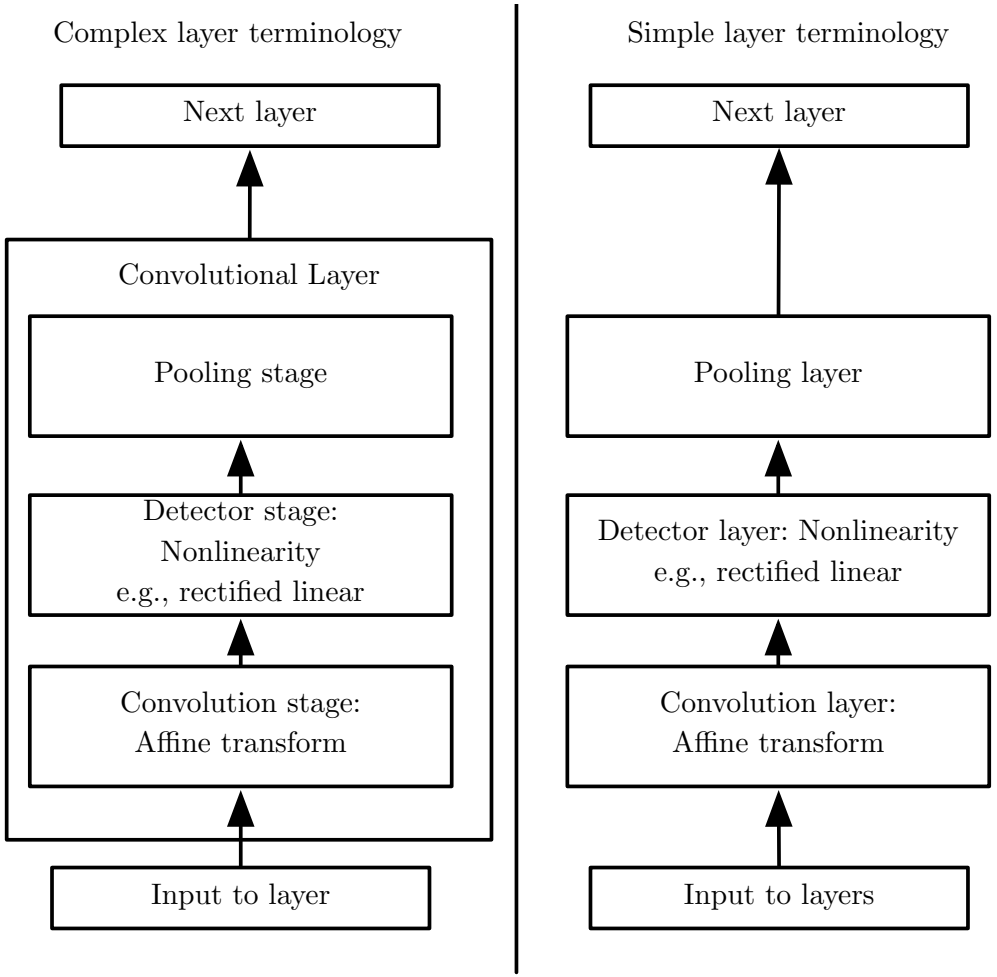


Figure 9.7

Dropout regularizes a broad class of models by testing lots of alternative structures

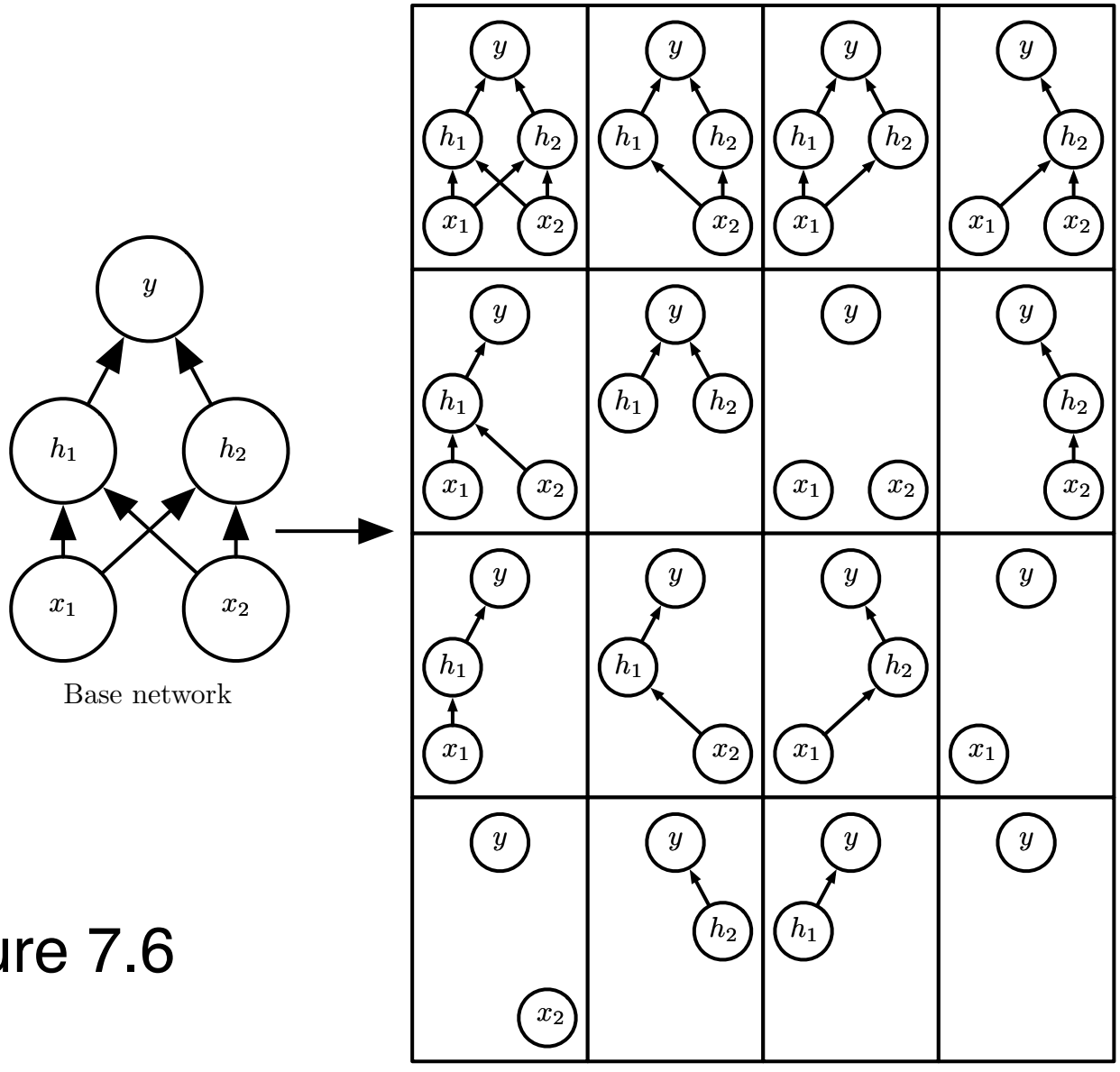


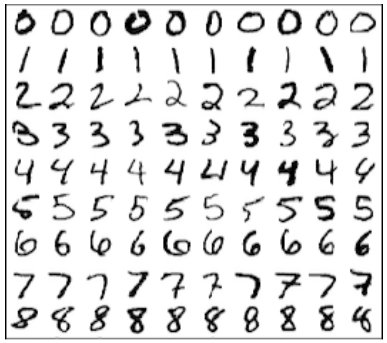
Figure 7.6

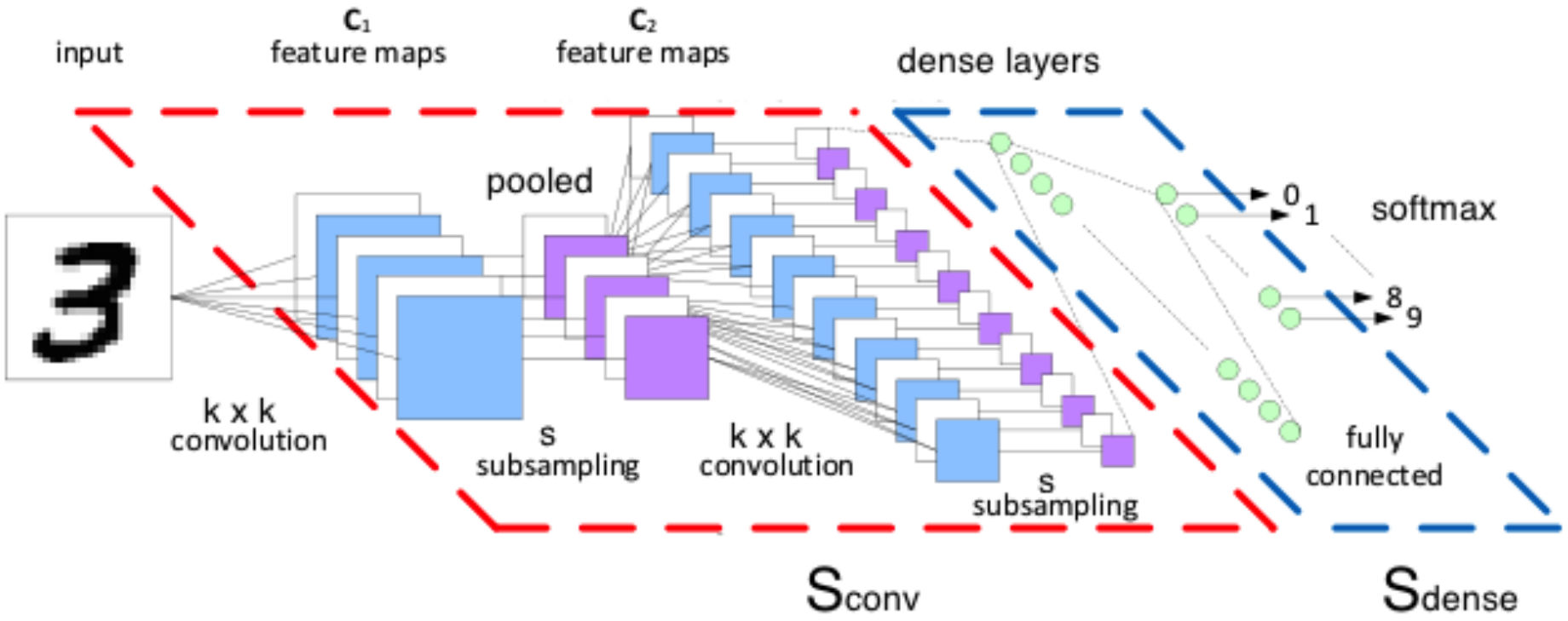
TF has a builtin dropout operator

```
# Apply Dropout
fc1 = tf.nn.dropout(fc1, dropout)
```

Probability of link dropout

Example MNIST network





Regularization is an important aspect of learning a stable model that generalizes well

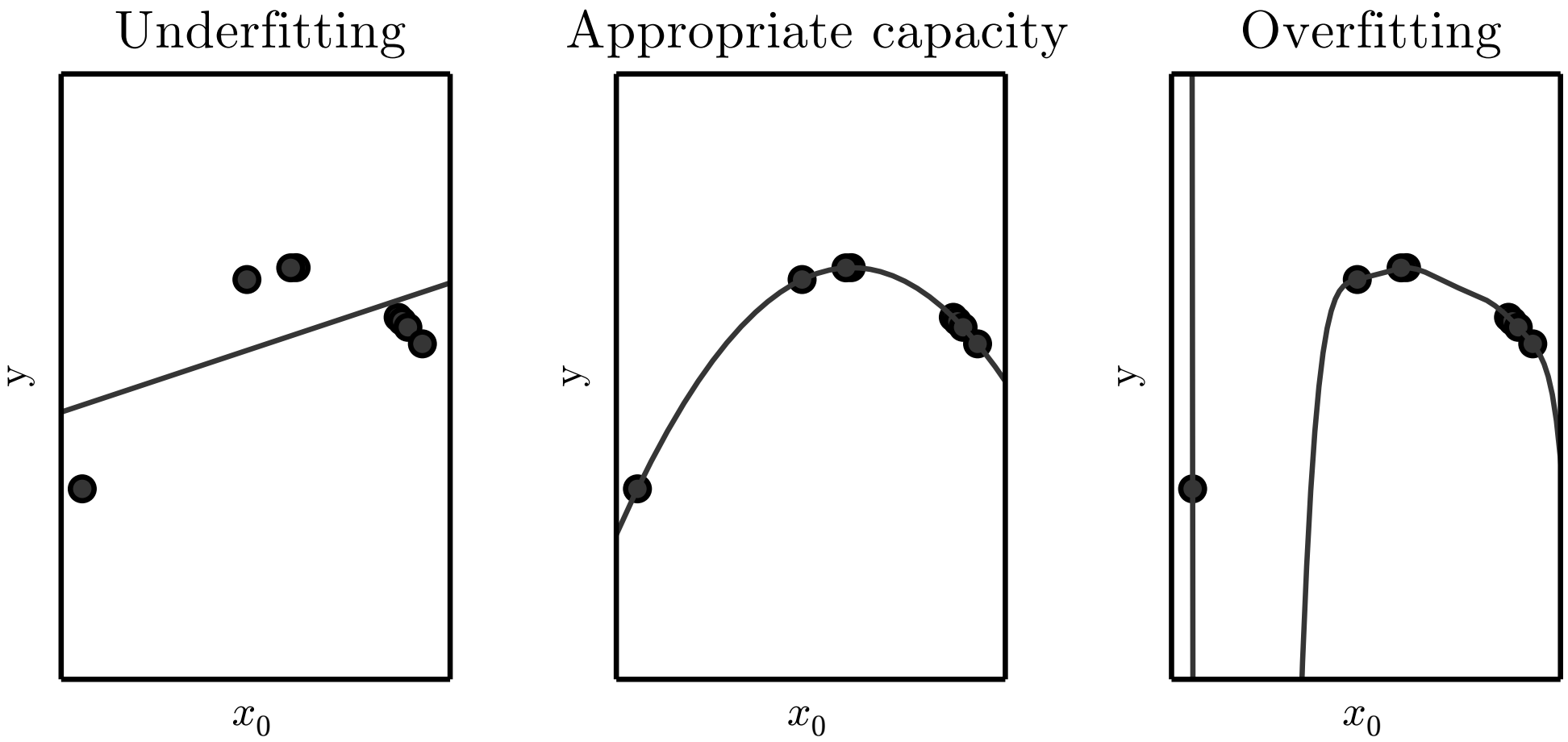


Figure 5.2

Penalizing Weights

- L1 Norm attempts to drive weights to 0 (make weight vector sparse) (p = 1)
- L2 Norm attempts to minimize magnitude of weights (p = 2)

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

Regularizing regression

$$\mathrm{Err}_{\mathrm{ridge}}(w, w_0) = \sum_{i=1}^{n} \left(w \cdot x^{(i)} + w_0 - y^{(i)} \right)^2 + \lambda \|w\|_2^2$$

$$z_j^{(i)} = x_j^{(i)} - \bar{x}_j \qquad y_c^{(i)} = y^{(i)} - \bar{y} \qquad \|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

$$\begin{aligned} E_{\mathrm{ridge}}(W) &= (Y_c - ZW)^T (Y_c - ZW) + \lambda W^T W \\ \nabla_W E_{\mathrm{ridge}}(W) &= Z^T (ZW - Y_c) + \lambda W = 0 \\ W_{\mathrm{ridge}} &= (Z^T Z + \lambda I)^{-1} Z^T Y_c \ . \end{aligned}$$

Weight Decay

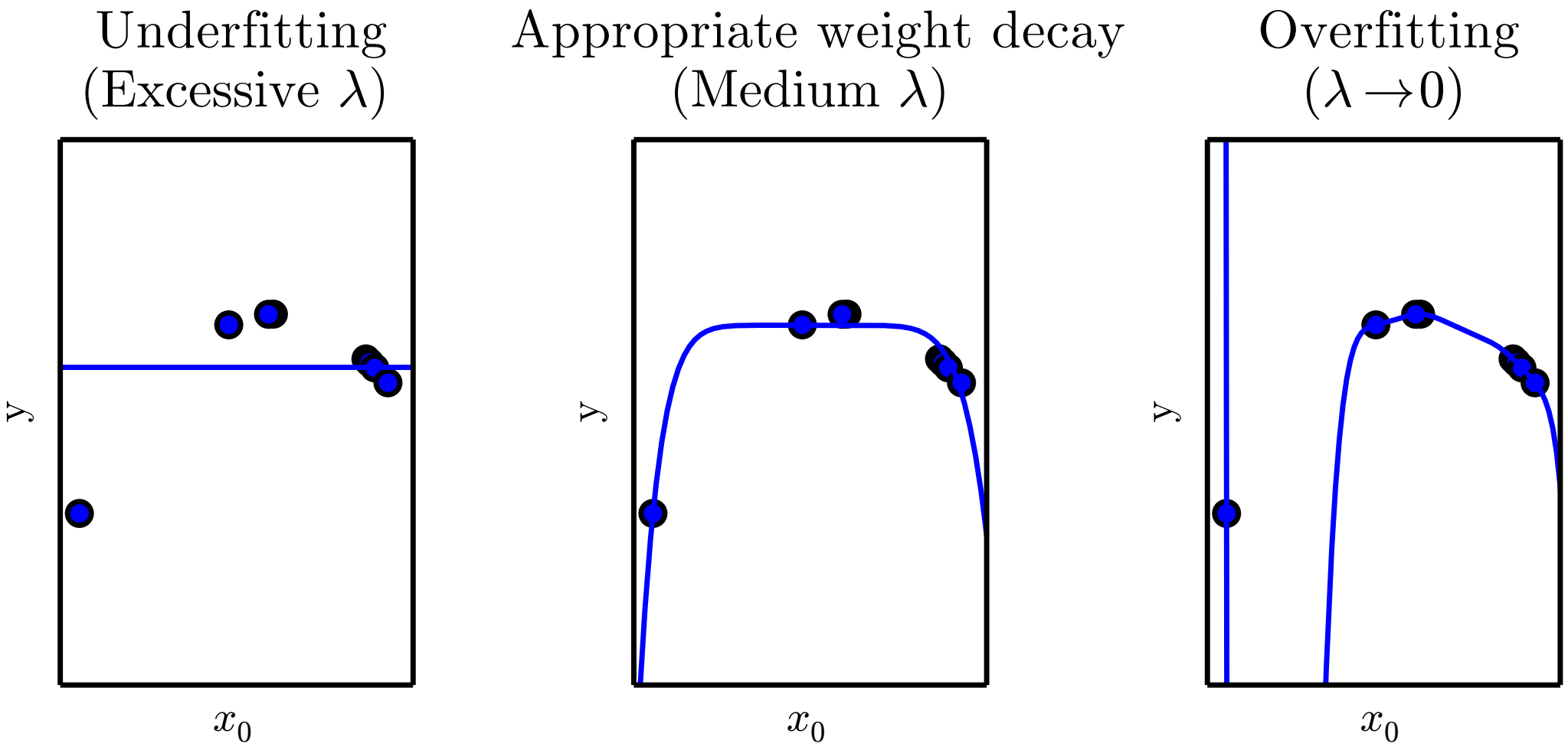


Figure 5.5

Getting the right “fit” to the training data is important

- Underfitting - high error on training set
- Overfitting - large gap between training and test error
- Correct fit is important so the model generalizes to new examples

Model capacity

- Capacity - ability to fit a wide range of functions (hypothesis space)
- Vapnik-Chervonenkis dimension - size of largest unique training set of examples a binary classifier can label arbitrarily is a measure of capacity

Generalization and Capacity

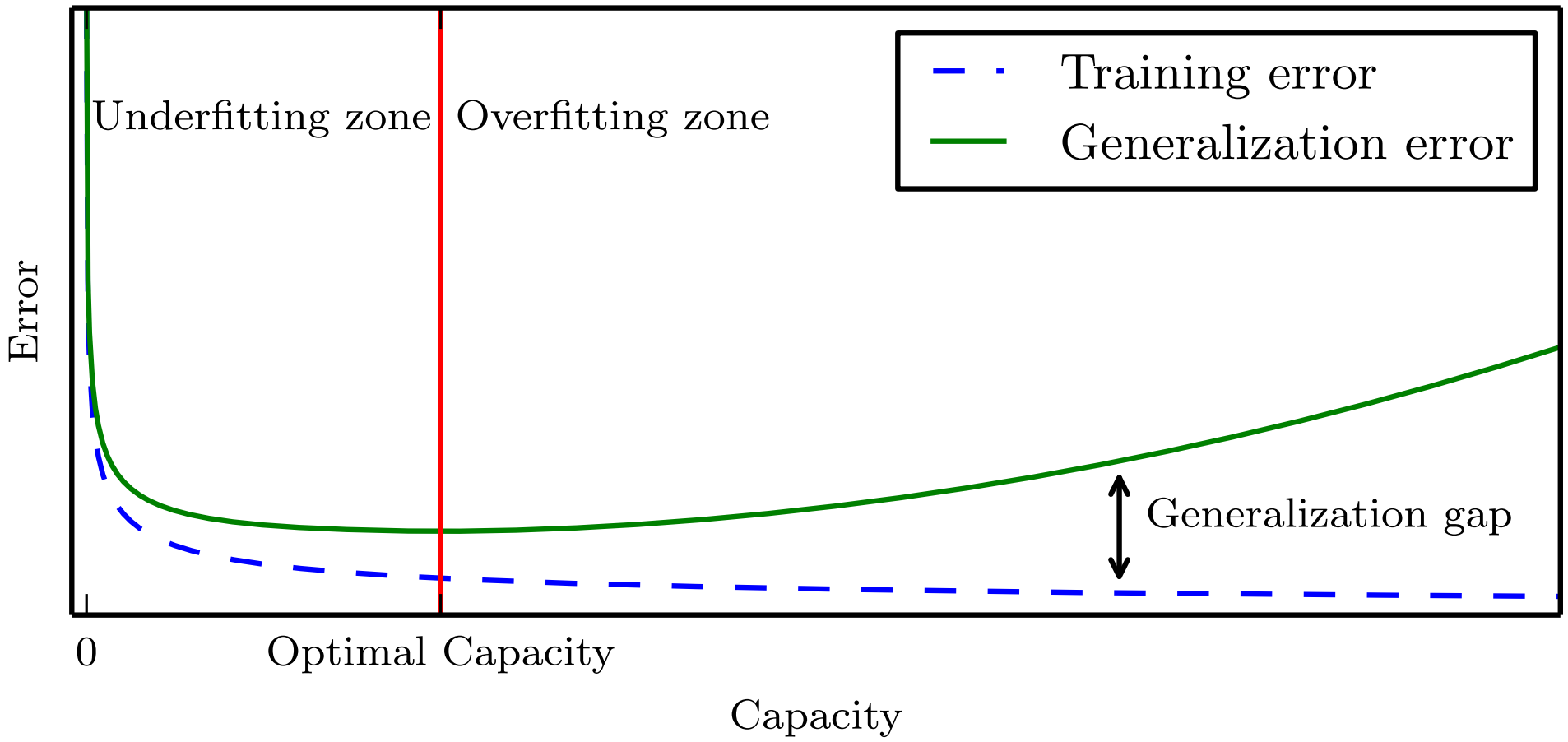


Figure 5.3

The capacity of non-parametric models is defined by the size of their training set

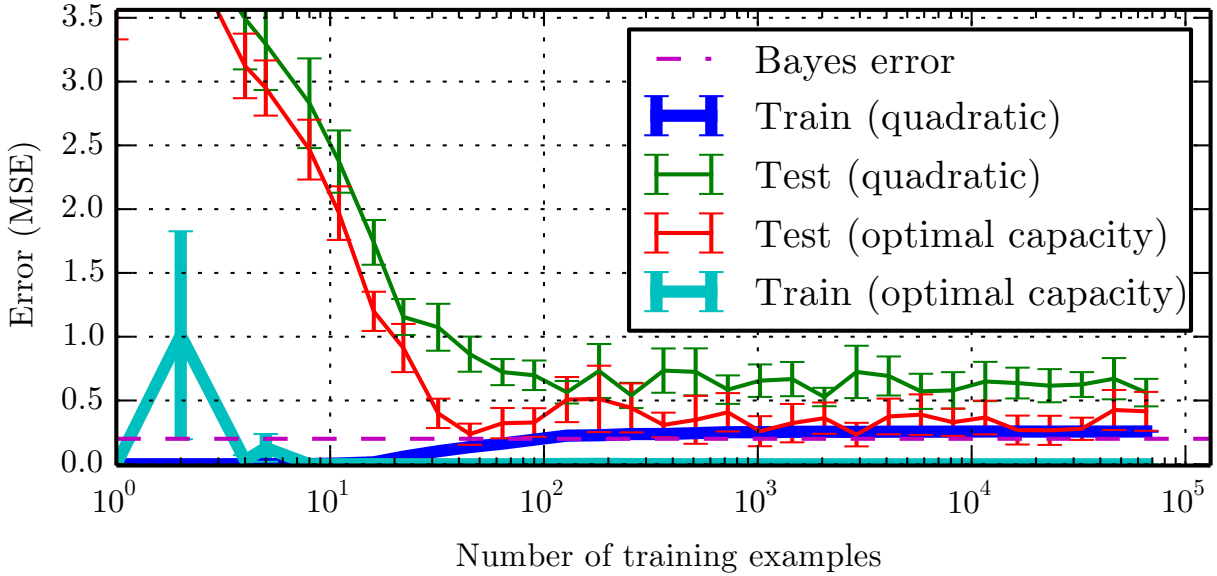
- k-nearest neighbor (KNN) regression computes its output based upon the k “nearest” training examples
- Often the best method, and certainly a baseline to beat

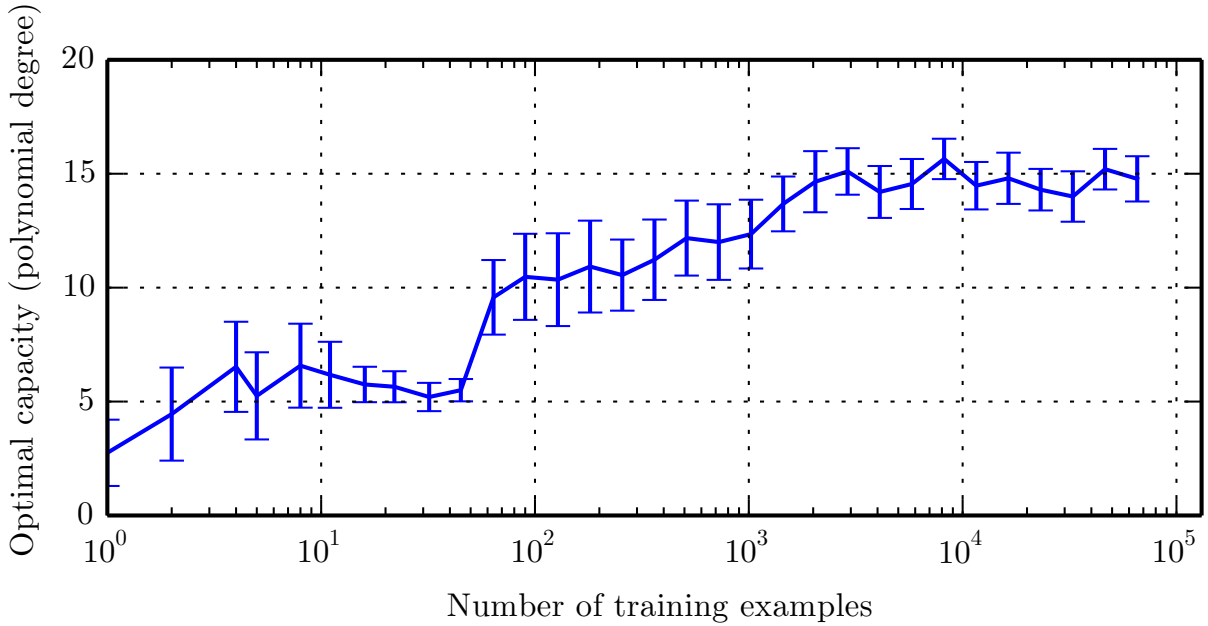
Bayes error is residual noise from confounders and observation noise

- Bayes error is the error made by an Oracle given the training set
- For example, if there is an unobserved variable that affects the labels the Oracle's predictions will be noisy

Sufficient training data are necessary to generalize well

Figure 5.4





Machine Learning has limits

- The *no free lunch* theorem: You can only obtain generalization from finitely many training examples if the algorithm searches a limited hypothesis space.
- We desire a learning algorithm to perform *generalization* and to be *stable*. It generalize if the training error on a data set will converge to the expected error. It is stable if small perturbations in the data result in only small perturbations in the output hypothesis.

Recurrent Networks (Part II)

Recurrent networks share parameters across time indexes

- Recurrent networks have inter-time dependencies with each time-step sharing parameters.
- For example, such dependencies can include the value of the hidden units or outputs of the previous stage

We can unfold a simple one layer network to handle sequential data with all parameters shared between time points

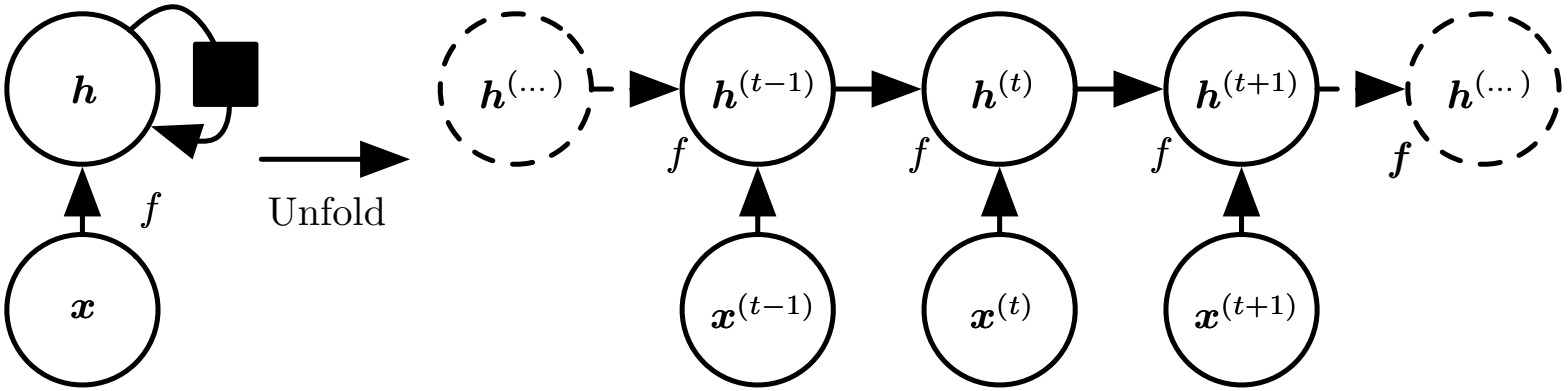


Figure 10.2

We can train an RNN based upon data likelihood as before

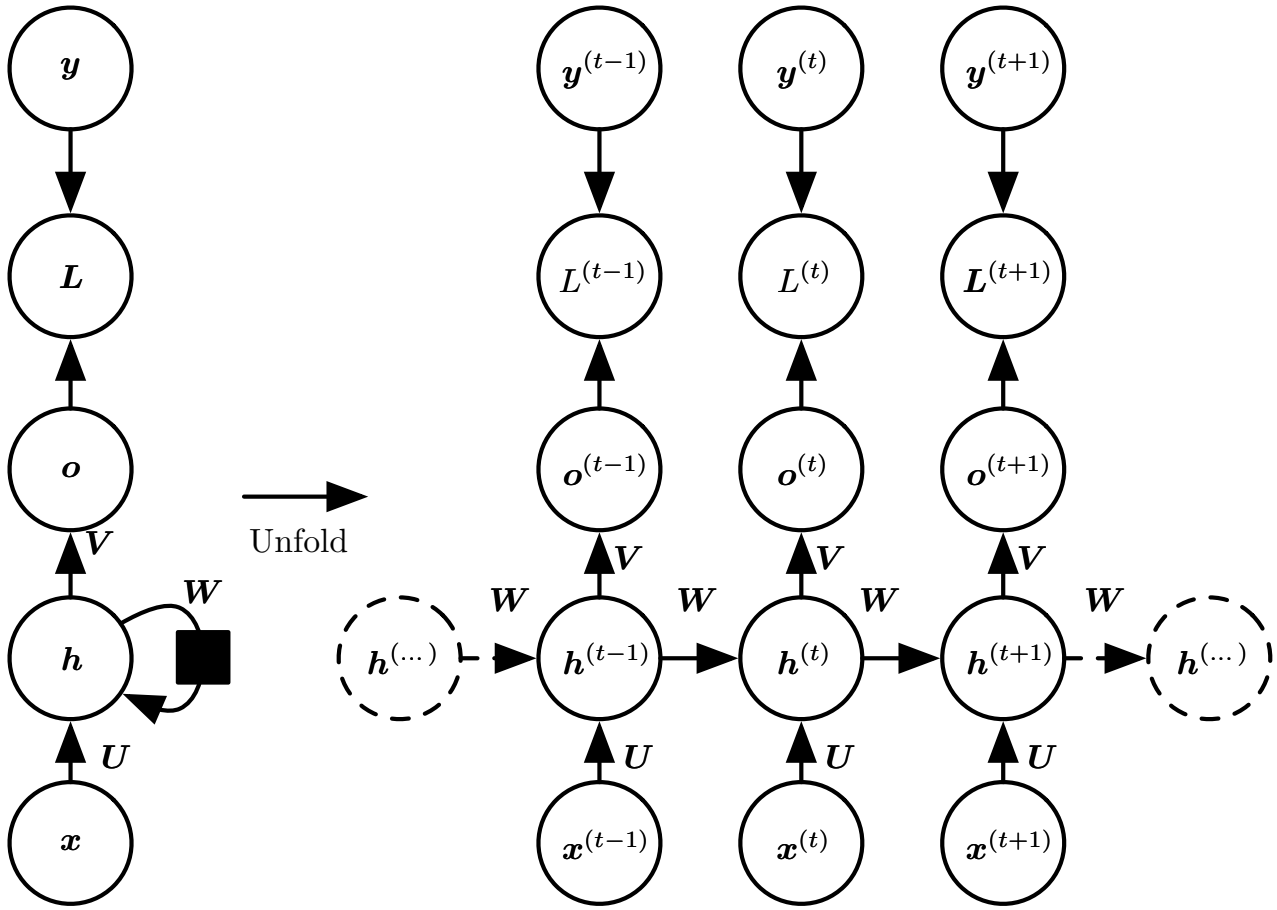
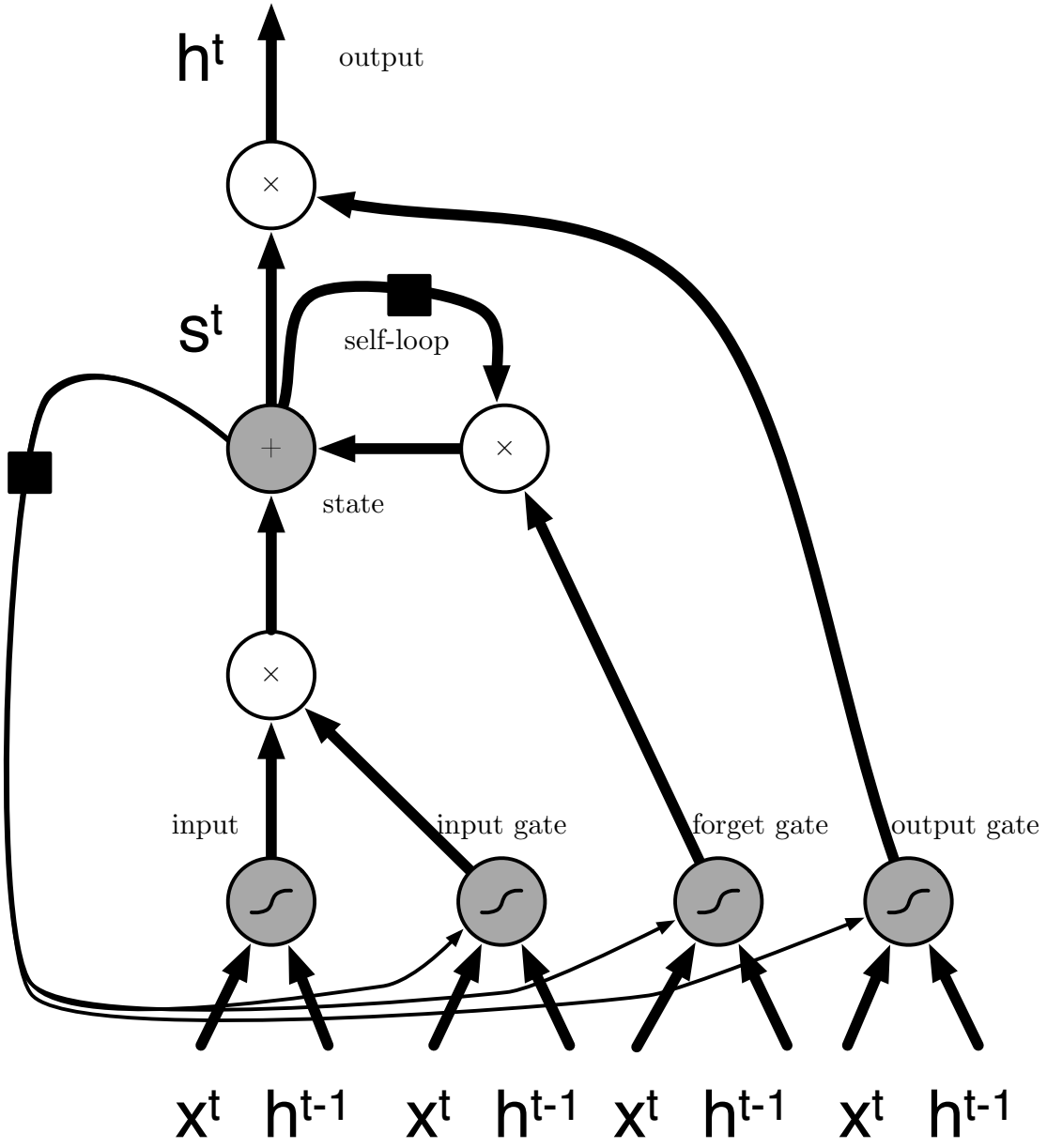


Figure 10.3

The Long-Short Term Memory (LSTM) is a gated RNN



Today's lecture

- Convolution and pooling permit parameter sharing
  - Network architecture is key
  - Backprop works with convolution and pooling
  - Need to consider network capacity for stable, generalizable results
- Recurrent networks unroll a network to permit the processing of variable length sequential data
  - Text is typically processed by an RNN
  - LSTMs are a form of RNN that is optimized for the communication of information through time

FIN - Thank You