UCI University of California, Irvine

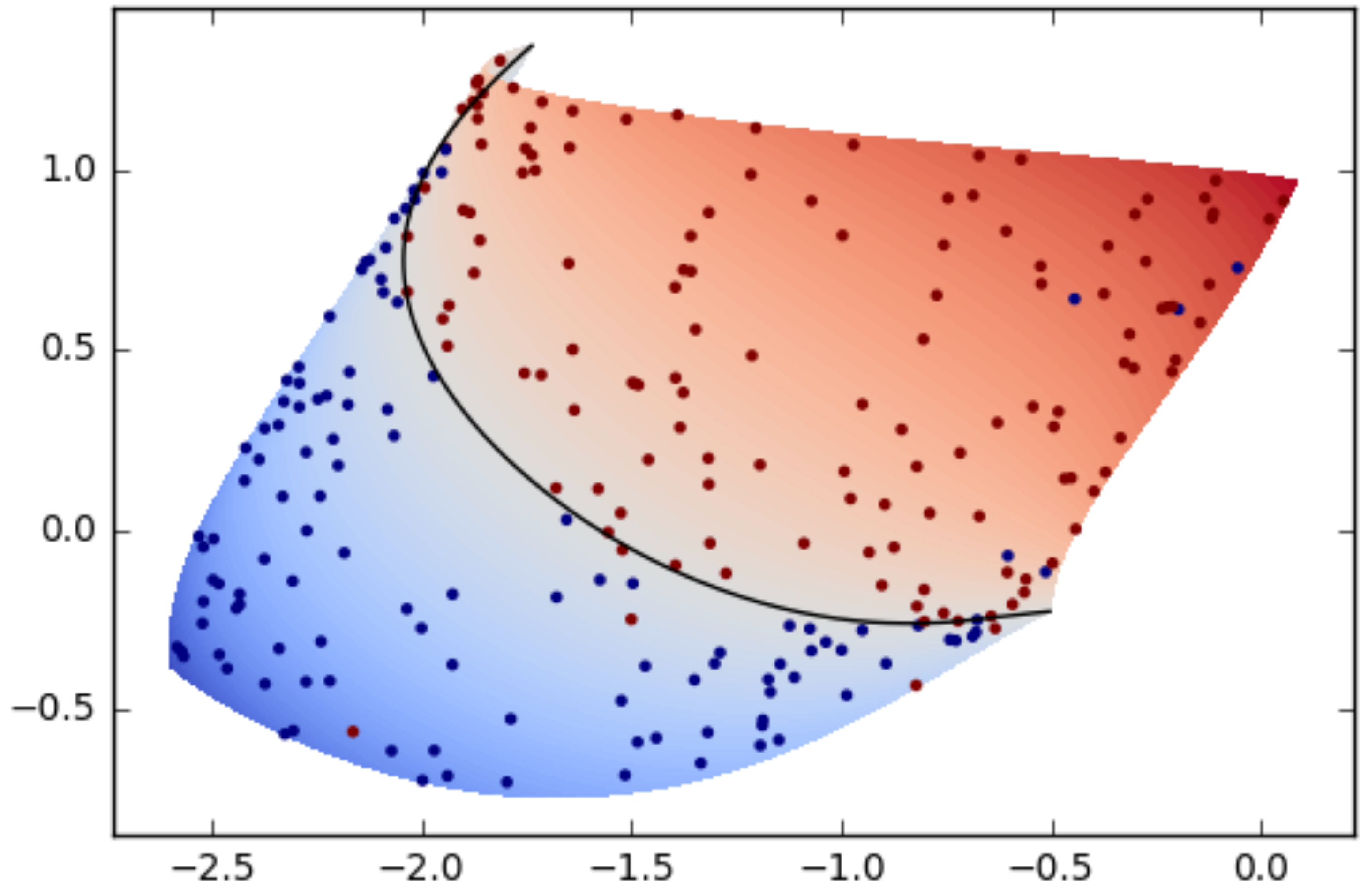
CS 273A: Machine Learning

Winter 2021

Lecture 11: Neural Networks

Roy Fox
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

All slides in this course adapted from Alex Ihler & Sameer Singh



Logistics

project

- Project abstract due Tue, Feb 16

assignments

- Assignment 4 to be published soon

Today's lecture

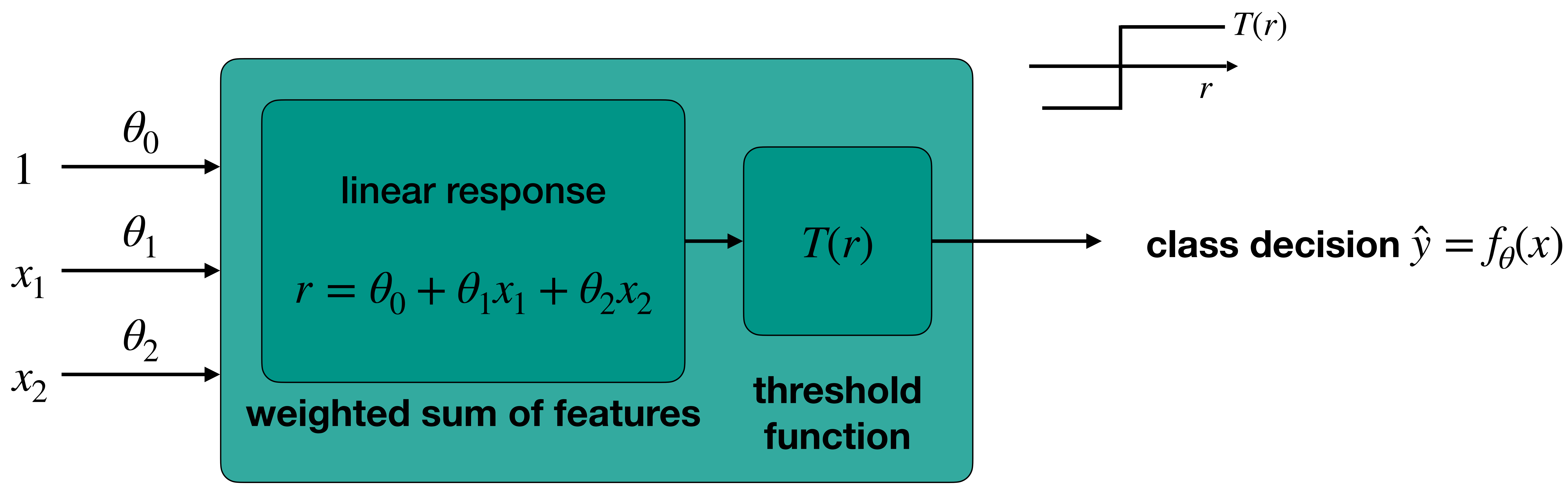
Multilayer Perceptrons

Backpropagation

Advanced Neural Networks

Linear classifiers

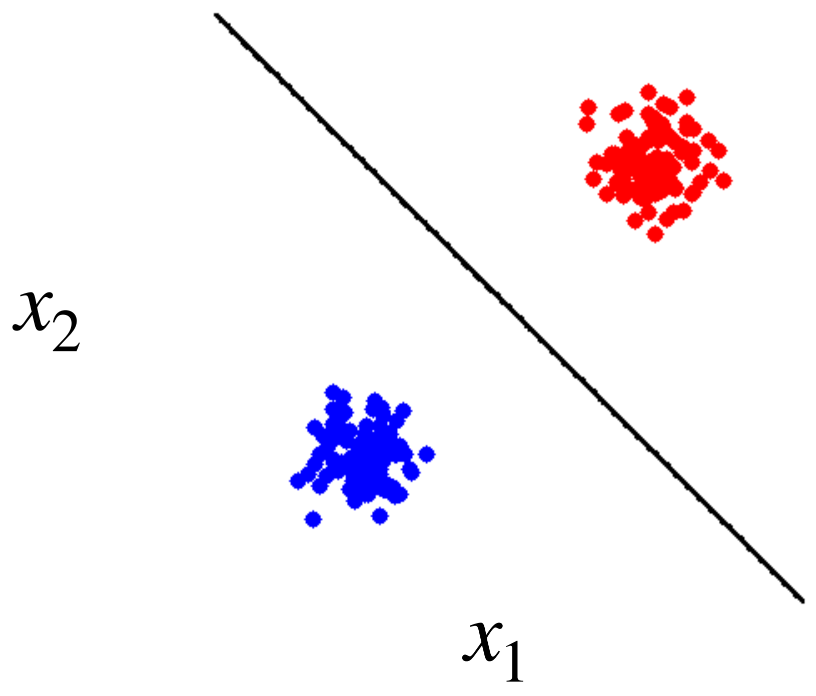
- Perceptron = use hyperplane to partition feature space → classes
    - Soft classifiers (logistic) = sensitive to margin from decision boundary



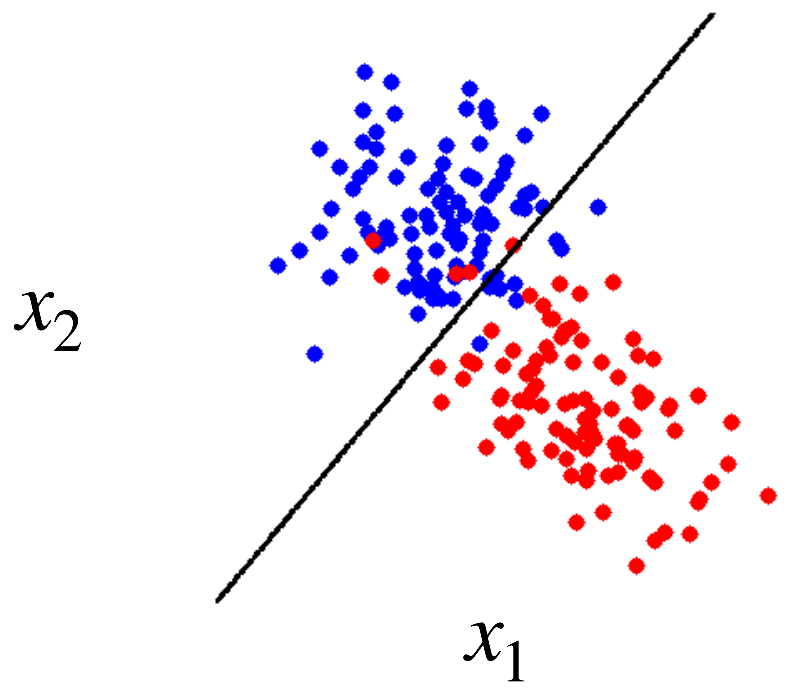
Adding features

- If data is non-separable in current feature space
  - Perhaps it will be separable in higher dimension $\Longrightarrow$ add more features
  - E.g., polynomial features: linear classifier $\rightarrow$ polynomial classifier
- Which features to add?
  - Perhaps outputs of simpler perceptrons?

Linearly separable data



Linearly non-separable data

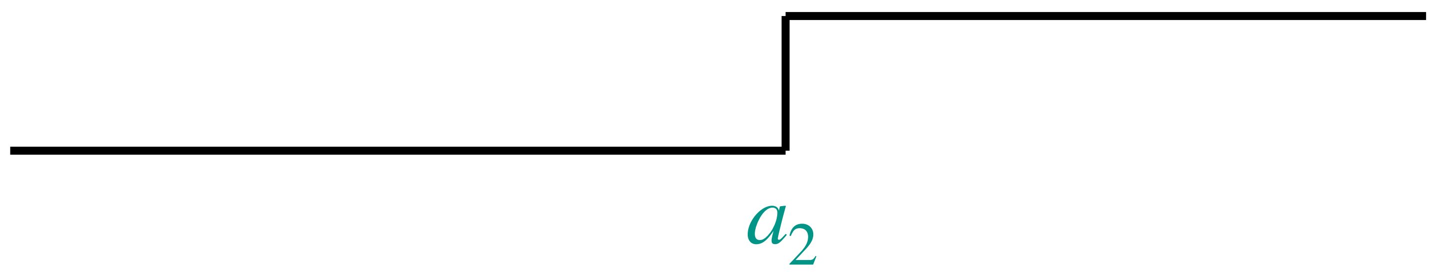


Combining step functions

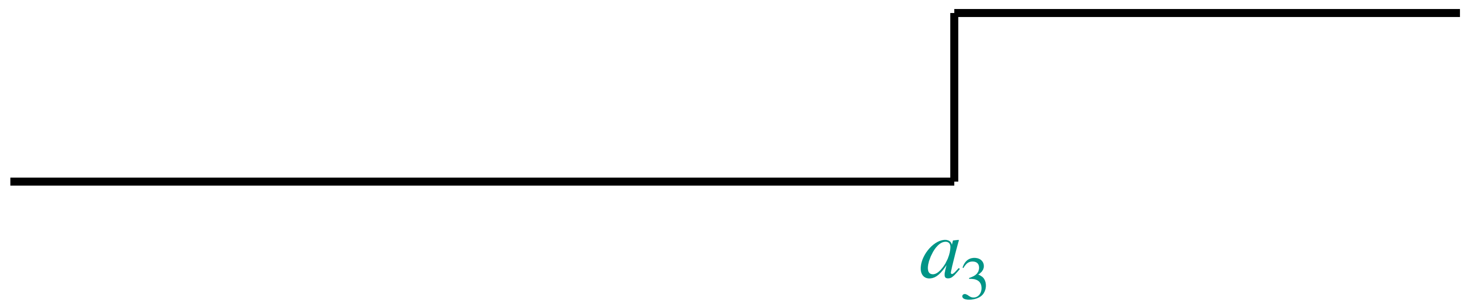
- Combinations of step functions allow more complex decision boundaries

$F_1(x) = T(x - a_1)$

$F_2(x) = T(x - a_2)$



$F_3(x) = T(x - a_3)$



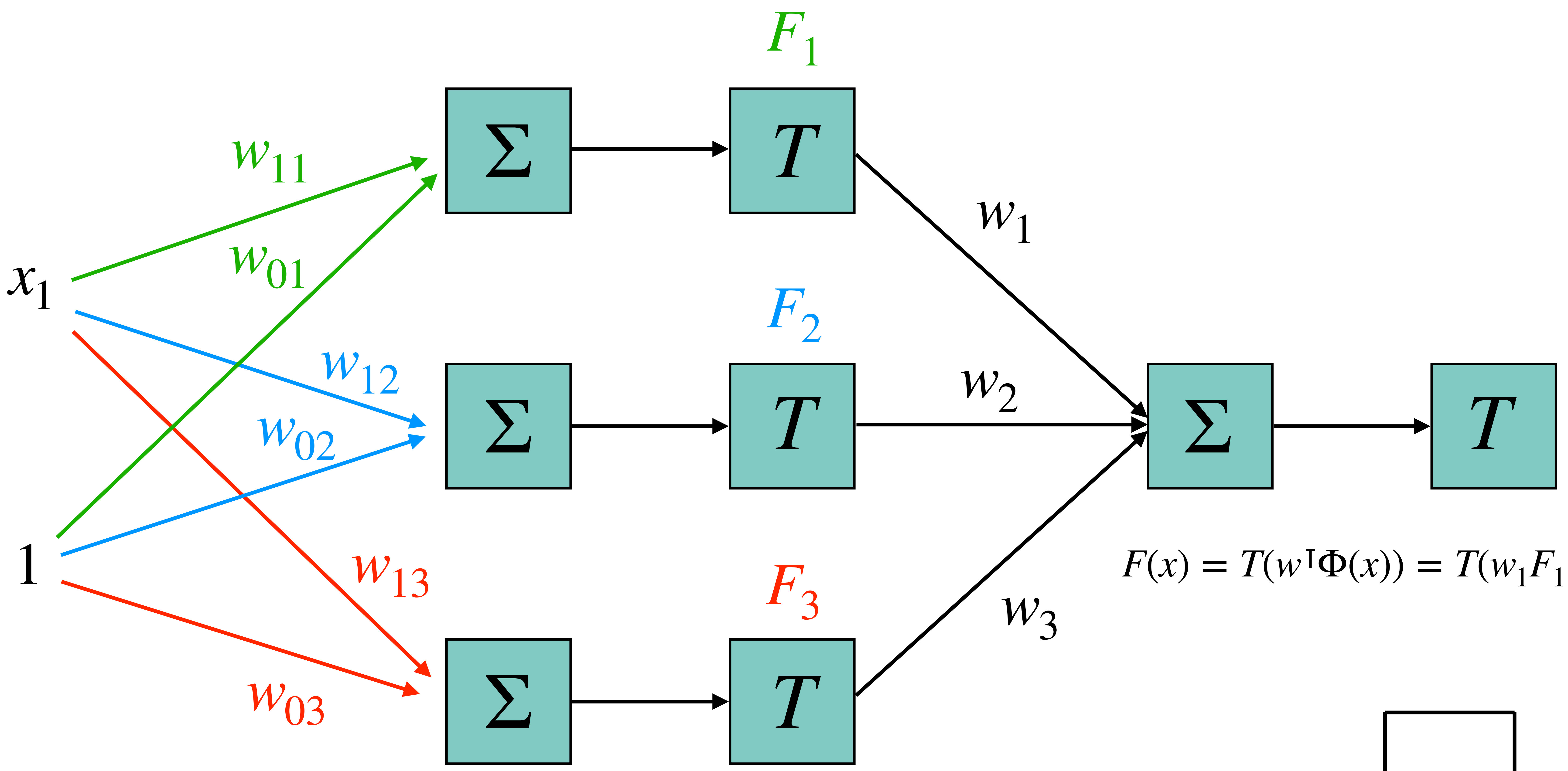
$$\Phi(x) = \begin{bmatrix} F_1(x) & F_2(x) & F_3(x) \end{bmatrix}$$

is piecewise constant

$$F(x) = T(w^\intercal \Phi(x)) = T(w_1 F_1(x) + w_2 F_2(x) + w_3 F_3(x) + w_4)$$

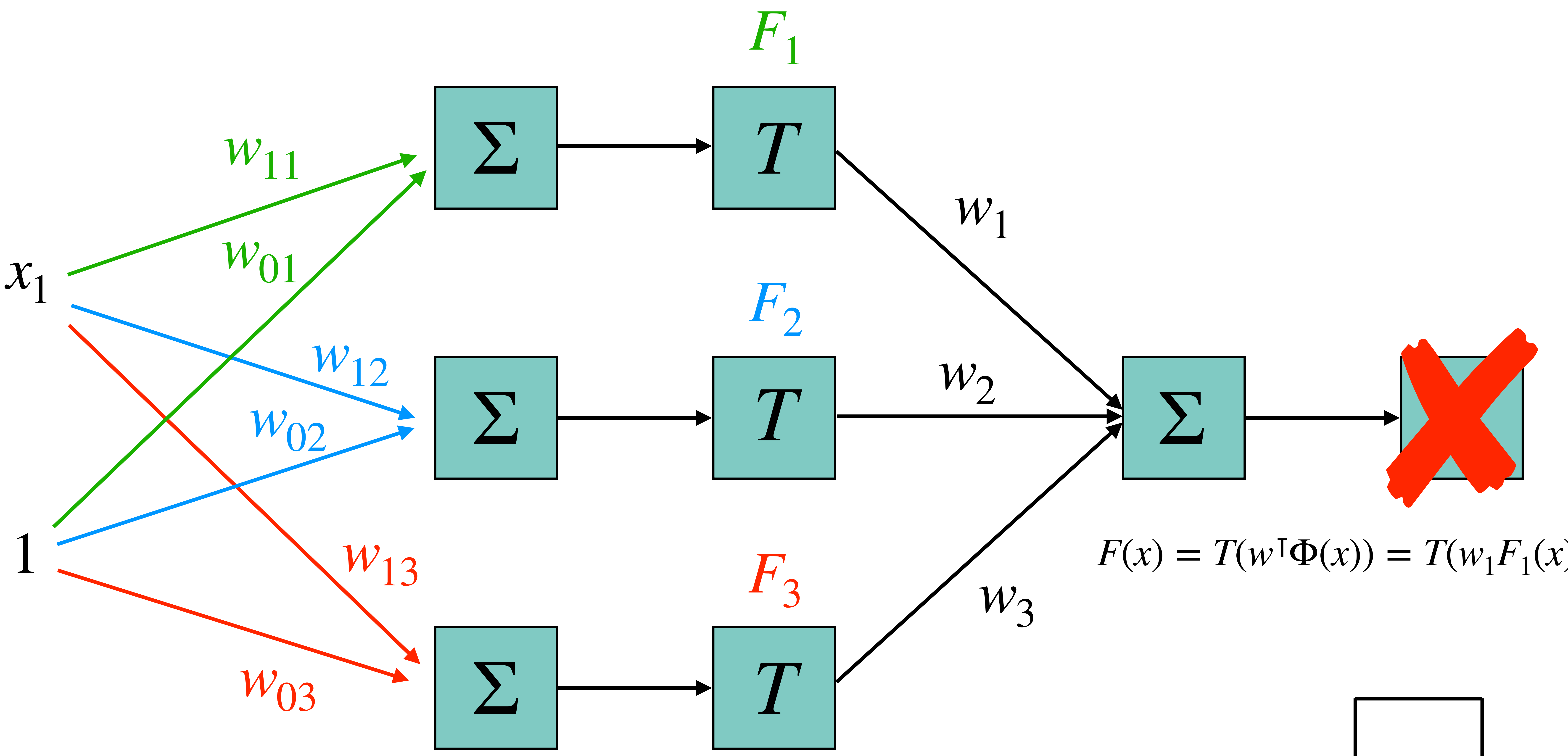
- Need to learn:
  - Thresholds a_1, a_2, a_3
  - Weights w_1, w_2, w_3, w_4

Multi-Layer Perceptron (MLP)



$$F(x) = T(w^\intercal \Phi(x)) = T(w_1 F_1(x) + w_2 F_2(x) + w_3 F_3(x) + w_4)$$

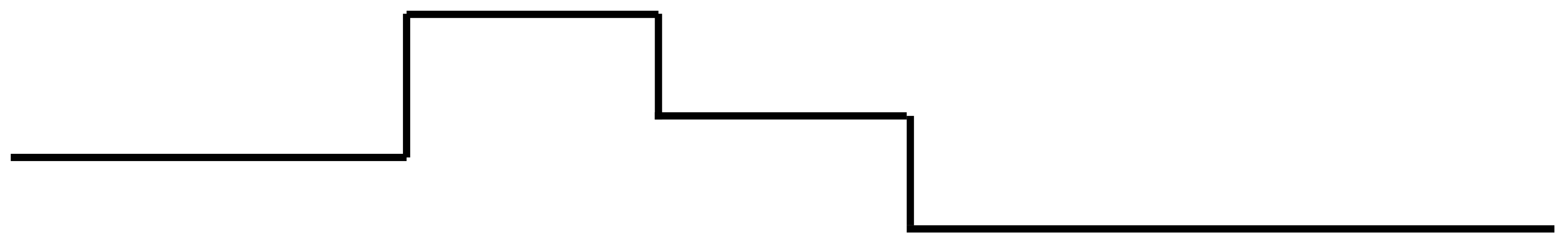
Multi-Layer Perceptron (MLP)



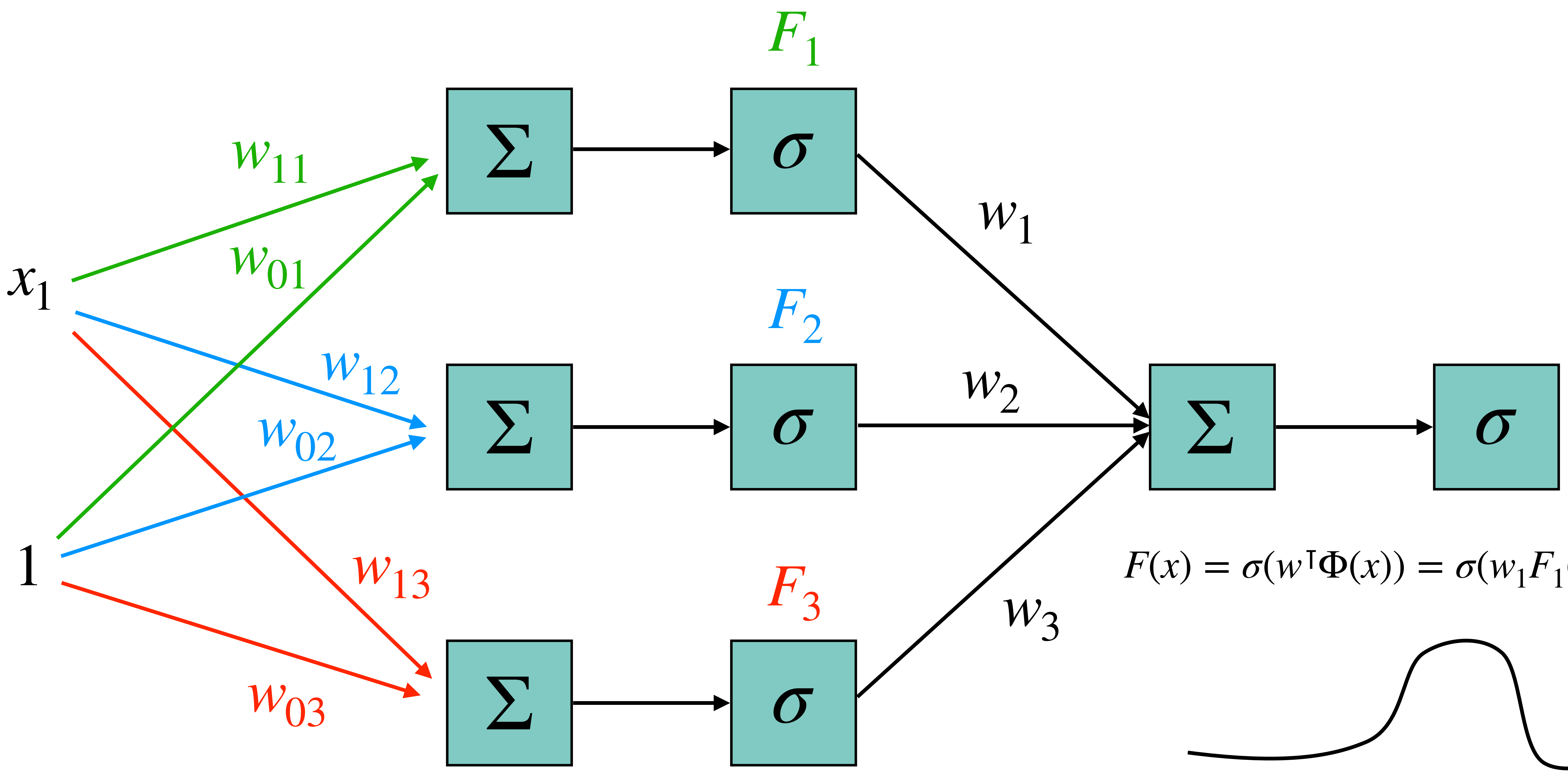
$F(x) = T(w^\intercal \Phi(x)) = T(w_1 F_1(x) + w_2 F_2(x) + w_3 F_3(x) + w_4)$

regression

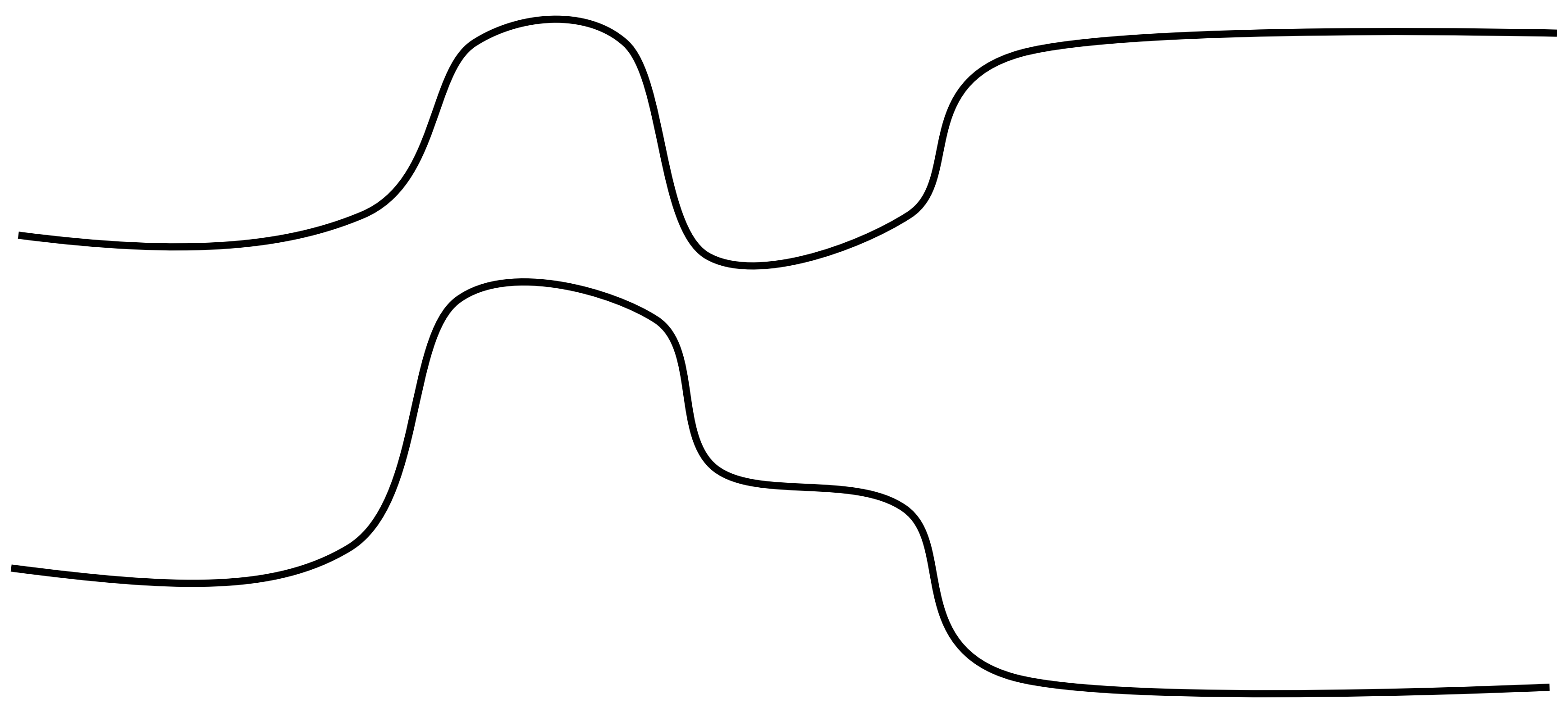
$F(x) = w^\intercal \Phi(x)$



Multi-Layer Perceptron (MLP)



$F(x) = \sigma(w^\intercal \Phi(x)) = \sigma(w_1 F_1(x) + w_2 F_2(x) + w_3 F_3(x) + w_4)$

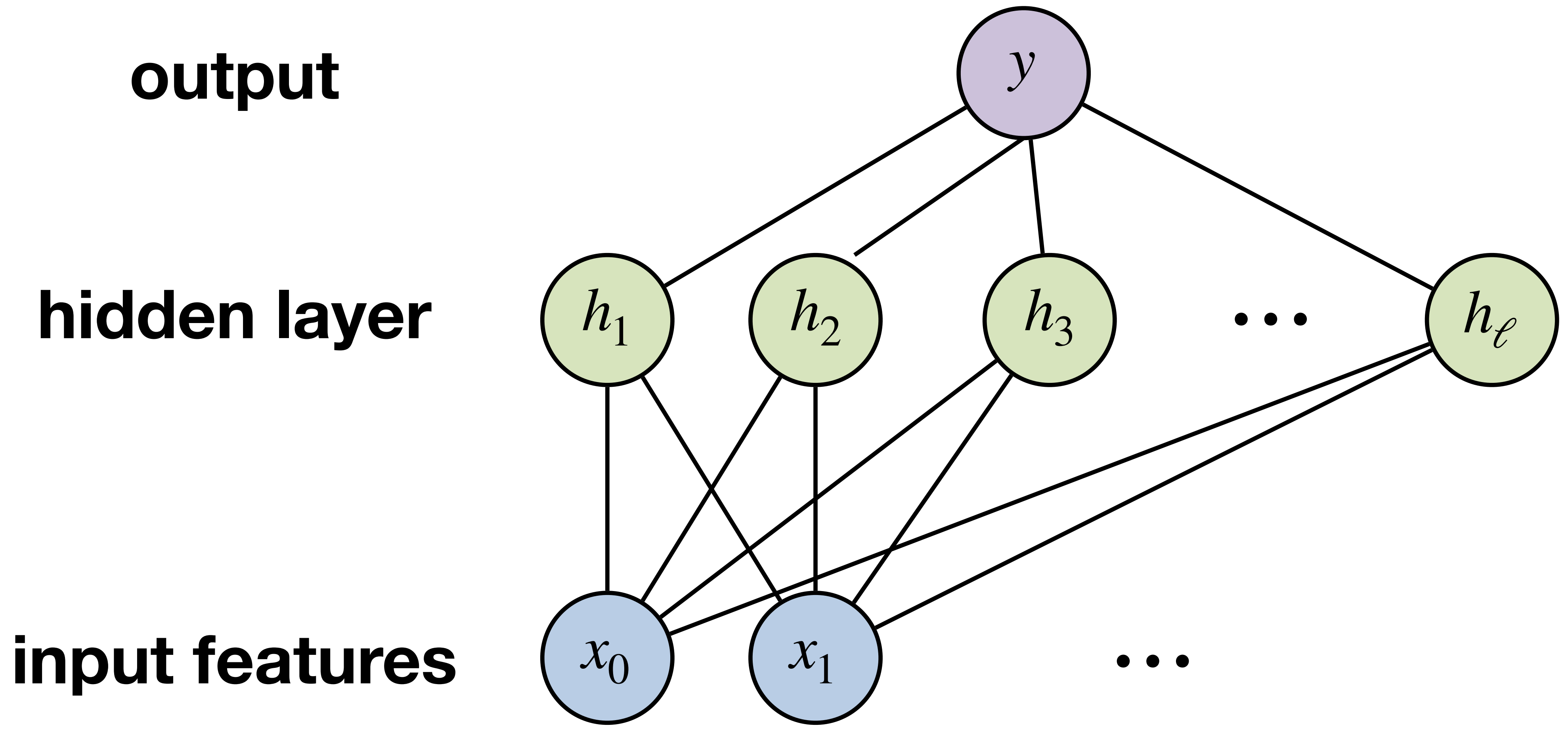


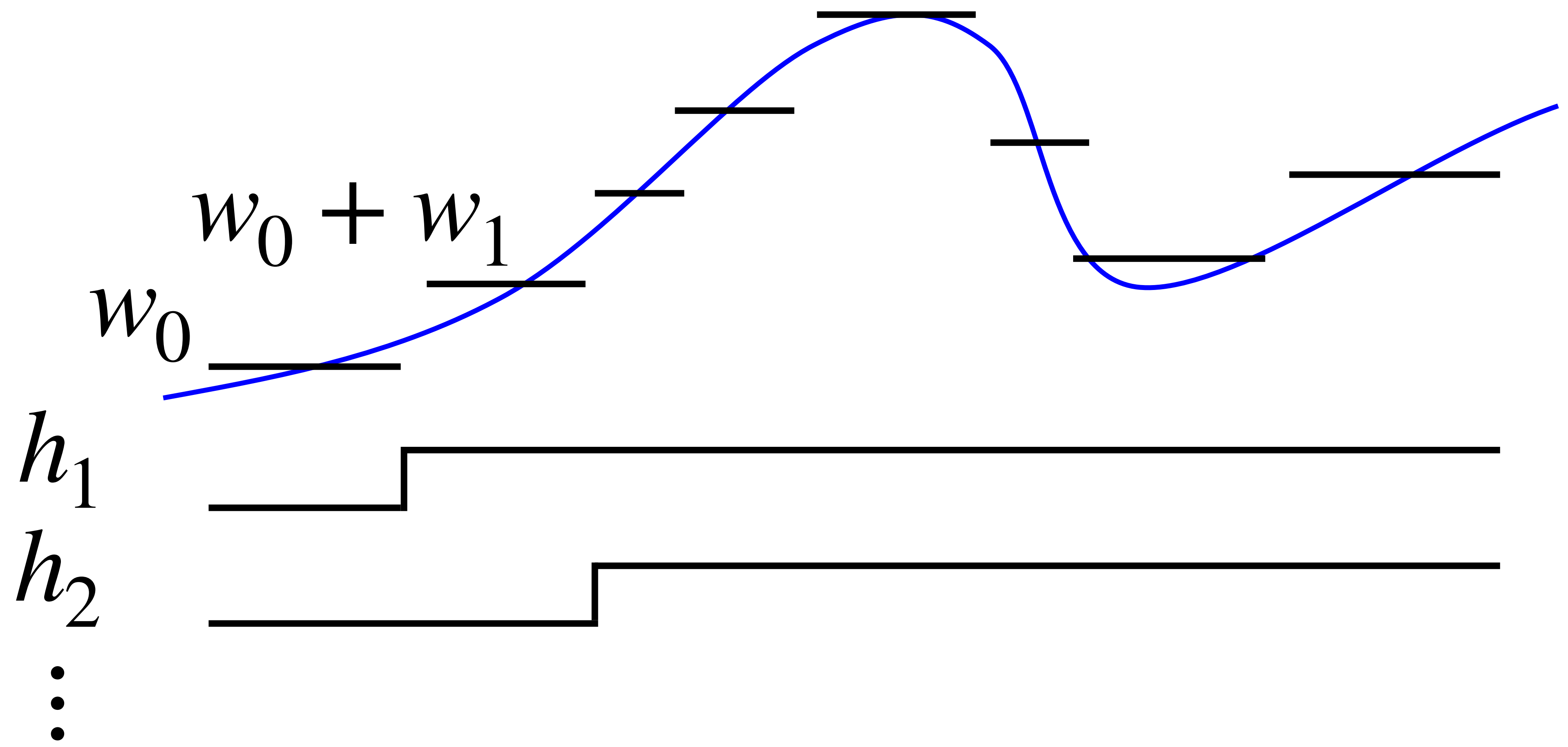
regression

$F(x) = w^\intercal \Phi(x)$

MLPs: properties

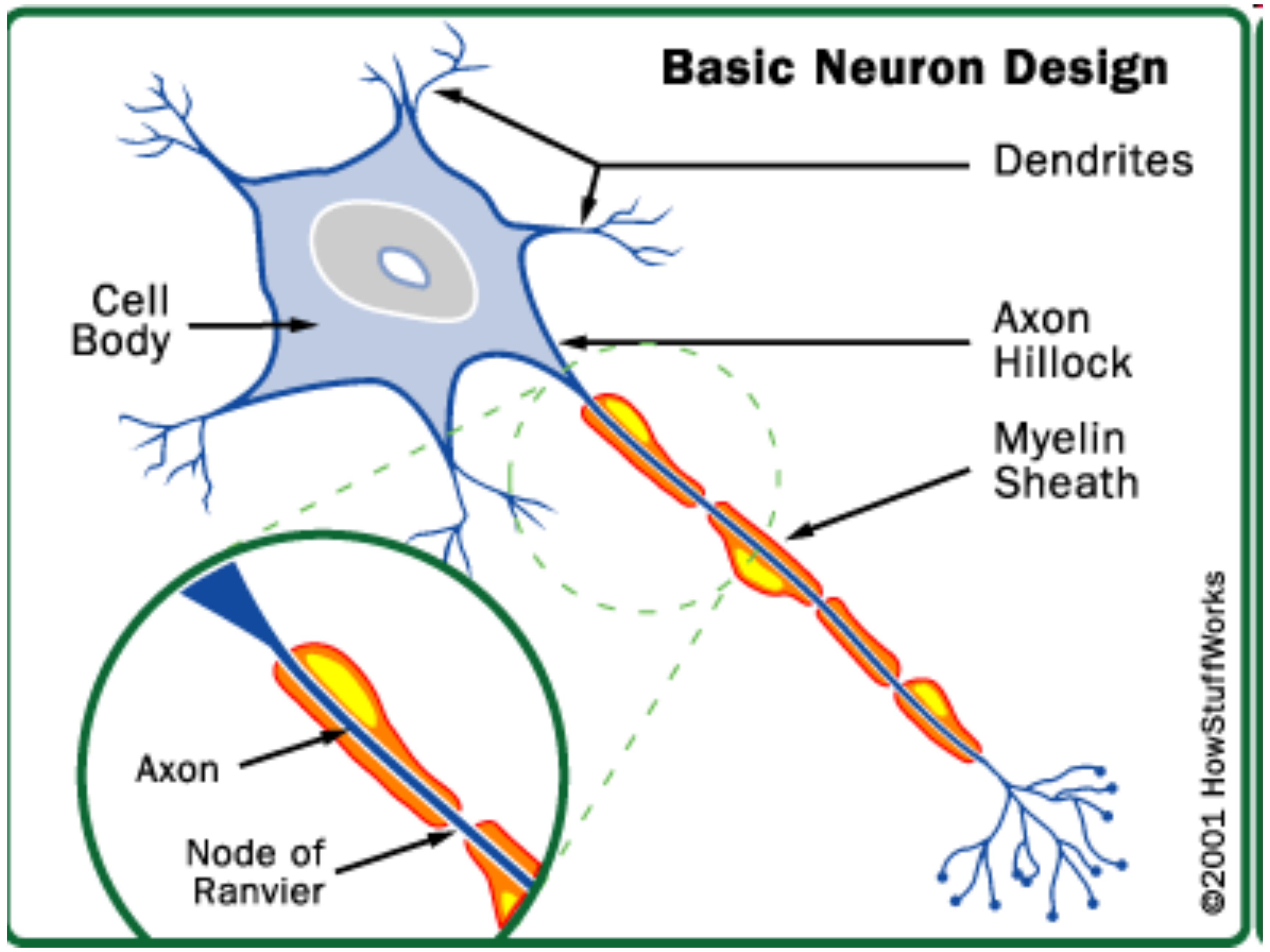
- Simple building blocks
  - Each unit is a perceptron: linear response → non-linear activation
- MLPs are universal approximators:
  - Can approximate any function arbitrarily well, with enough units





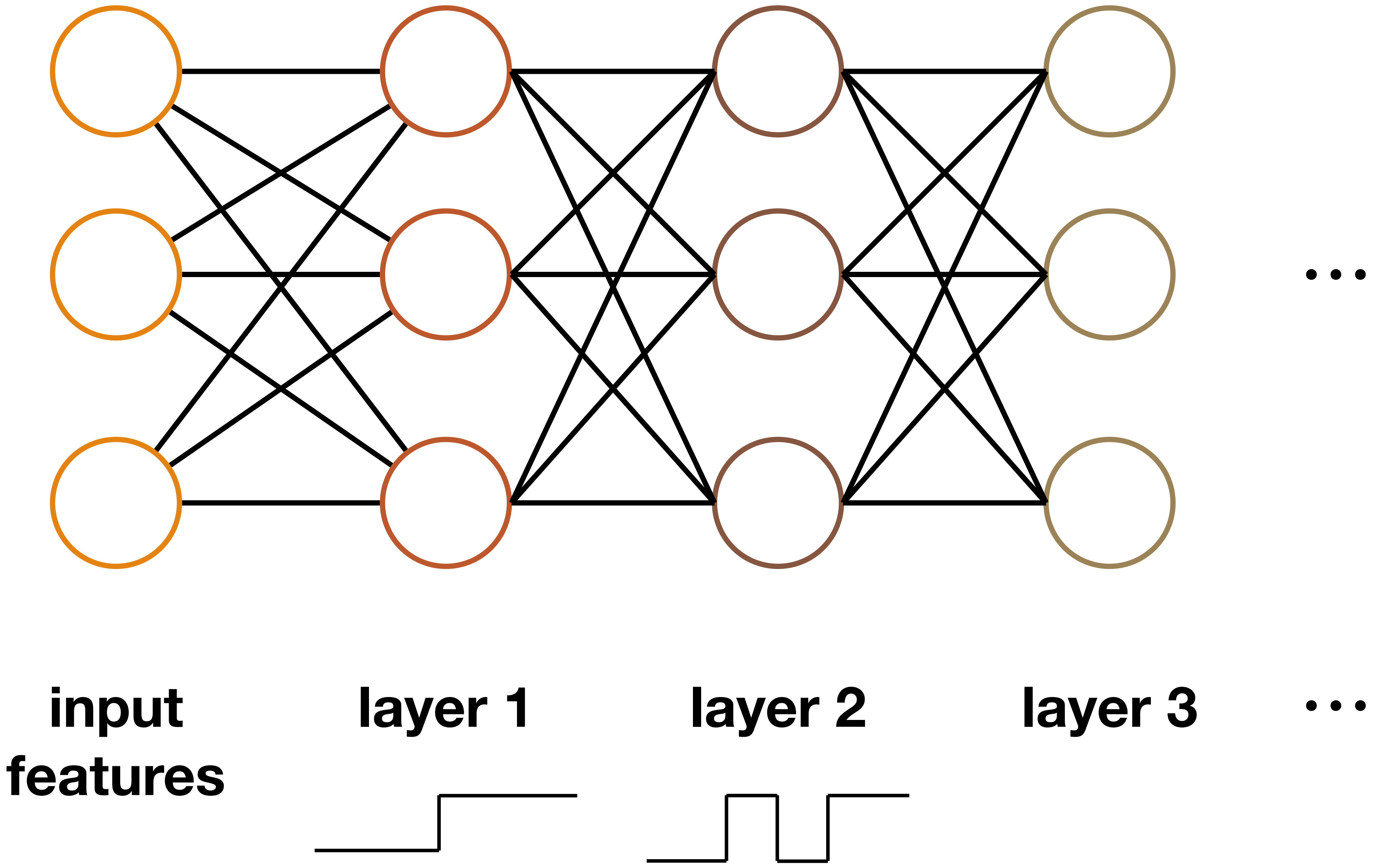
“Neural” Networks

- Biologically inspired
- Neurons:
  - “Simple” cells
  - Dendrites take input voltage
  - Cell body “weights” inputs
  - Axons “fire” voltage
  - Synapses connect to other cells



Deep Neural Networks (DNNs)

- Layers of perceptrons can be stacked deeply
  - Deep architectures are subject of much current research



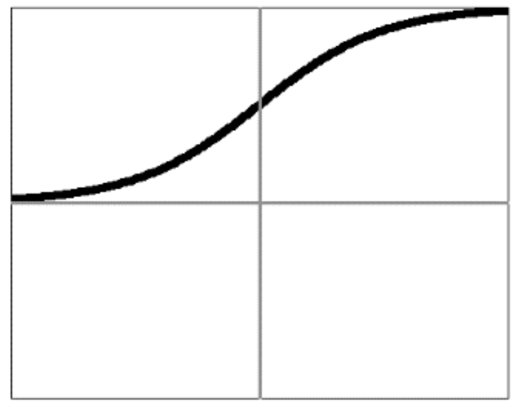

```
r1 = w[0].T @ x + b[0]    # linear response
h1 = sig(r1)              # activation function

r2 = w[1].T @ h1 + b[1]   # linear response
h2 = sig(r2)              # activation function

                          # ...
```

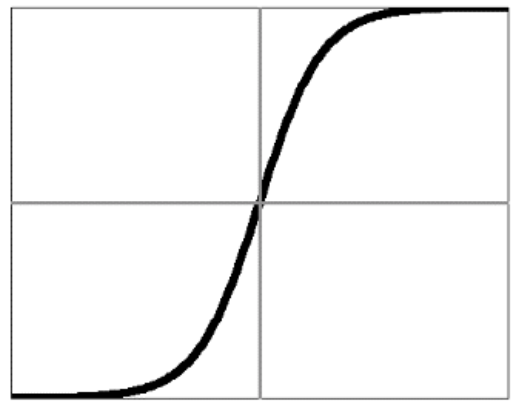
Activation functions

- Logistic $\sigma(z) = \dfrac{1}{1 + \exp(-z)}$



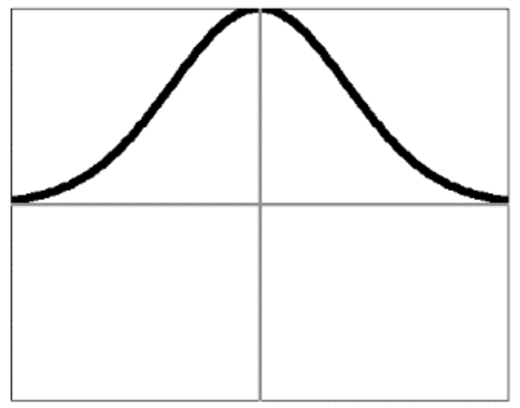
$\sigma'(z) = \sigma(z)(1 - \sigma(z))$

- Hyperbolic tangent $\sigma(z) = \dfrac{1 - \exp(-2z)}{1 + \exp(-2z)}$



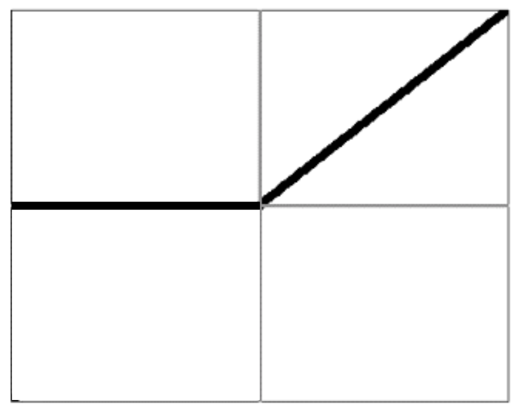
$\sigma'(z) = 1 - \sigma^2(z)$

- Gaussian $\sigma(z) = \exp(-\tfrac{1}{2}z^2)$



$\sigma'(z) = -z\sigma(z)$

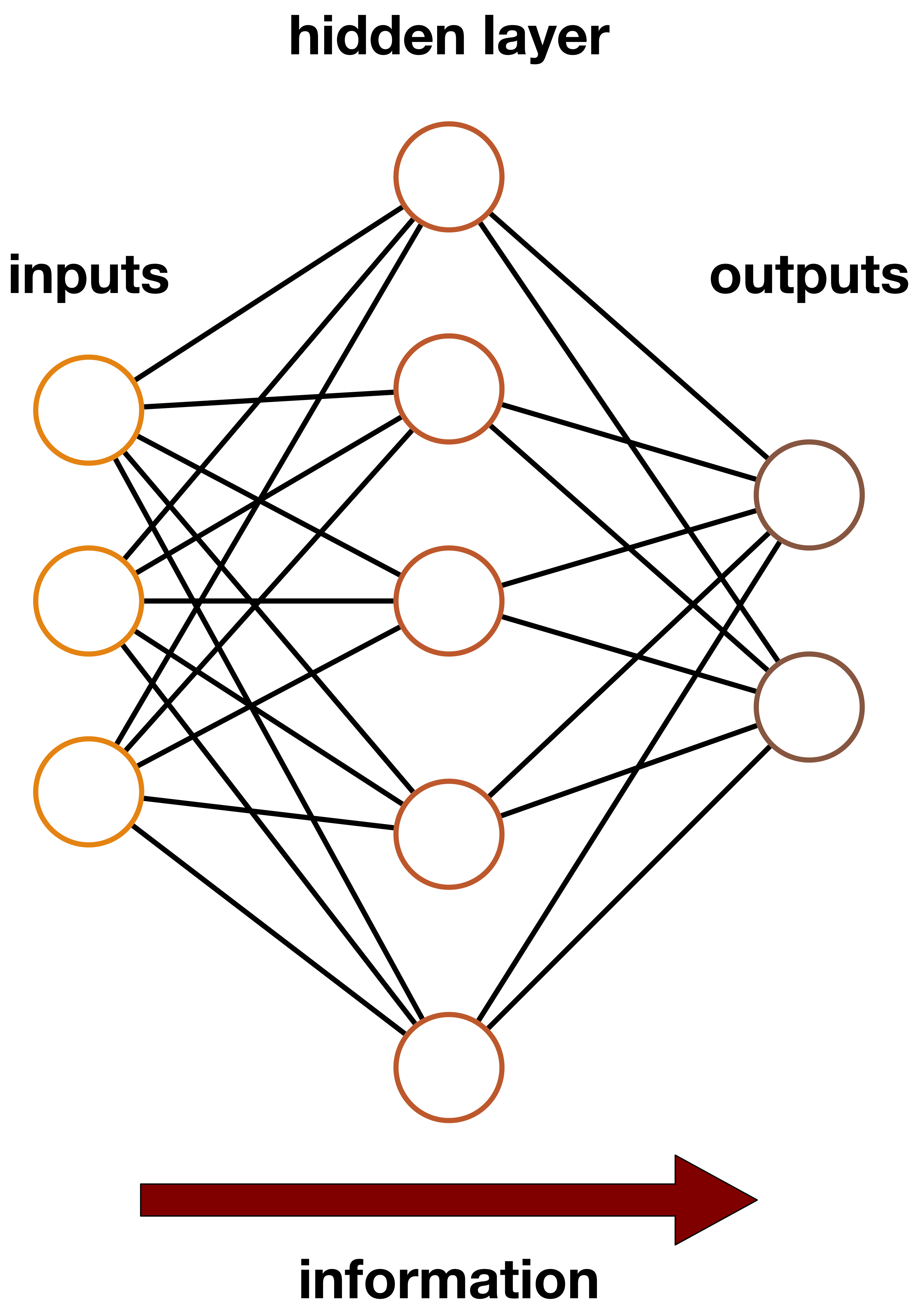
- Rectified linear (ReLU) $\sigma(z) = \max(0,z)$



$\sigma'(z) = \delta[z > 0]$

Feed-forward (FF) networks

- Information flow in feed-forward (FF) networks:
  - Inputs → shallow layers → deeper layers → outputs
  - Alternative: recurrent NNs (information loops back)
- Multiple outputs $\implies$ efficiency:
  - Shared parameters, less data, less computation
- Multi-class classification:
  - One-hot labels $y = [0 \quad 0 \quad 1 \quad 0 \quad \cdots]$
  - Multilogistic regression (softmax): $\hat{y}_c = \dfrac{\exp(h_c)}{\sum_{\bar{c}} \exp(h_{\bar{c}})}$



Today's lecture

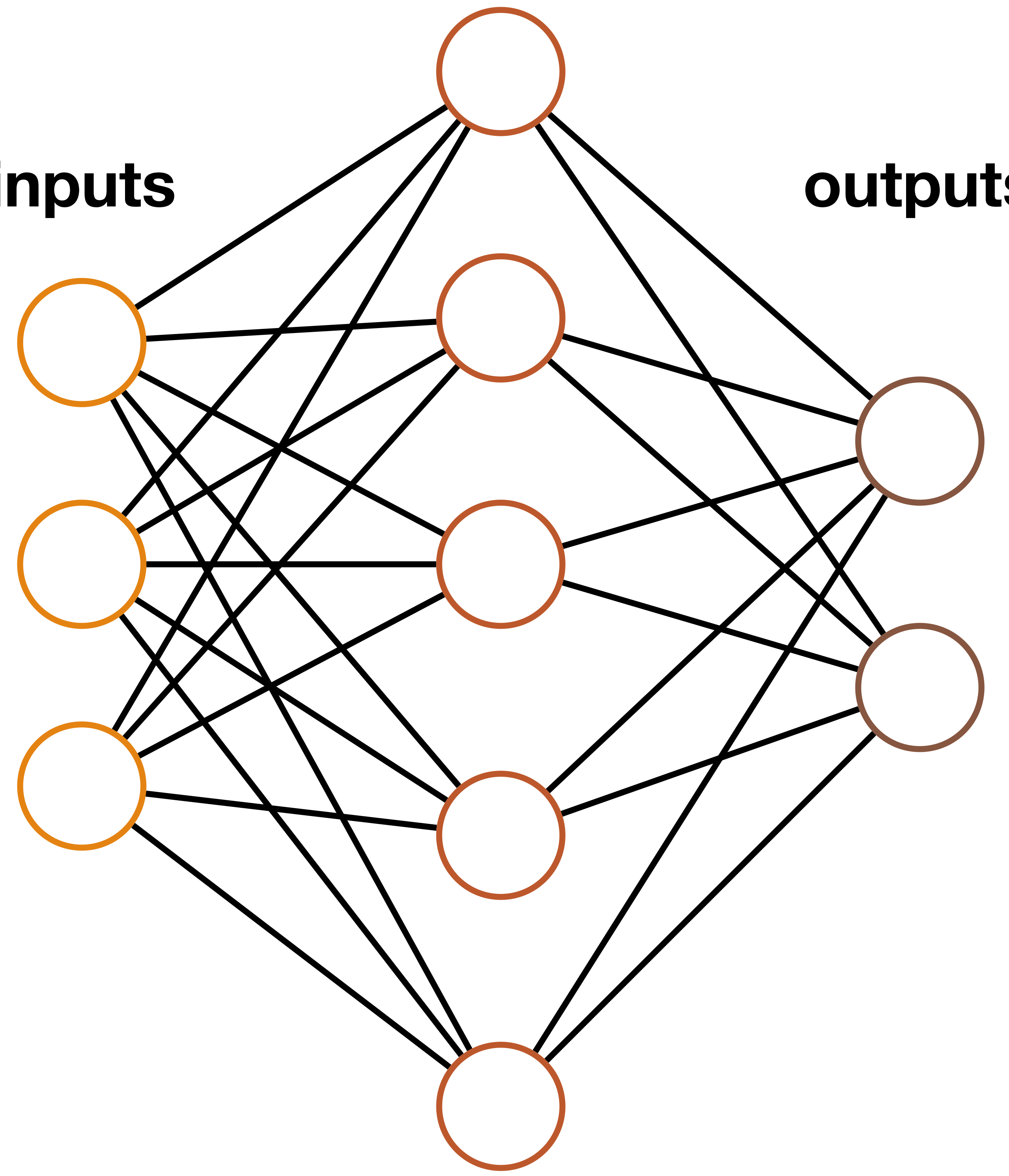
Multilayer Perceptrons

Backpropagation

Advanced Neural Networks

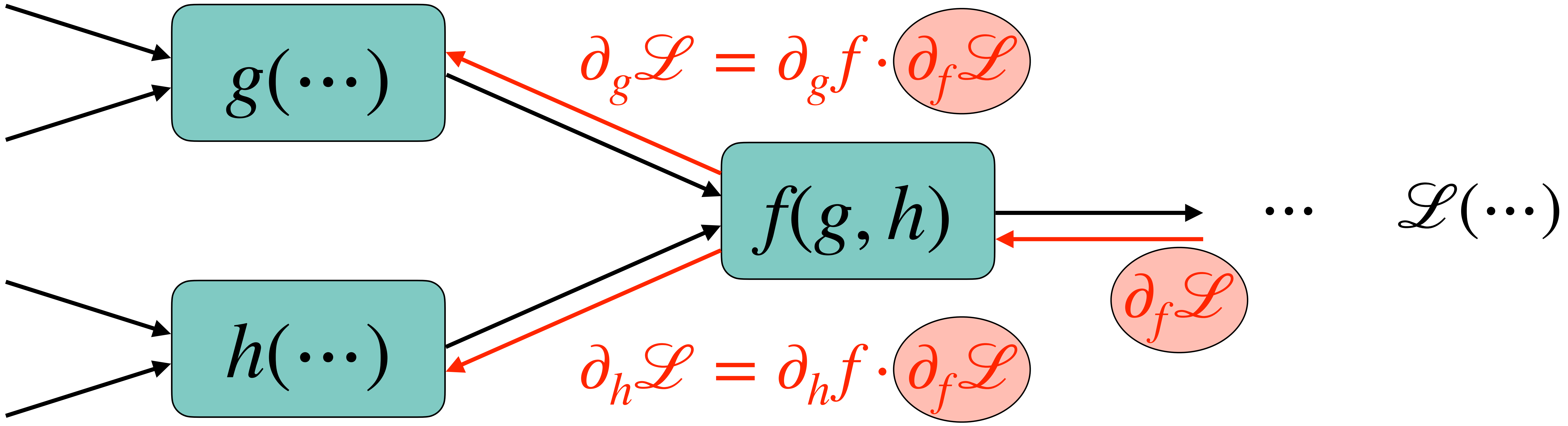
Training MLPs

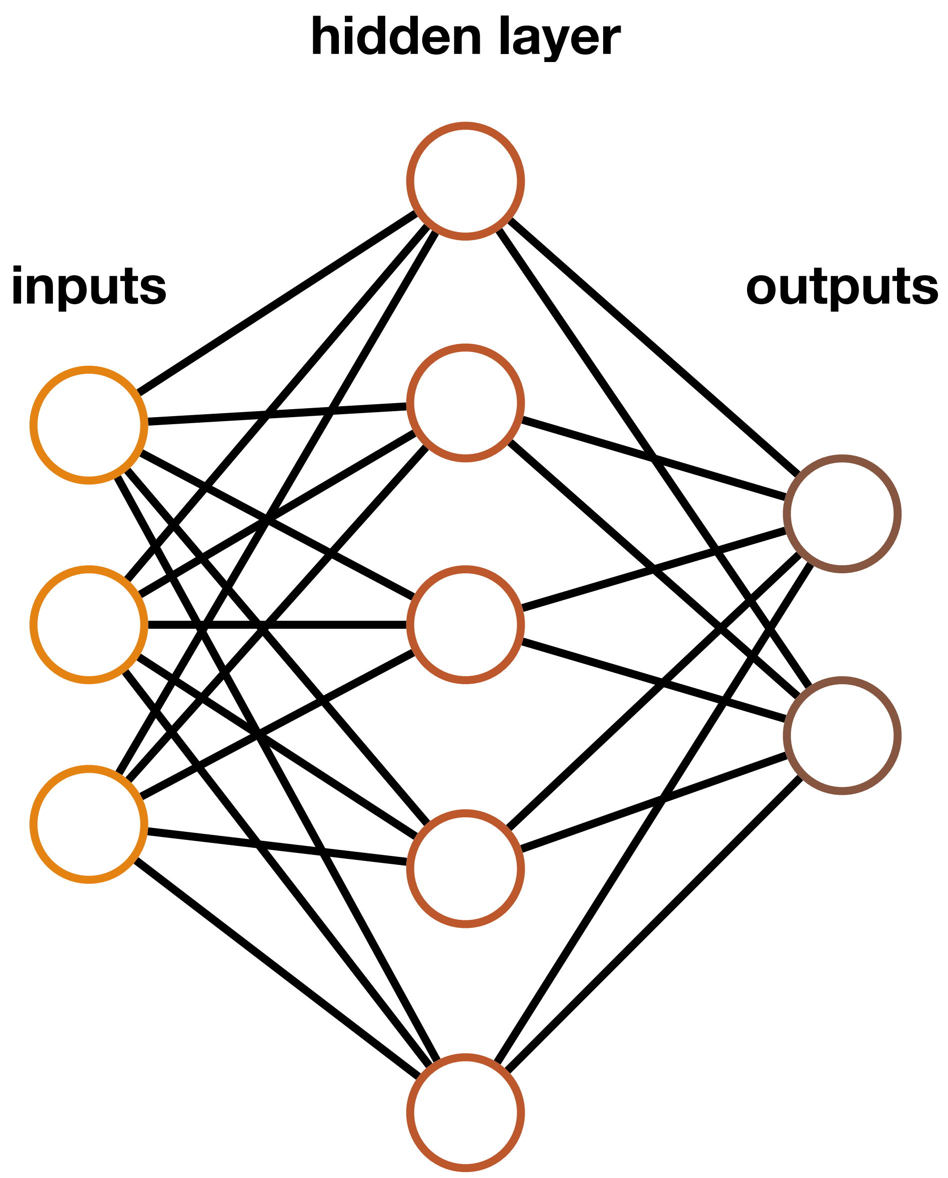
- Observe instance x, target y
- Feed x forward through NN = prediction $\hat{y}$
- Loss = $\ell_w(y, \hat{y}) = (y - \hat{y})^2$ (or another loss function)
- How should we update the weights?
- Single layer:
    - Use differentiable activation function, e.g. logistic
    - (Stochastic) Gradient Descent = logistic regression



Gradient computation

- MLPs are function compositions of single layers
  - Apply chain rule:

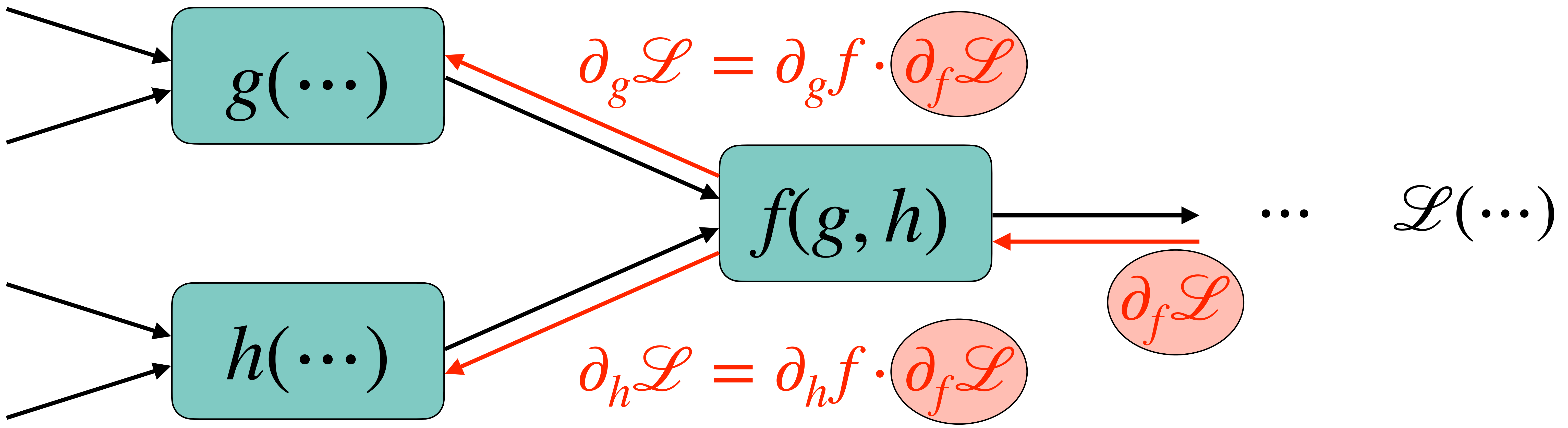


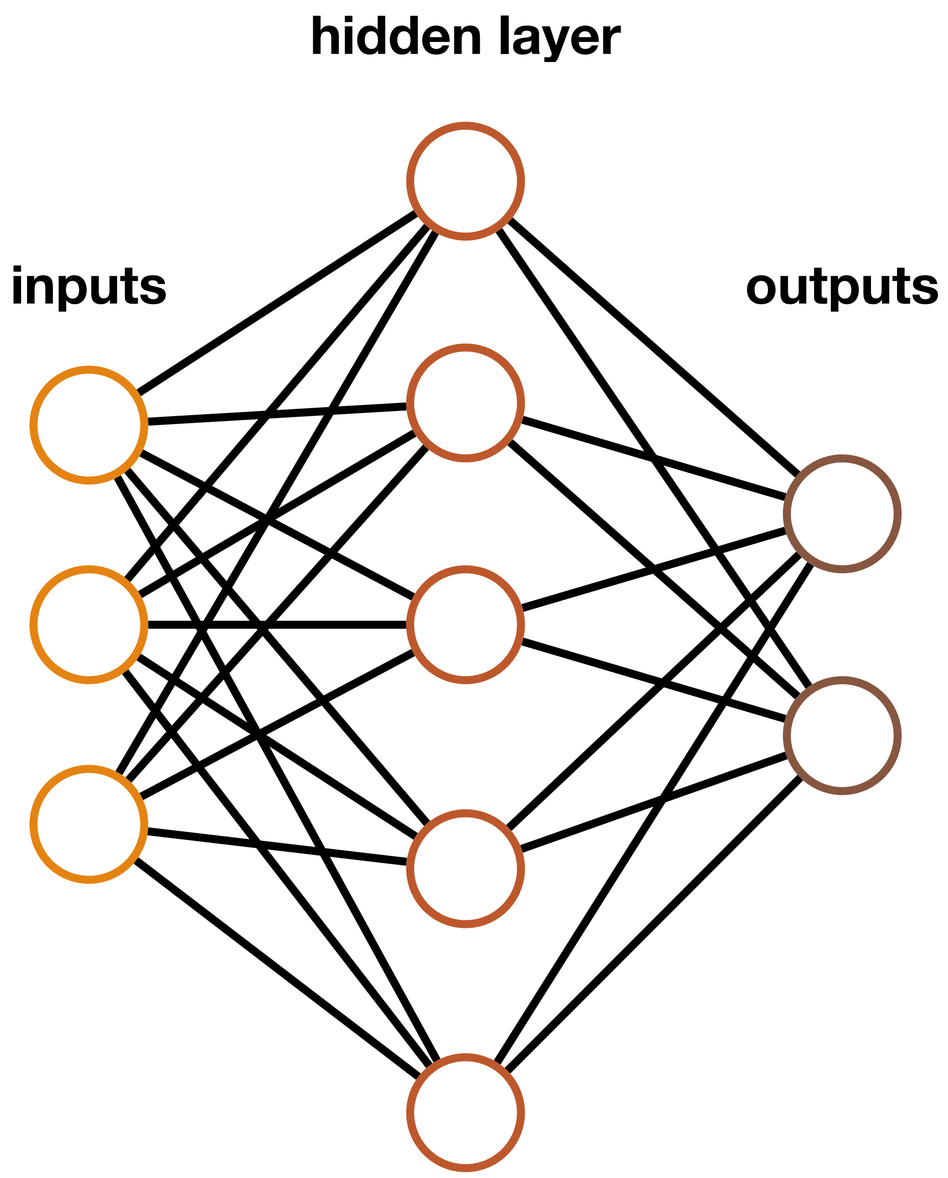


- Backpropagation = chain rule + dynamic programming to avoid repetitions

Gradient computation

- MLPs are function compositions of single layers
  - Apply chain rule:





example: $f(g,h) = \sigma(g+h) \implies \partial_g f = f(1-f)$

$\implies$ **reuse** f **from the forward pass**

- Backpropagation = chain rule + dynamic programming to avoid repetitions

Backpropagation

- Use chain rule (efficiently) to propagate gradients:
  - w_{ij}^{ℓ} = unit i in layer $(\ell - 1) \to$ unit j in layer ℓ

$$\partial_{w_{jk}^L} \mathscr{L}_w = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'}) \partial_{w_{jk}^L} \hat{y}_{k'}$$

same as logistic MSE regression

β_k^L

$$= \boxed{-2(y_k - \hat{y}_k)\sigma'(r_k^L)} h_j^{L-1}$$

$$\partial_{w_{ij}^{L-1}} \mathscr{L}_w = \sum_k -2(y_k - \hat{y}_k) \partial_{w_{ij}^{L-1}} \hat{y}_k$$

$$= \sum_k -2(y_k - \hat{y}_k)\sigma'(r_k^L) w_{jk}^L \partial_{w_{ij}^{L-1}} h_j^{L-1}$$

β_k^L

$$= \sum_k \boxed{-2(y_k - \hat{y}_k)\sigma'(r_k^L)} w_{jk}^L \sigma'(r_j^{L-1}) h_i^{L-2}$$

Forward pass:

loss function:

$$\mathscr{L}_w = \sum_k (y_k - \hat{y}_k)^2$$

output layer:

$$\hat{y}_k = \sigma(r_k^L) = \sigma\left(\sum_j h_j^{L-1} w_{jk}^L\right)$$

hidden layer:

$$h_j^{L-1} = \sigma(r_j^{L-1}) = \sigma\left(\sum_i h_i^{L-2} w_{ij}^{L-1}\right)$$

Backpropagation

- Use chain rule (efficiently) to propagate gradients:
  - w_{ij}^{ℓ} = unit i in layer $(\ell - 1) \to$ unit j in layer ℓ

$$\partial_{w_{jk}^L} \mathscr{L}_w = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'}) \partial_{w_{jk}^L} \hat{y}_{k'}$$

same as logistic MSE regression

$$= \underbrace{\boxed{-2(y_k - \hat{y}_k)\sigma'(r_k^L)}}_{\beta_k^L} h_j^{L-1}$$

```
b2 = -2 * (y - y_hat) * d_sig(r2)      # beta2: [K]
g2 = h.T @ b2  # w2_jk gradient: [J,1]@[1,K]=[JxK]

b1 = d_sig(r1) * w2 @ b2 # beta1: [J]*[JK]@[K]=[J]
g1 = x.T @ b1  # w1_ij gradient: [I,1]@[1,J]=[IxJ]
```

$$\partial_{w_{ij}^{L-1}} \mathscr{L}_w = \sum_k -2(y_k - \hat{y}_k) \partial_{w_{ij}^{L-1}} \hat{y}_k$$

$$= \sum_k -2(y_k - \hat{y}_k)\sigma'(r_k^L) w_{jk}^L \partial_{w_{ij}^{L-1}} h_j^{L-1}$$

$$= \sum_k \underbrace{\boxed{-2(y_k - \hat{y}_k)\sigma'(r_k^L)}}_{\beta_k^L} w_{jk}^L \sigma'(r_j^{L-1}) h_i^{L-2}$$

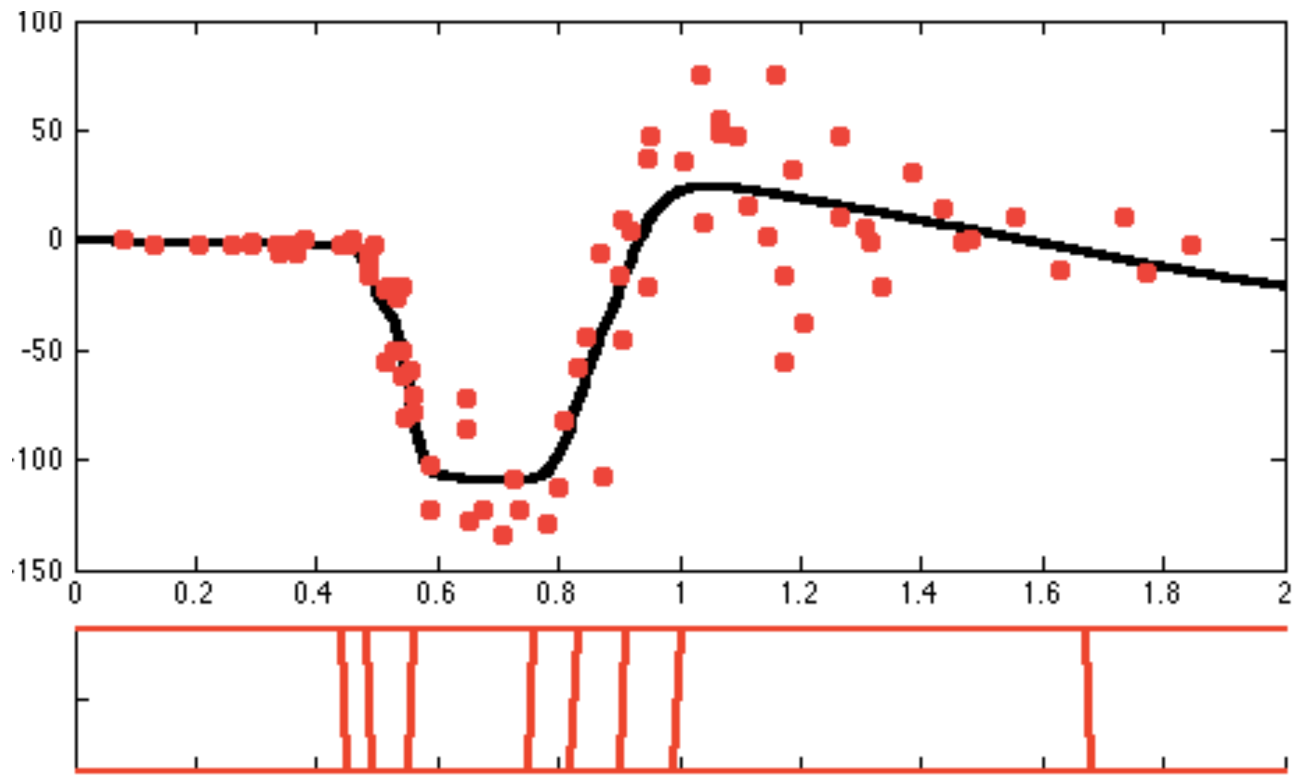
Example: MCycle data (regression)

- Train NN model, 2 layer
  - 1 input features = 1 input units
  - 10 hidden units
  - 1 target = 1 output units
  - Logistic sigmoid activation for hidden layer, linear for output layer

data

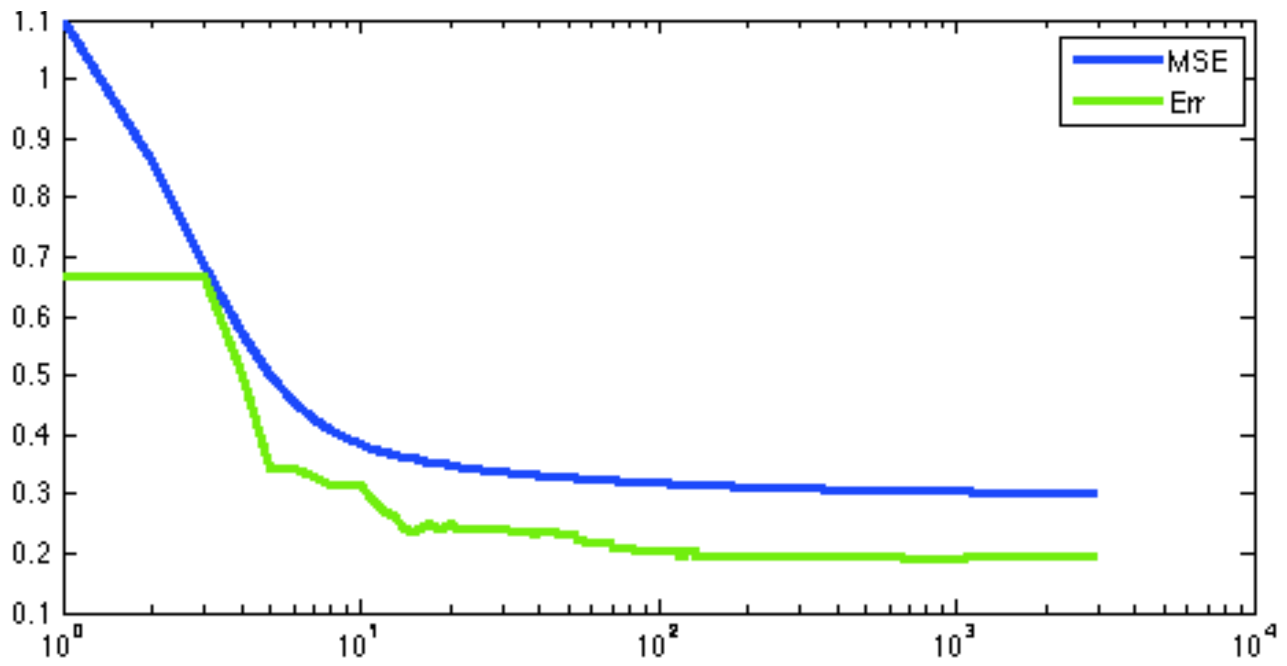
prediction

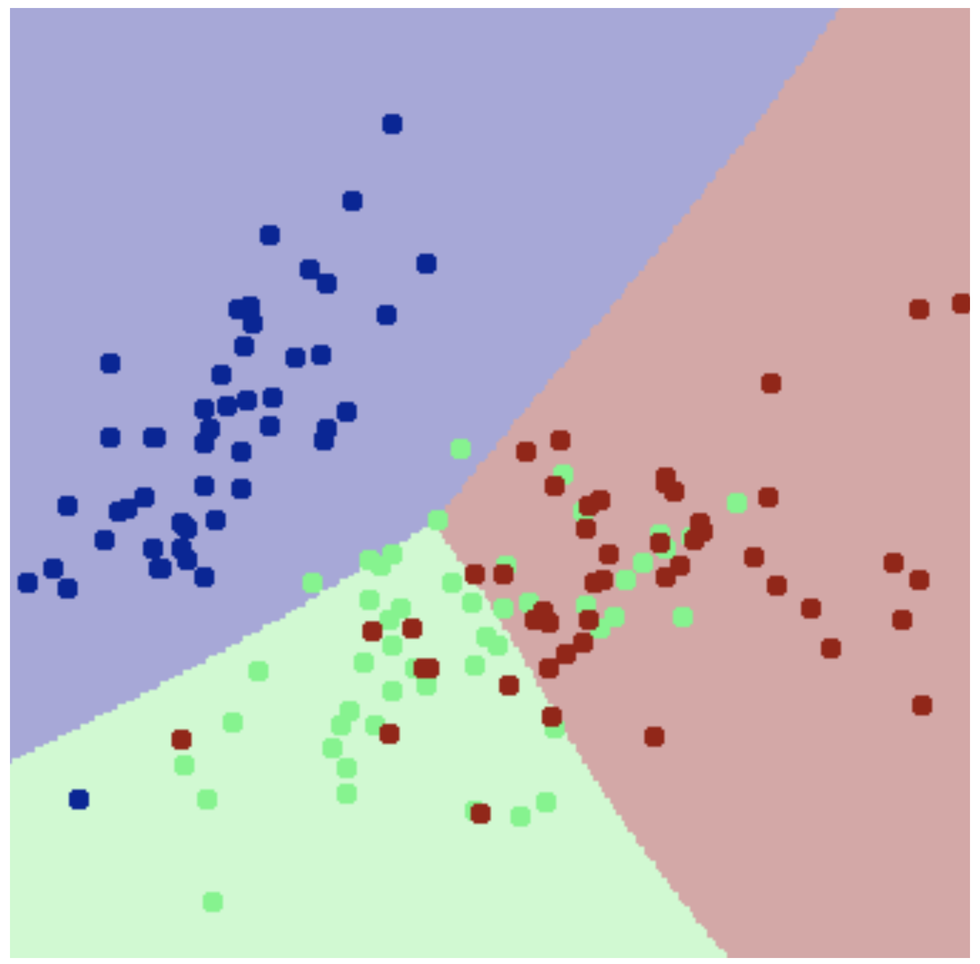
response of hidden nodes (overlaid)



Example: Iris data (classification)

- Train NN model, 2 layer
  - 2 input features = 2 input units, 10 hidden units
  - 3 classes = 3 output units (e.g., $y = [0 \quad 0 \quad 1]$)
  - Logistic sigmoid activation functions
  - Optimize MSE of predictions using stochastic gradient





Demo

- http://playground.tensorflow.org/

Today's lecture

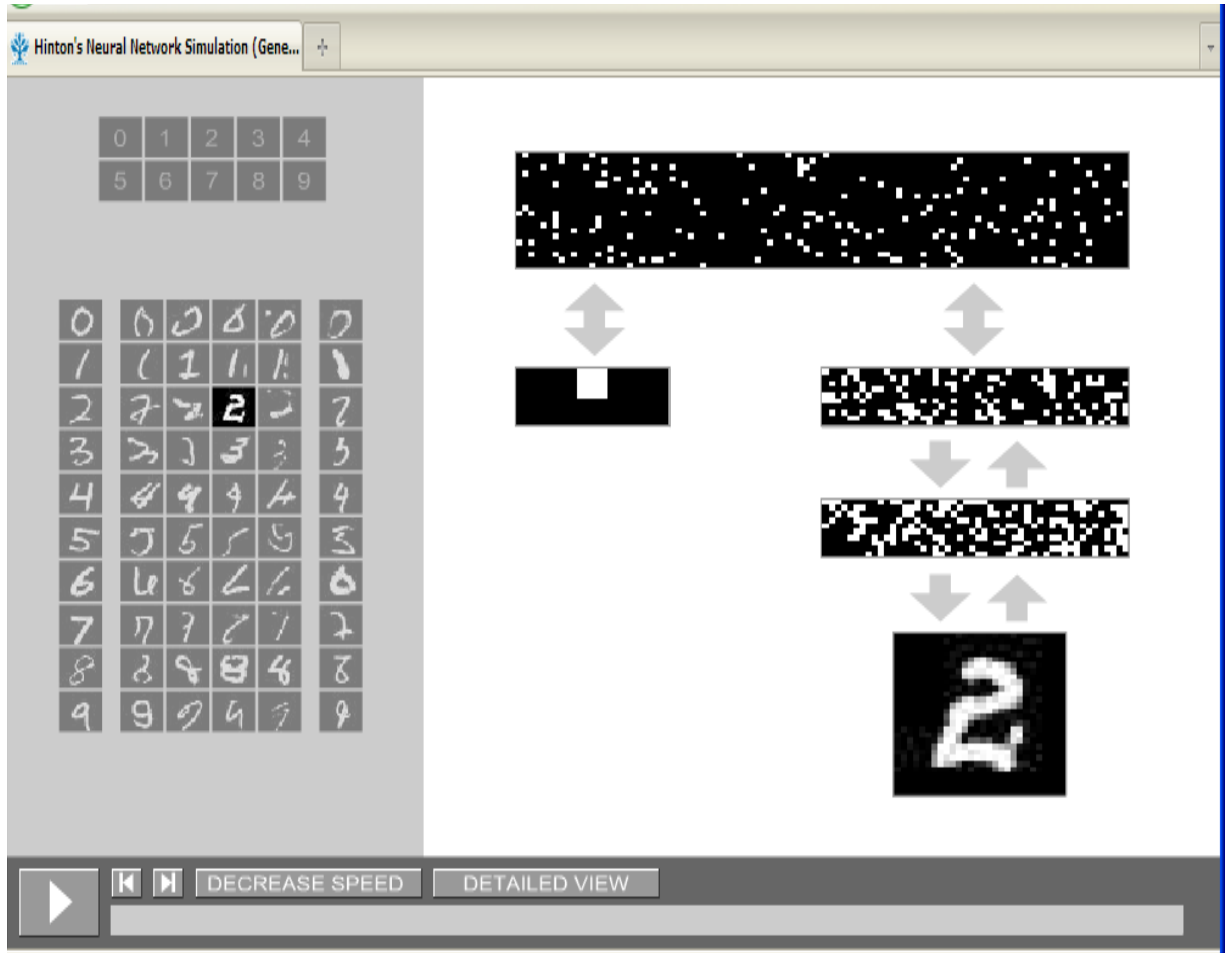
Multilayer Perceptrons

Backpropagation

Advanced Neural Networks

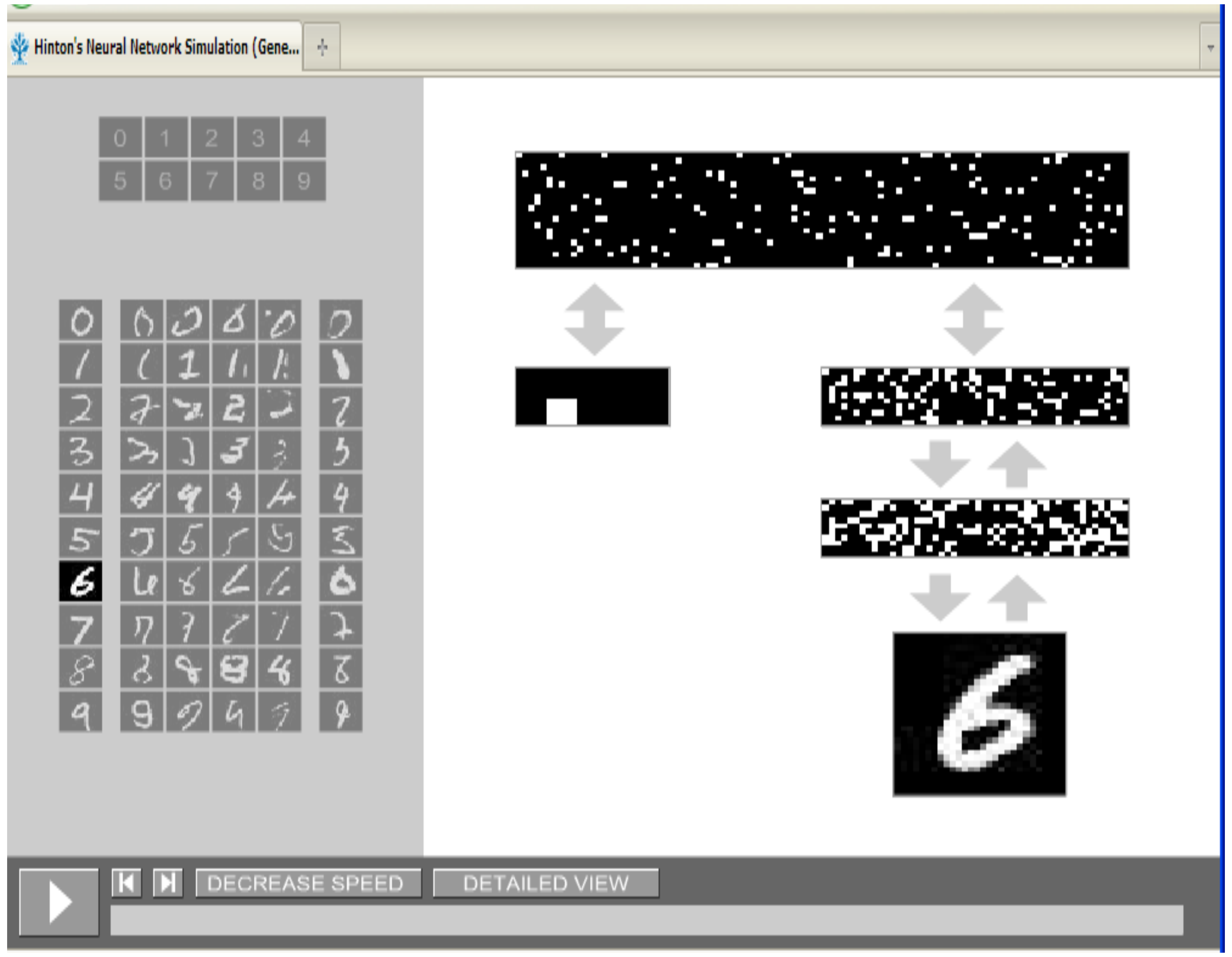
MLPs in practice

- Example: Deep belief nets
  - Handwriting recognition
  - 784 pixels ⇔ 500 mid layer ⇔ 500 high ⇔ 2000 top ⇔ 10 labels



MLPs in practice

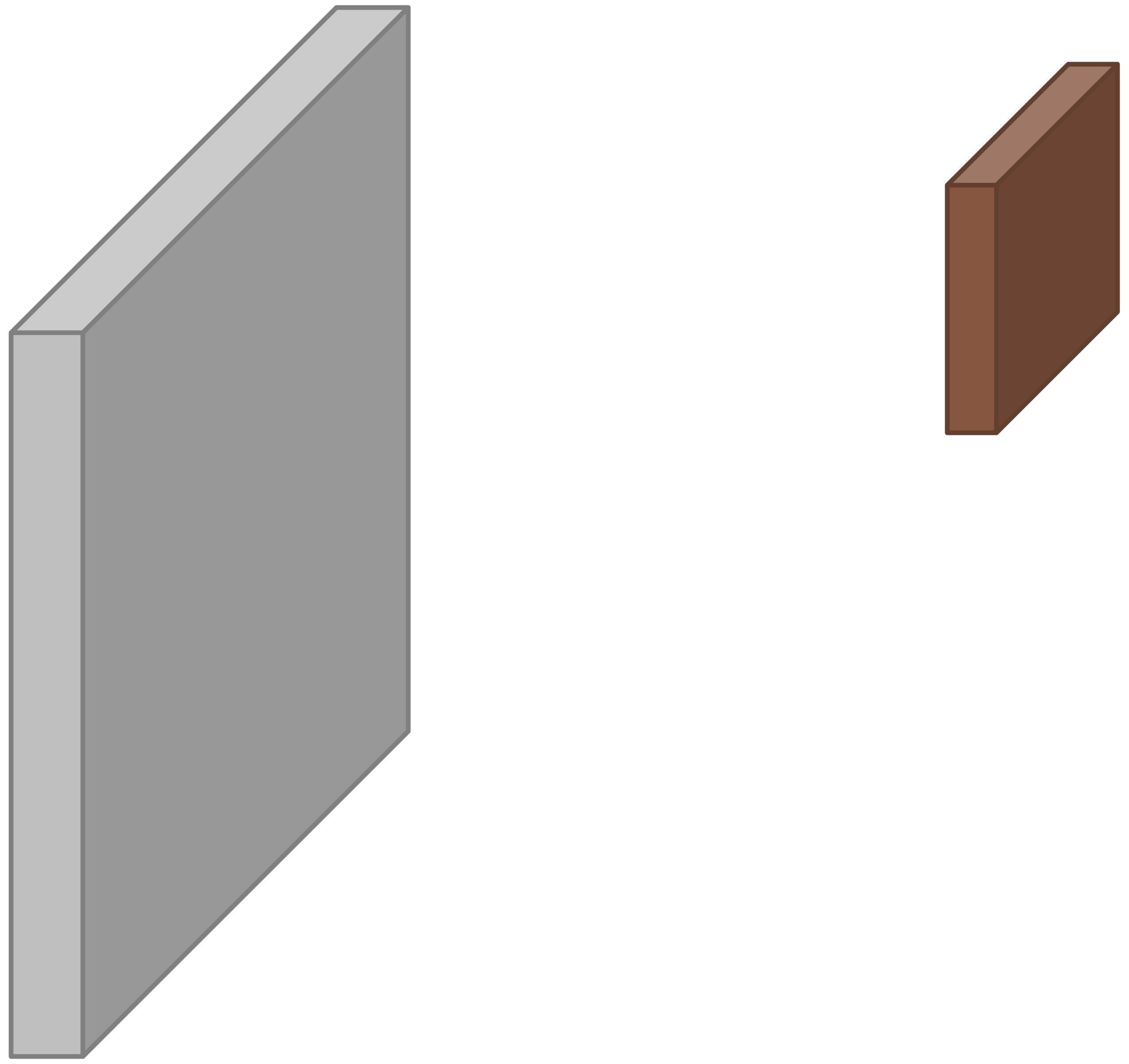
- Example: Deep belief nets
  - Handwriting recognition
  - 784 pixels ⇔ 500 mid layer ⇔ 500 high ⇔ 2000 top ⇔ 10 labels



Convolutional Networks (ConvNets)

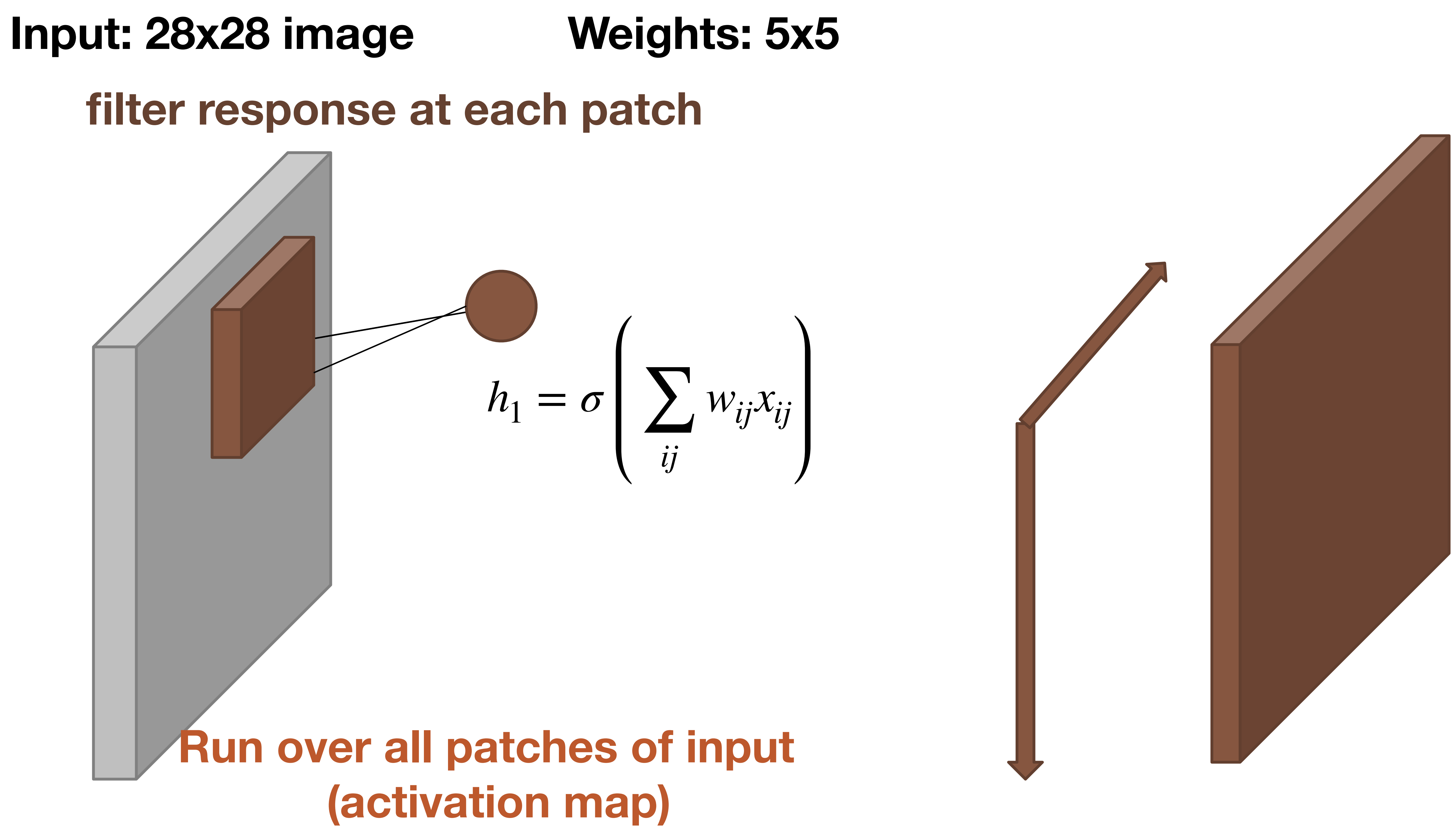
- Group and share weights to use inductive bias:
  - Images are translation invariant

Input: 28x28 image **Weights: 5x5**



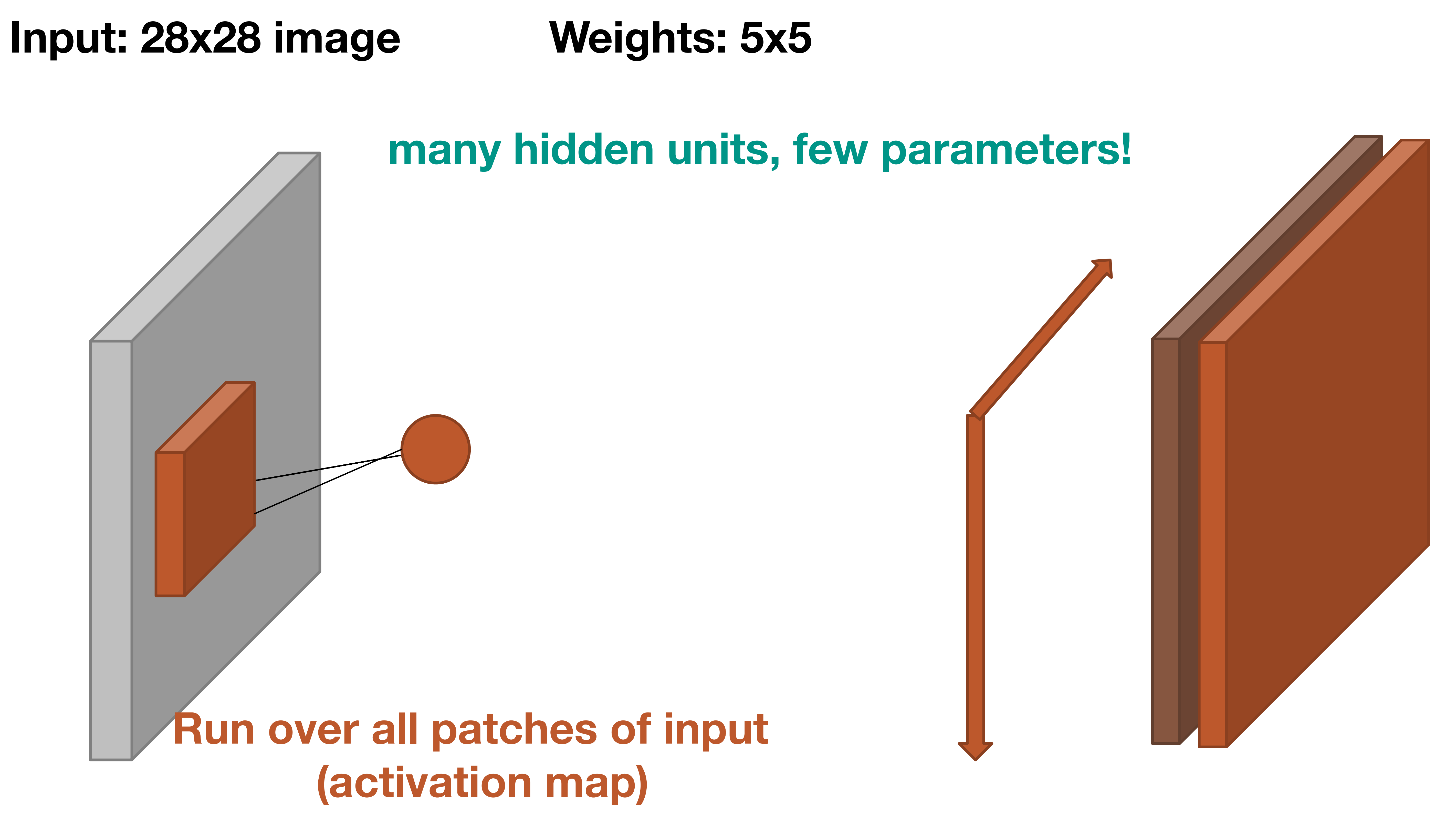
Convolutional Networks (ConvNets)

- Group and share weights to use inductive bias:
    - Images are translation invariant



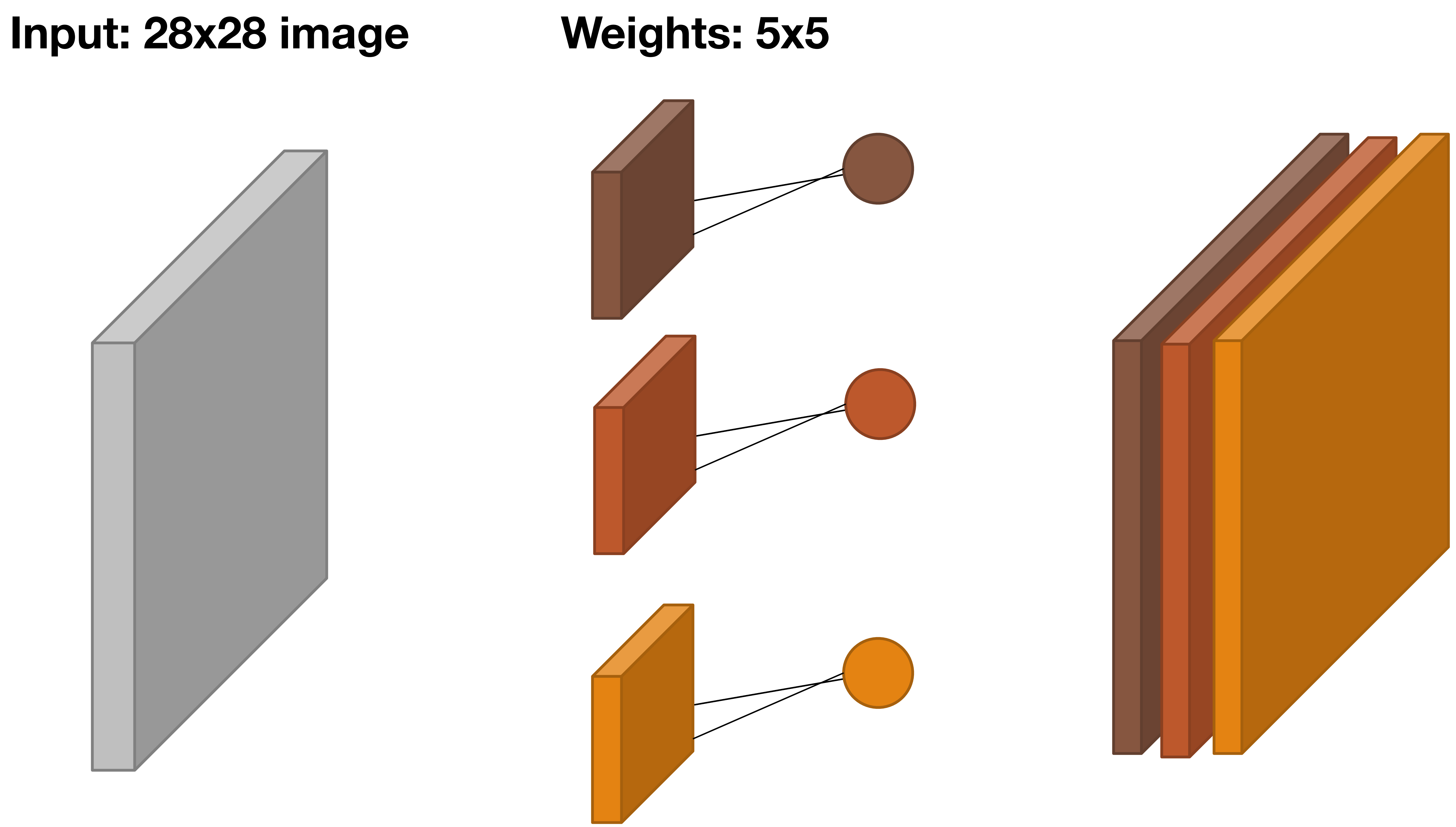
Convolutional Networks (ConvNets)

- Group and share weights to use inductive bias:
  - Images are translation invariant



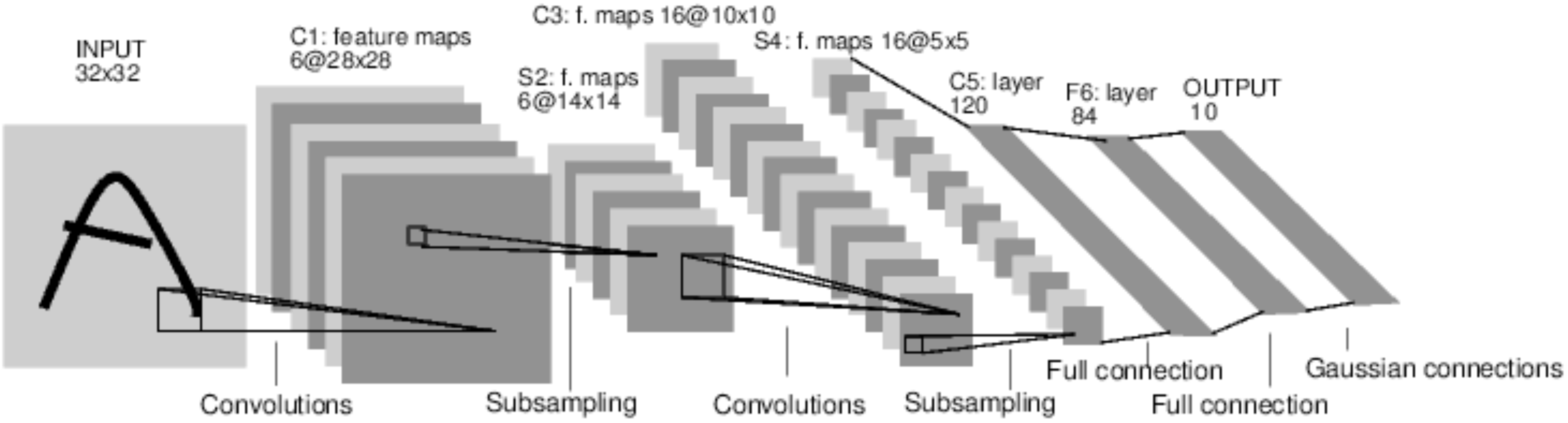
Convolutional Networks (ConvNets)

- Group and share weights to use inductive bias:
  - Images are translation invariant



Convolutional Networks (ConvNets)

- As before: view components as composable building blocks
  - Design deep structure from parts
    - Convolutional layers
    - Max-pooling (sub-sampling) layers
    - Densely connected layers



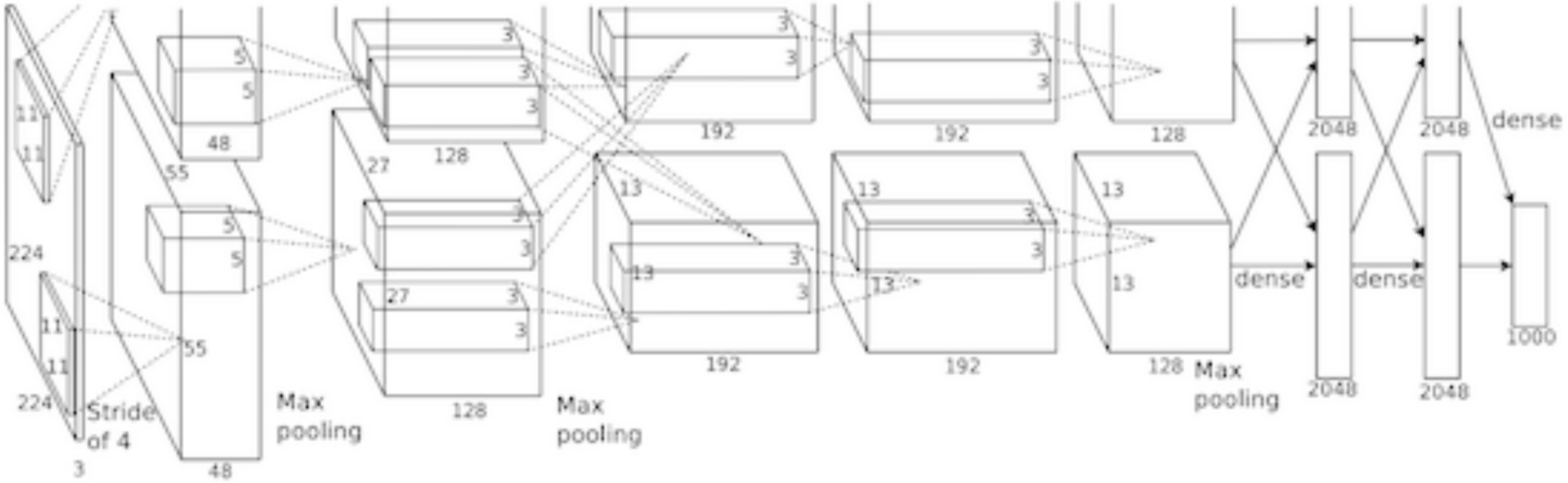
Example: AlexNet

- Deep NN model for ImageNet classification
  - 650k units; 60m parameters
  - 1m data; 1 week training (GPUs)
  - Can be use pre-trained, or fine-tuned (trained again on new data)

5 convolutional layers

3 dense layers

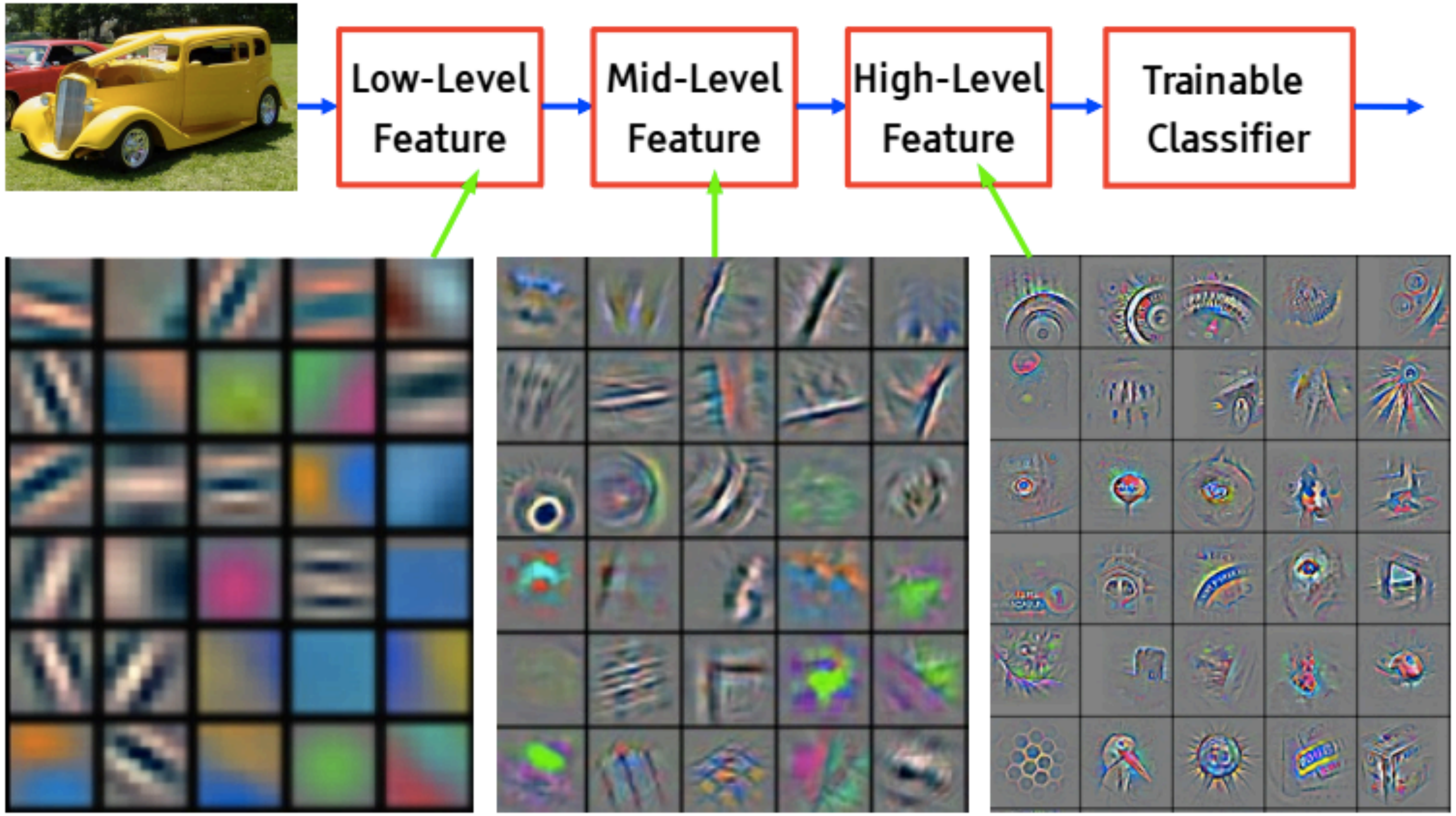
input
224x224x3



output
(1000 classes)

Hidden layers as “features”

- Visualizing a convolutional network’s filters:



Recap

- Multi-layer perceptrons (MLPs); other neural networks architectures
- Composition of simple perceptrons
  - Each just a linear response + non-linear activation
  - Hidden units used to create new features
  - Jointly form universal function approximators: enough units → any function
- Training via backprop = gradient chain rule + dynamic programming
- Much more: deep nets (DNNs), ConvNets, ...

Logistics

project

- Project abstract due Tue, Feb 16

assignments

- Assignment 4 to be published soon