Deep Learning Basics

Computer Vision: CS 600.461/661

Deep Learning Basics

Train functions from data to accomplish visual tasks

Topics:

(1) Deep Neural Network Basics

(2) Fitting DNNs to data

(3) Measures of fit and good training practices

(4) Convolutional Networks

What Is “Deep Learning”?

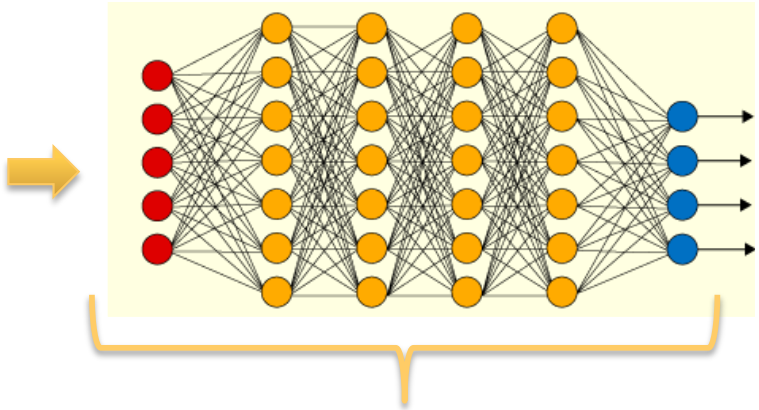
Most Common: Supervised Learning

F()

“Brown Bear”

What Is “Deep Learning”?

Most Common: Supervised Learning



“Brown Bear”

$F(\text{  };W)$

A few 10’s of millions of parameters that are fit to a very large **labelled** data set including lots of brown bears

The Perceptron

$$f\colon \mathbb{R}^d \to \{1,0\}$$

$$f(x) = \begin{cases} 1 & \text{if } \; w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

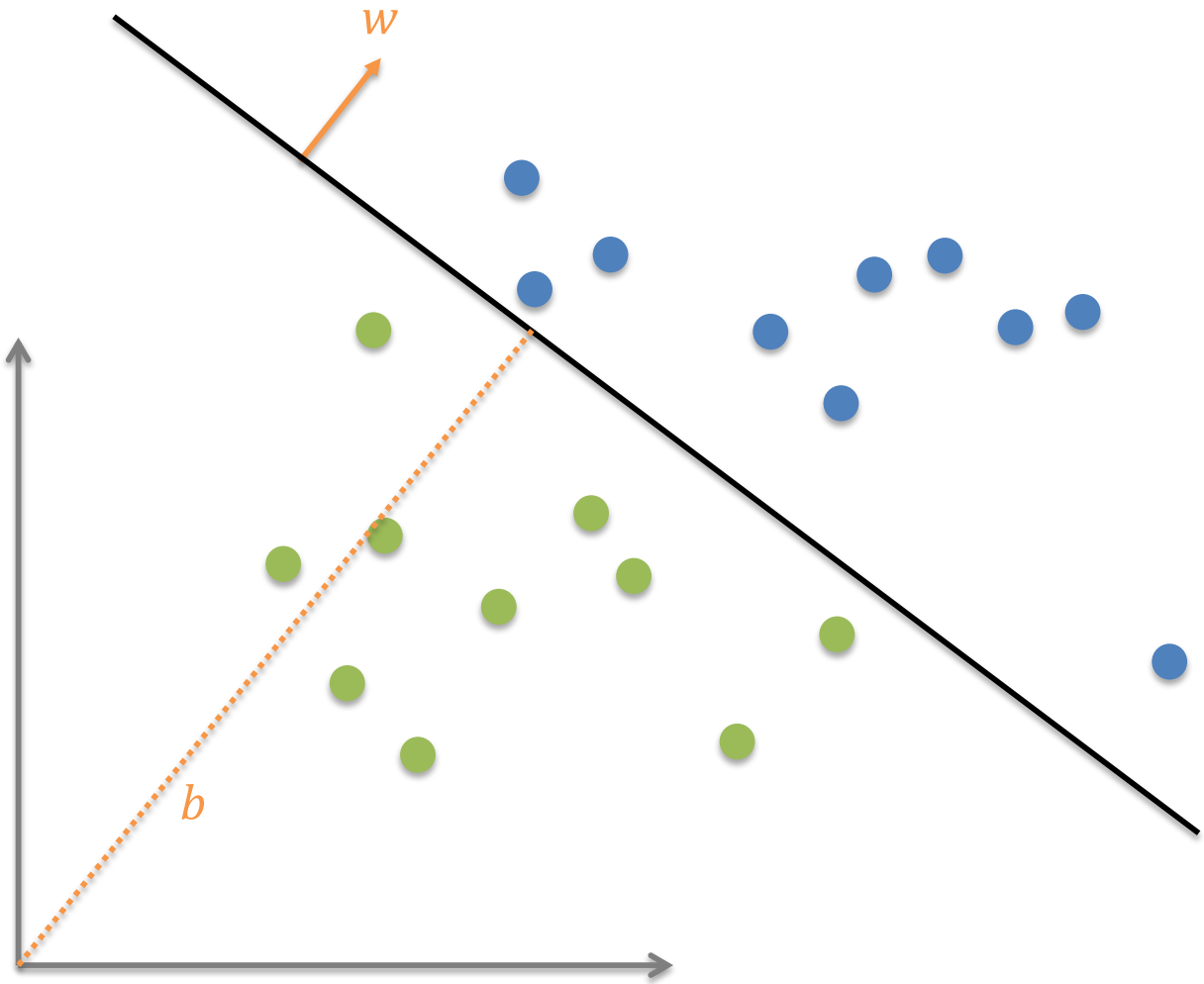
$$x, w \in \mathbb{R}^d \qquad b \in \mathbb{R}$$

Linear Classifier

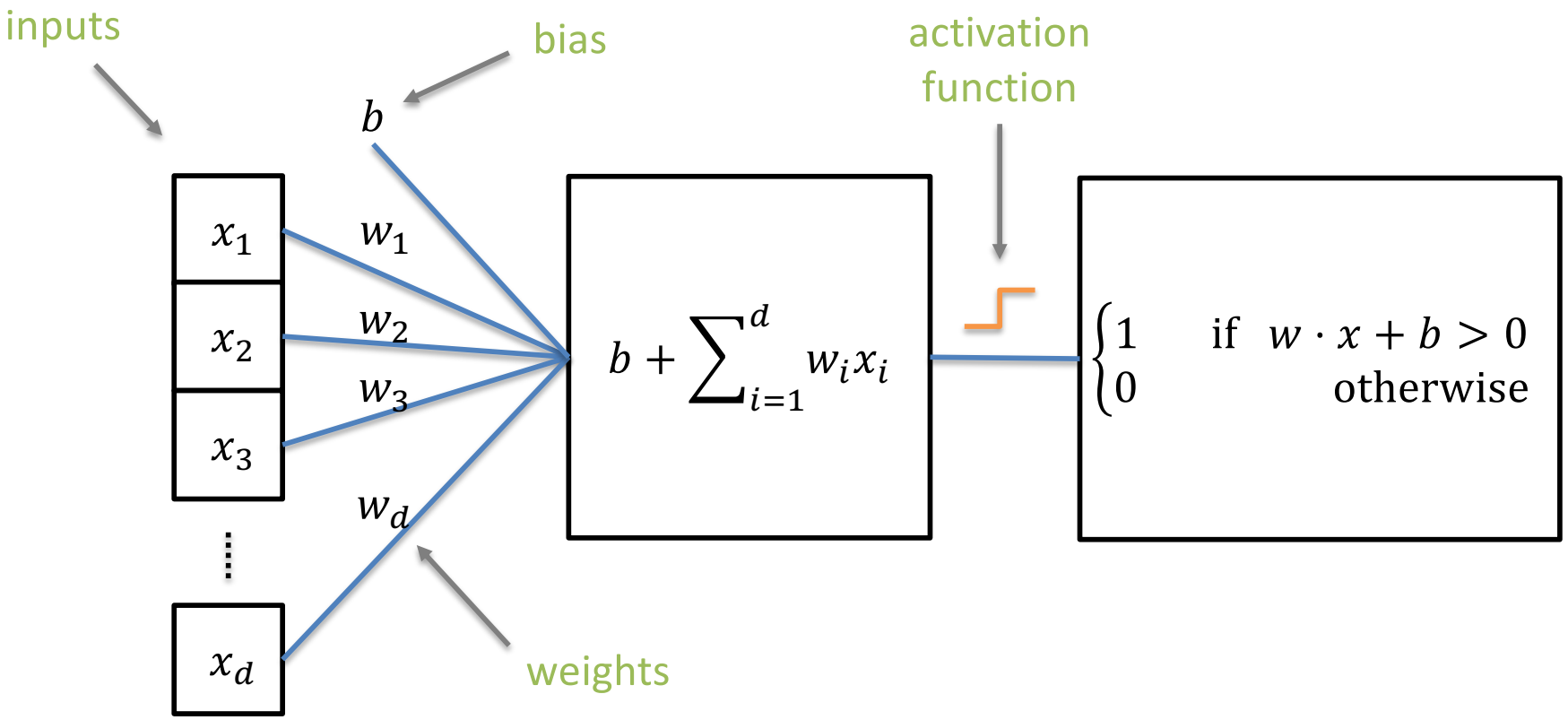
Lead to invention of support vector machines (SVM)

Only works well on linearly separable problems

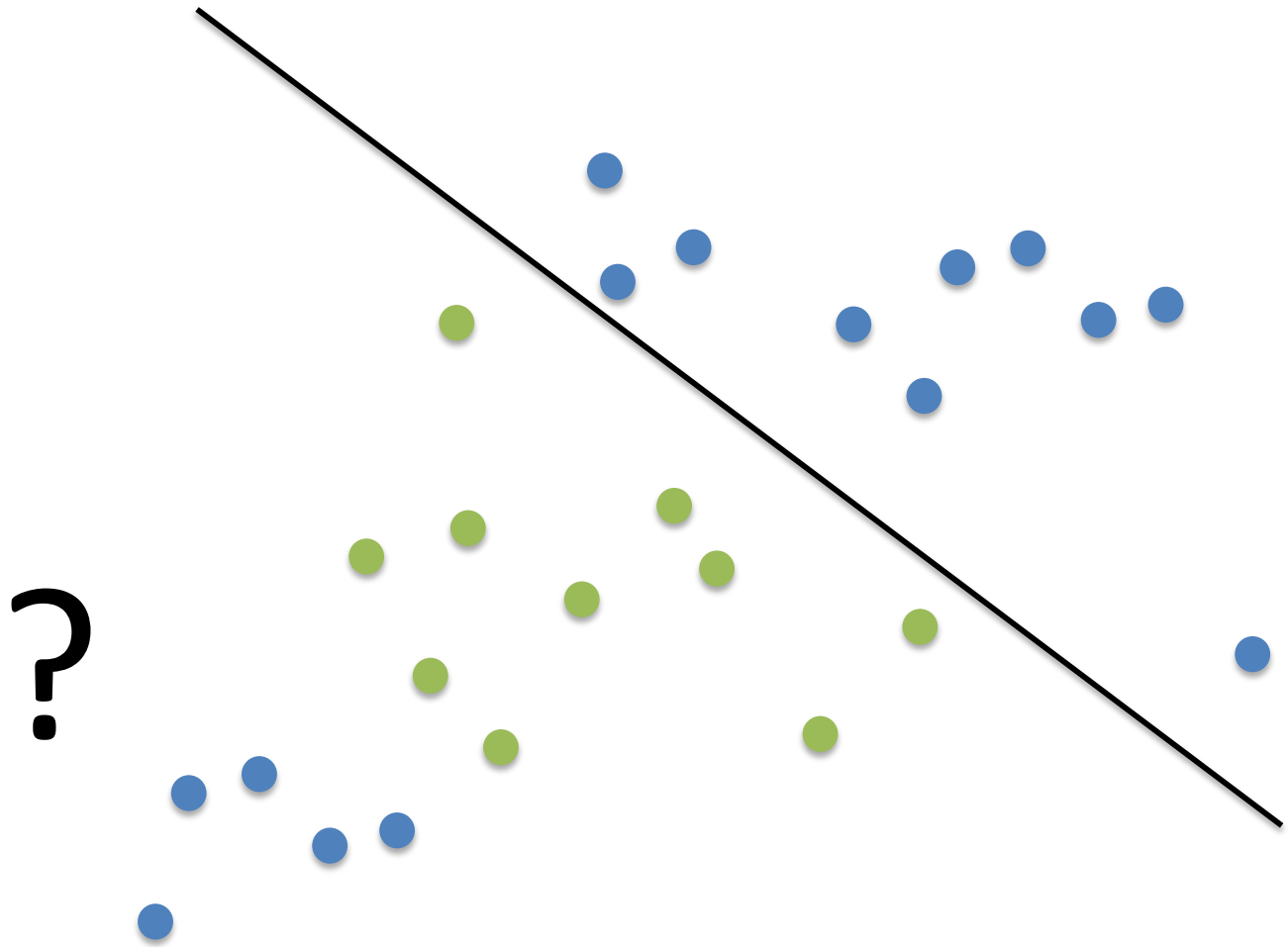
The Perceptron



The Perceptron



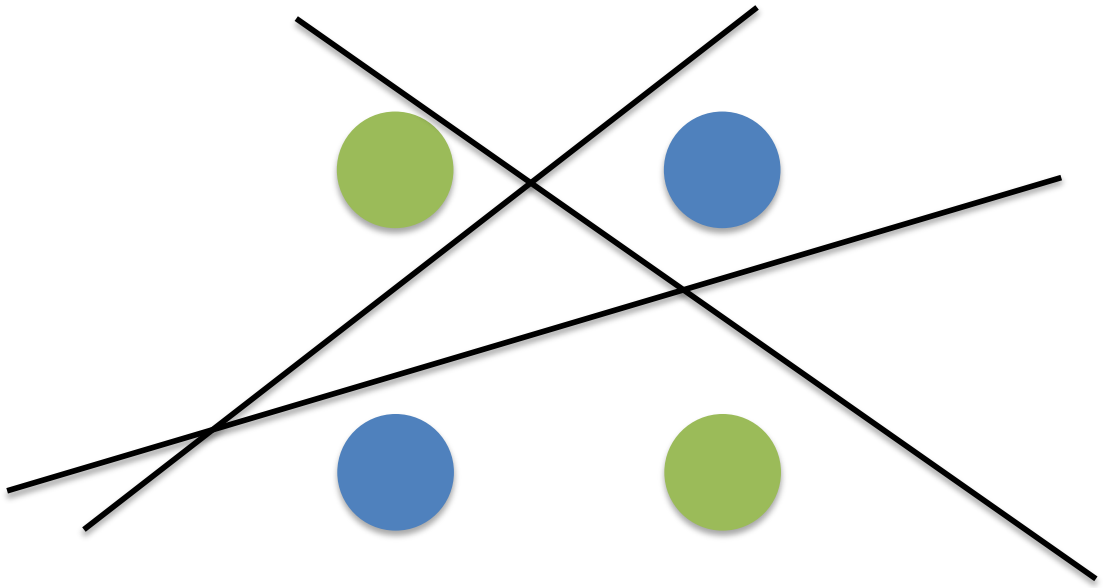
The Perceptron



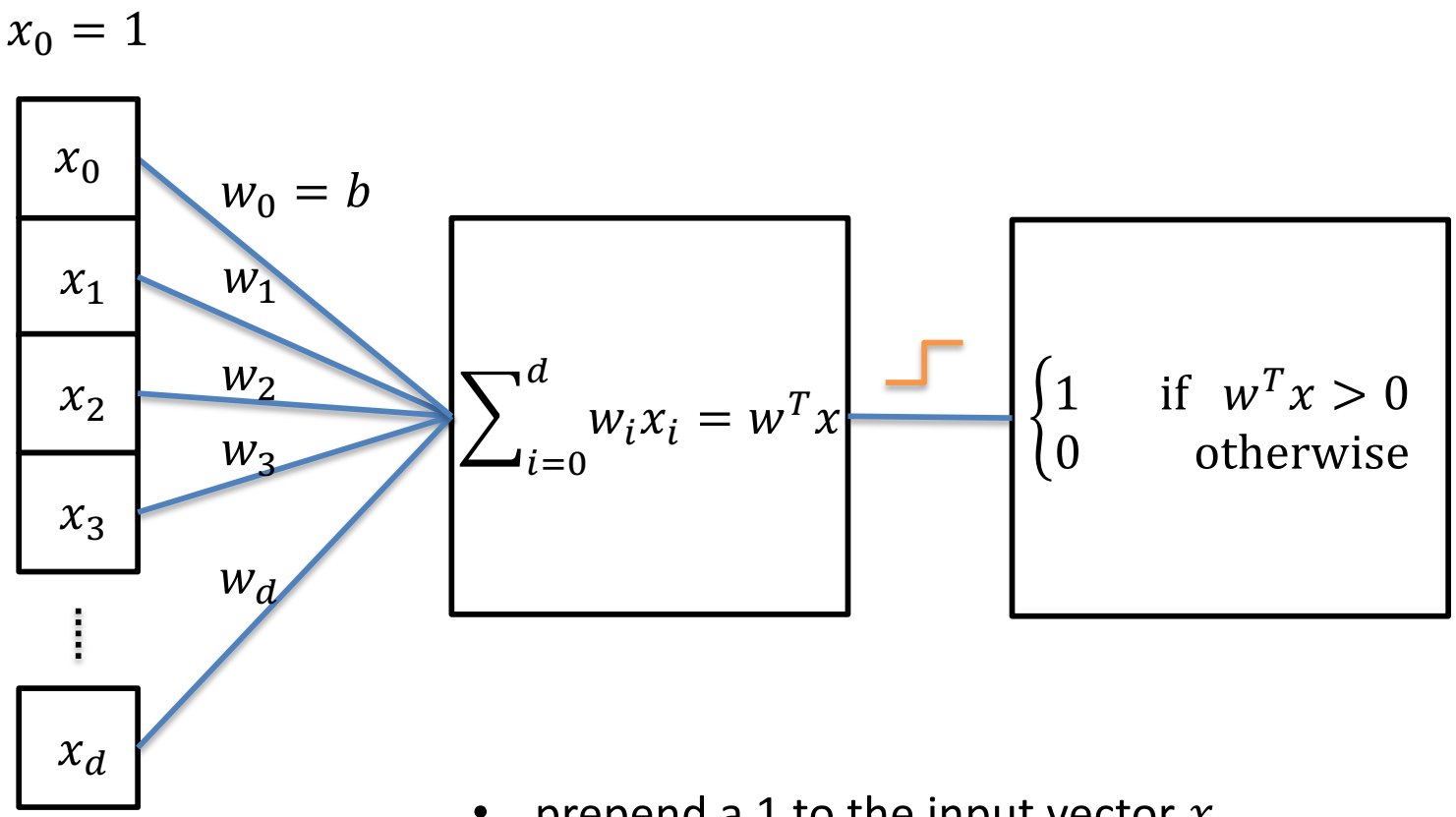
what to do when the problem is not linearly separable?

XOR

Minsky, Marvin, and Seymour Papert. "Perceptrons." (1969): Perceptrons cannot learn the XOR function

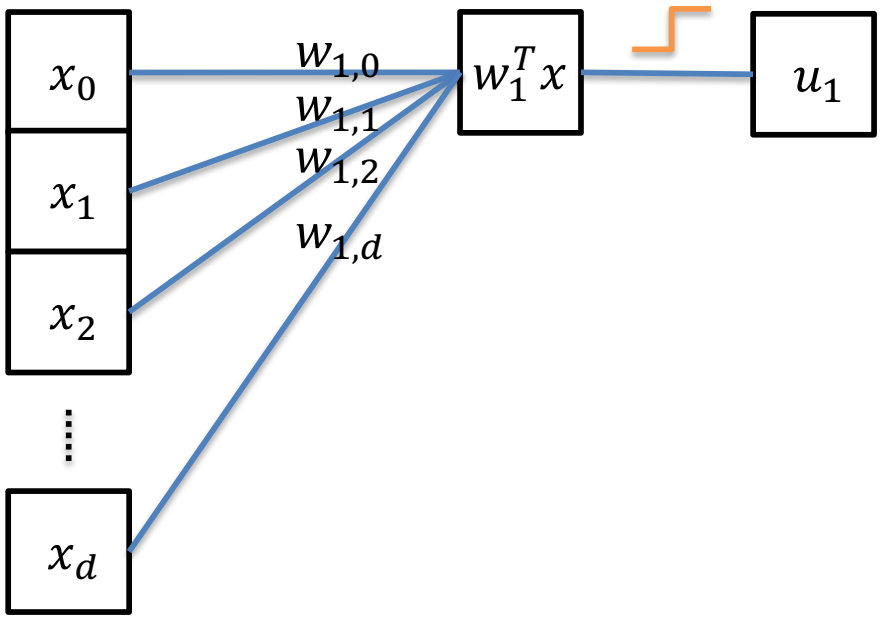


More Layers

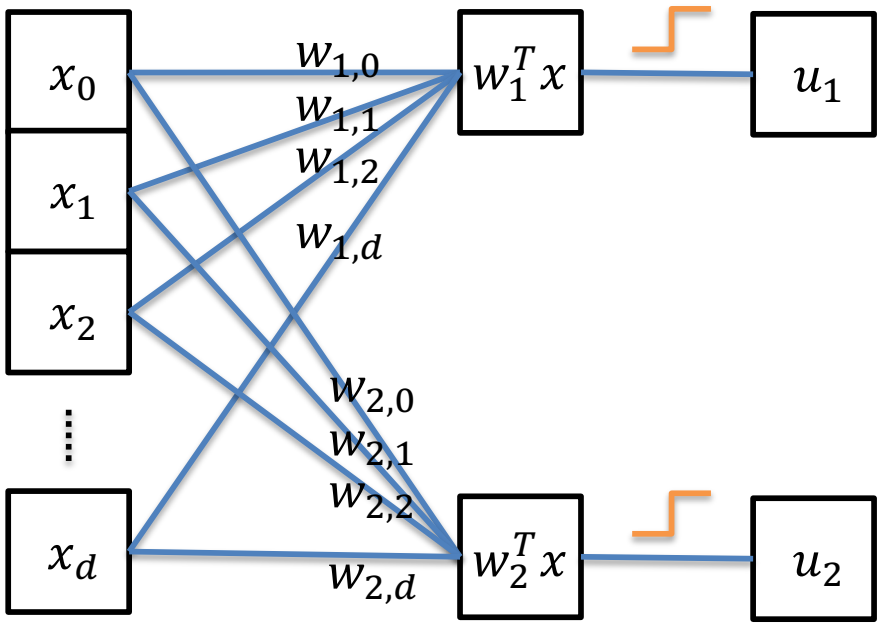


- prepend a 1 to the input vector x
- include the bias into the weights
- write everything as an inner product

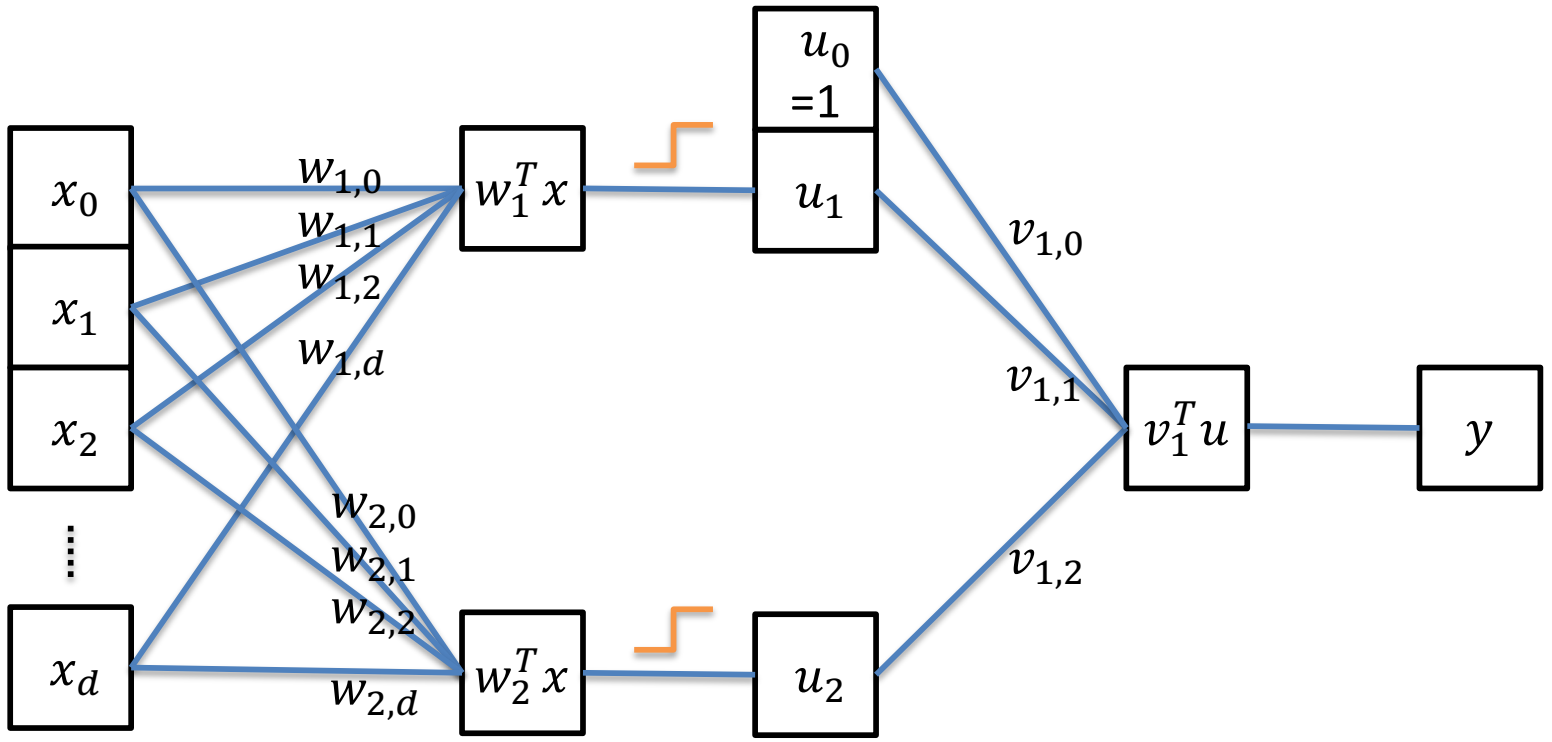
More Layers



More Layers



More Layers

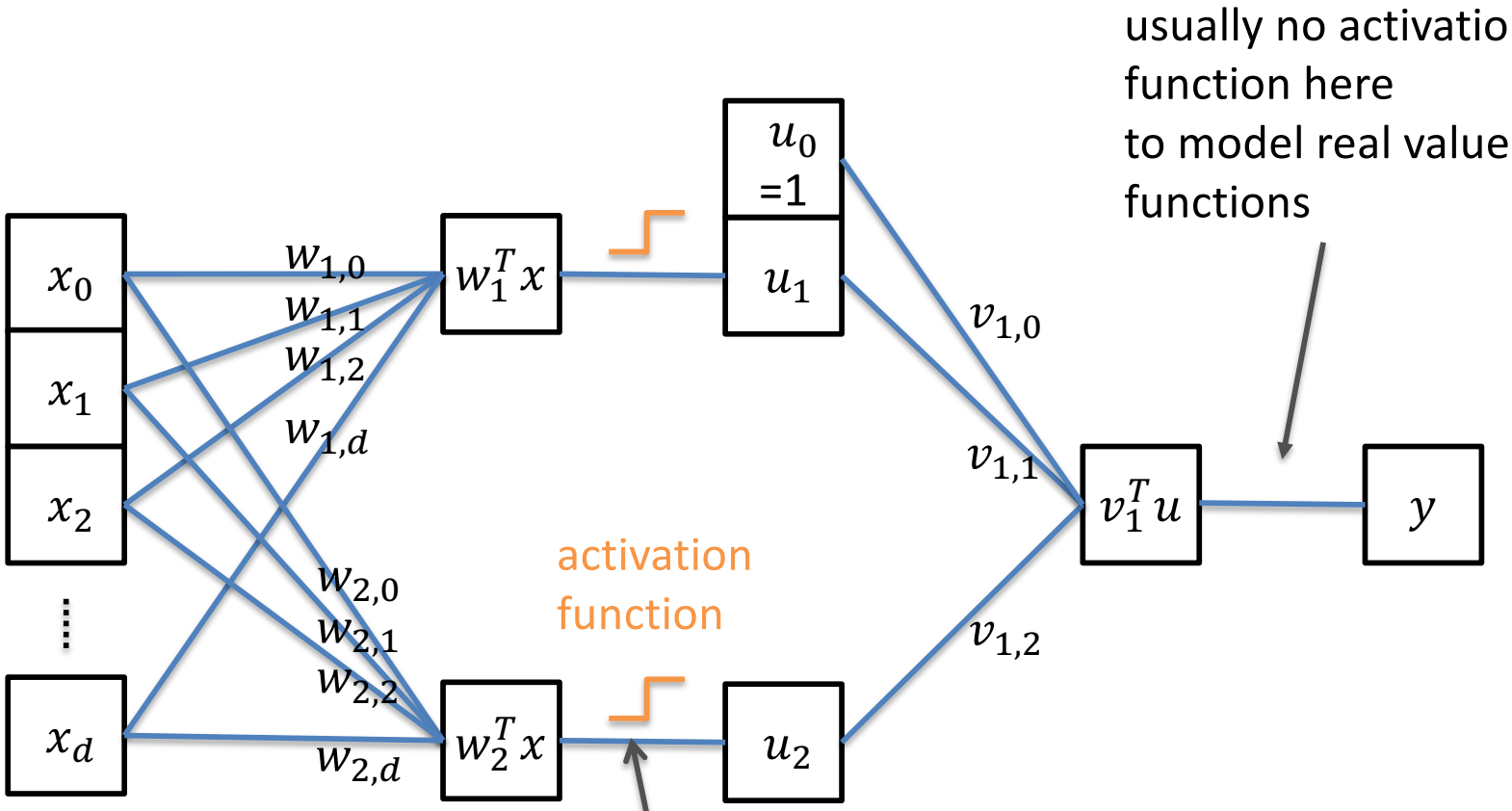


input layer

hidden layer

output layer

More Layers



$$u_j = \sigma\left(\sum_{i=0}^{d} w_{j,i}\, x_i\right) = \sigma(w_j\, x)$$

$$y = \sum_{j=0}^{2} u_{1,j}\, v_j$$

More Layers

$$u_j = \sigma(\sum_{i=0}^{d} w_{j,i}\, x_i) = \sigma(w_j\, x)$$

$$u = \sigma(wx)$$

$$y = uv = v\sigma(wx)$$

Matrix notation greatly simplifies writing down the computations in the layers

Universal Approximation Theorem

If f^* satisfies mild regularity conditions, then for every quality level $\epsilon > 0$, it is possible to choose the parameters of

$$f(x) = \sum_i^N v_i \sigma(w_i^T x + b_i)$$

such that

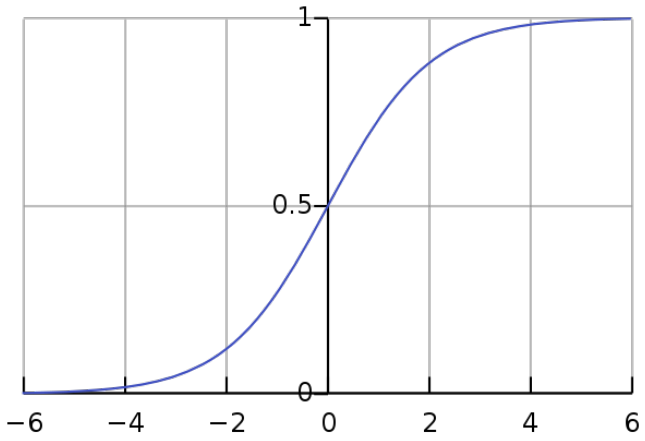
$$|f(x) - f^*(x)| < \epsilon \quad \forall x,$$

given that the number of parameters N is large enough.

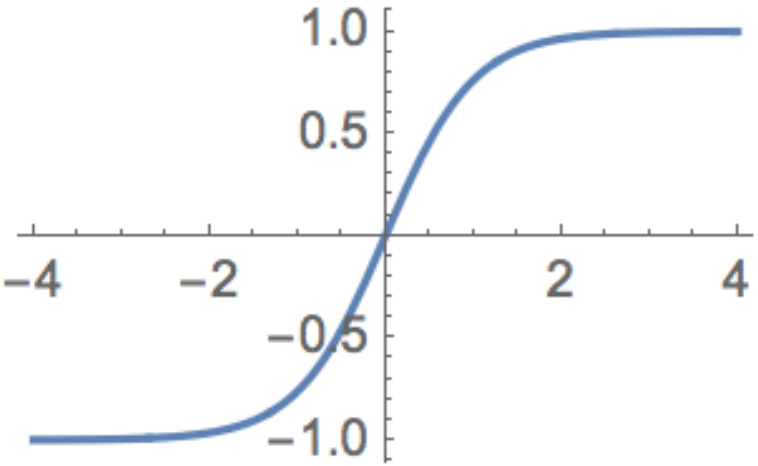
Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

Activation Functions

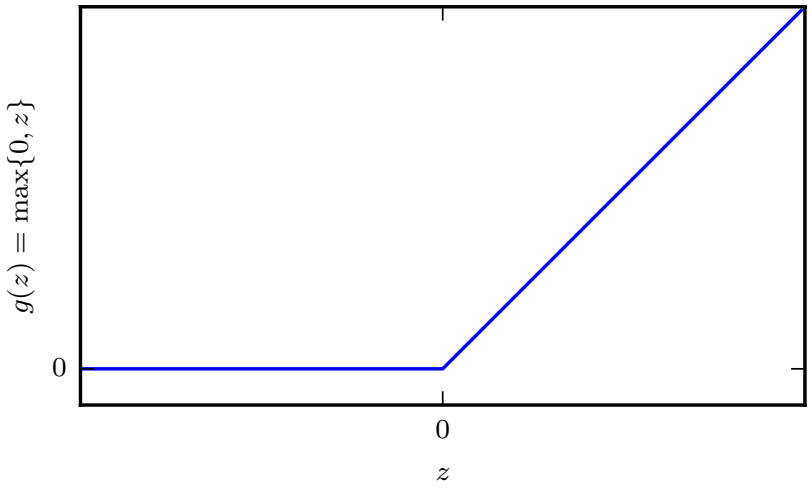
- $g(z) = \sigma(z)$



- $g(z) = \tanh(z)$ (preferred as tanh(0) = 0 and closer to identity near 0)



- $g(z) = ReLU(z)$

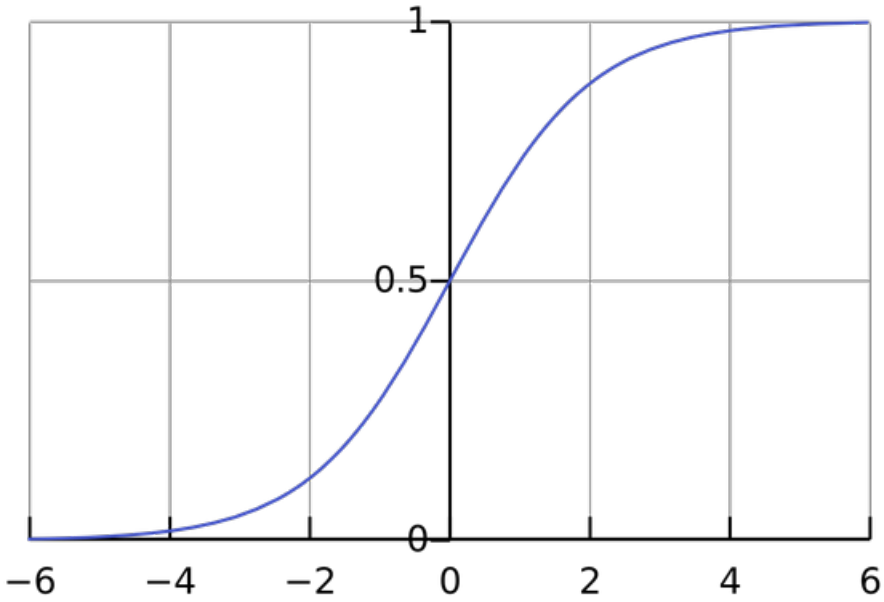


Other Options

- Softmax
- Radial Basis $g(z) = \exp\left(-\frac{1}{\sigma^2}||w - z||^2\right)$
  - "template matching"
- Softplus $\varsigma(z) = \log(1 + e^z)$
- Bounded tanh (between -1 and 1)
- Leaky ReLu
- ….

Softmax Output

- "Squashes" K-dimensional vector **z** of real values to K-dimensional vector of values in range [0.0,1.0]
  - Great for probabilities
- Represents **categorical probability distributions**: probability distribution over K possible outcomes



$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

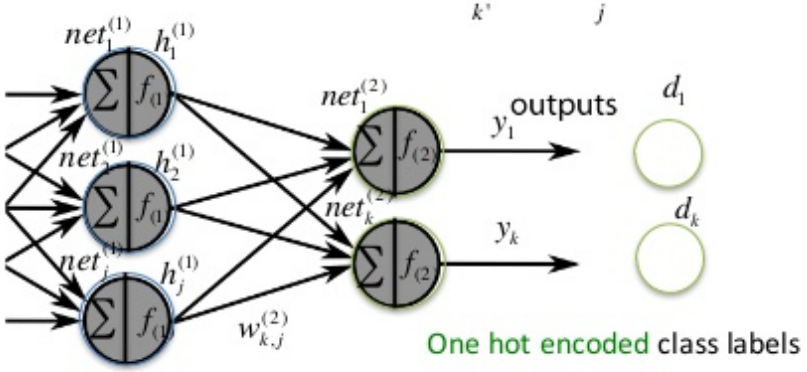
Multi-Class Classification via Softmax

Error/Loss function

- Softmax (cross-entropy loss) for multiple classes:

(Class labels follow multinomial distribution)

$$E(w) = -\sum_k (d_k \log y_k + (1 - d_k)\log(1 - y_k)) \quad \text{where} \quad y_k = \frac{\exp(\sum_j w_{k,j}^{(2)} h_j^{(1)})}{\sum_{k'} \exp(\sum_j w_{k',j}^{(2)} h_j^{(1)})}$$



"One-hot" encoding:

Suppose we have a 5-class pro blem.

A ground-truth sample labeled as class "3" would be encoded as (assumes 0-based indices):

y = [0,0,0,1,0]

How To Train Your Network

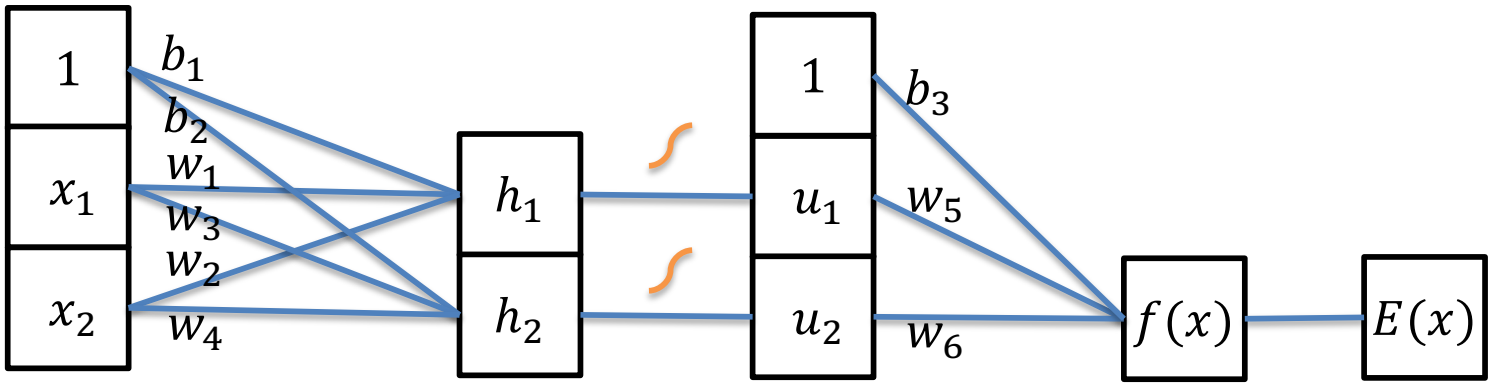
Set of N samples $\left(x^{(i)}, y^{(i)}\right)$

Define loss $\mathrm{E}(x) \in \mathbb{R}$ to measure error for a sample

Training: finding weights that minimize $\sum_i E\left(x^{(i)}\right)$ for the samples

Often uses simple gradient descent methods

Backpropagation computes the gradients of all parameters



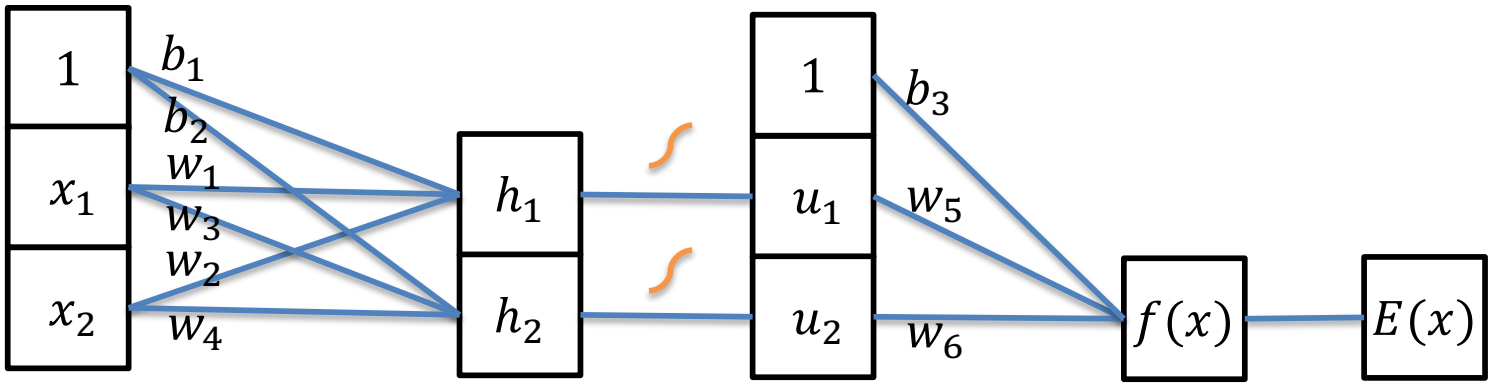
Error and Activation

Here, as an example, we choose:

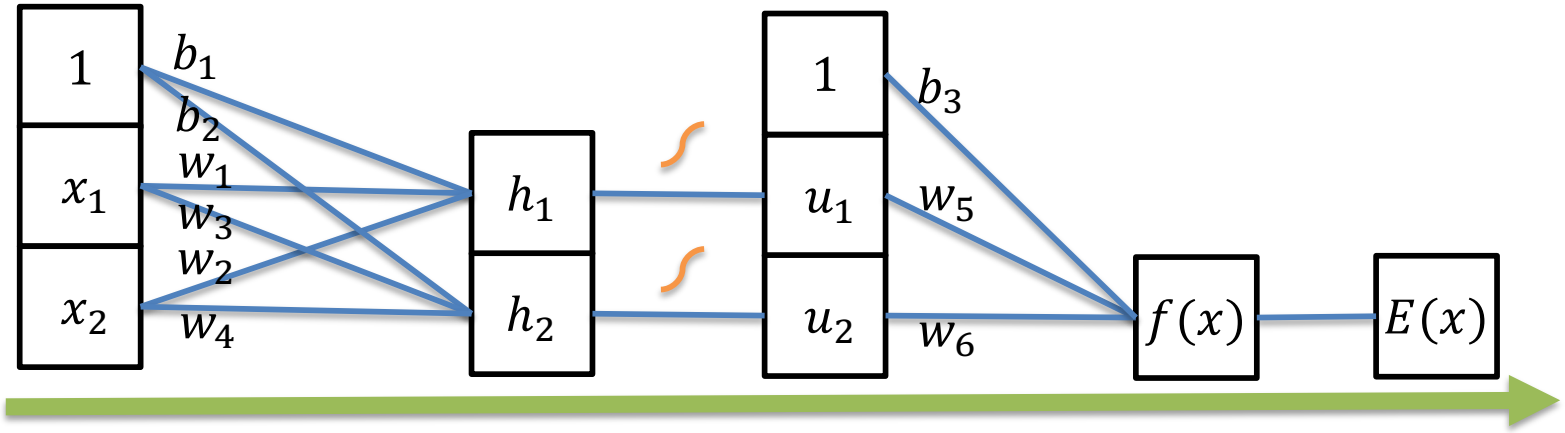
$$E\left(x^{(i)}\right) = \frac{1}{2}\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$\sigma(z) = \tanh(z)$$

BP is an efficient way to compute gradients for all parameters



Forward Pass



$$h_1 = x_1 w_1 + x_2 w_2 + b_1$$
$$h_2 = x_1 w_3 + x_2 w_4 + b_2$$

$$u_1 = \tanh h_1$$
$$u_2 = \tanh h_2$$

$$f(x) = u_1 w_5 + u_2 w_6 + b_3$$

$$E(x) = \frac{1}{2}(f(x) - y)^2$$

Backward Pass

Idea: use chain rule

$$f\left(g\left(h(x)\right)\right)$$

$$\frac{\partial}{\partial x} f\left(g\left(h(x)\right)\right) = \frac{\partial f(u)}{\partial u}\left(g\left(h(x)\right)\right)\frac{\partial g(v)}{\partial v}\left(h(x)\right)\frac{\partial h(x)}{\partial x}(x)$$

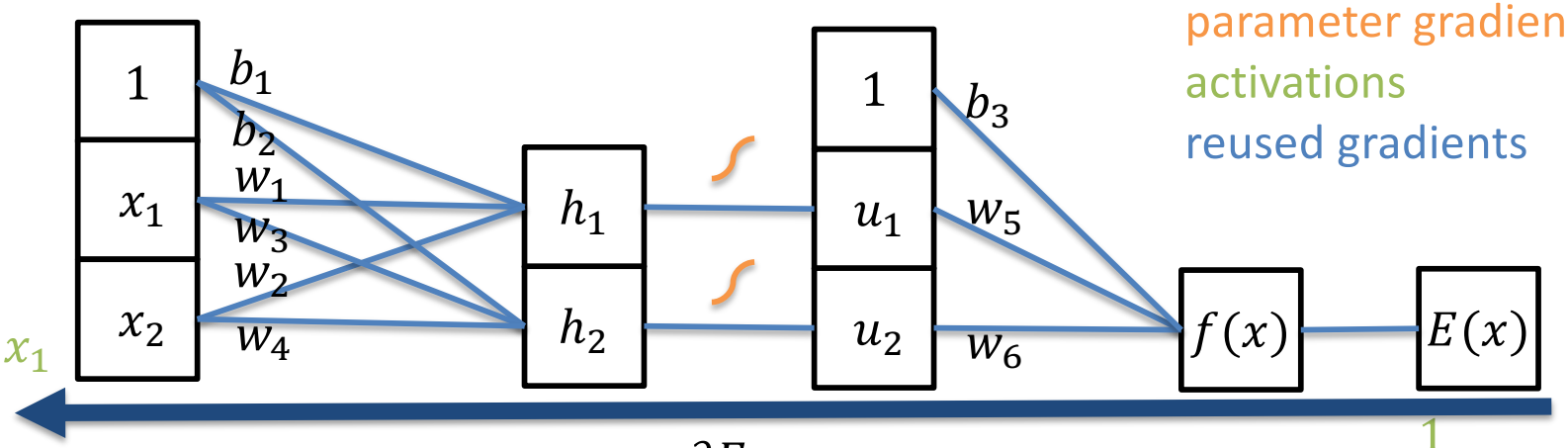
simpler (Leibnitz's) notation:

$$t = f(u), \qquad u = g(v), \qquad v = h(x)$$

$$\frac{\partial t}{\partial x} = \frac{\partial t}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$$

Backward Pass

parameter gradients
activations
reused gradients



$$\frac{\partial E}{\partial b_1} = \delta_4$$

$$\frac{\partial E}{\partial w_1} = \delta_4 x_1$$

$$\frac{\partial E}{\partial w_2} = \delta_4 x_2$$

$$h_1 = x_1 w_1 + x_2 w_2 + b_1$$

$$h_2 = x_1 w_3 + x_2 w_4 + b_2$$

$$\frac{\partial E}{\partial u_1} = \delta_1 w_5 = \delta_2$$

$$\frac{\partial E}{\partial u_2} = \delta_1 w_6 = \delta_3$$

$$E(x) = \frac{1}{2}(f(x) - y)^2$$

$$\frac{\partial E}{\partial f(x)} = f(x) - y = \delta_1$$

$$\frac{\partial E}{\partial b_2} = \delta_5$$

$$\frac{\partial E}{\partial w_3} = \delta_5 x_1$$

$$\frac{\partial E}{\partial w_4} = \delta_5 x_2$$

$$u_1 = \tanh h_1$$

$$u_2 = \tanh h_2$$

$$f(x) = u_1 w_5 + u_2 w_6 + b_3$$

$$\frac{\partial E}{\partial h_1} = \frac{\partial E}{\partial u_1}\frac{\partial u_1}{\partial h_1} = \delta_2(1 - \tanh^2 h_1) = \delta_4$$

$$\frac{\partial E}{\partial h_2} = \frac{\partial E}{\partial u_2}\frac{\partial u_2}{\partial h_2} = \delta_3(1 - \tanh^2 h_2) = \delta_5$$

$$\frac{\partial E}{\partial b_3} = \frac{\partial E}{\partial f}\frac{\partial f}{\partial b_3} = \delta_1 \cdot 1$$

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial f}\frac{\partial f}{\partial w_5} = \delta_1 u_1$$

$$\frac{\partial E}{\partial w_6} = \frac{\partial E}{\partial f}\frac{\partial f}{\partial w_6} = \delta_1 u_2$$

Gradient Descent

Initialize all weights (randomly, close to solution, …)

How to use the derivatives?

- Gradient descent to minimize error function

$$w^{(t+1)} = w^{(t)} - \lambda \frac{\partial E}{\partial w}$$

Update the weights in every iteration of training with a small gradient step

Learning rate λ adjusts the stepsize

Stochastic Gradient Descent

Error was defined over the whole training set: $\sum_i E(x^{(i)})$

Need to compute and sum the derivatives of all samples before gradient step

Slow but accurate updates

Stochastic Gradient Descent (SGD): approximate derivative from small, random subset ([mini]batch) of training set

Noisy but faster

Usually: make sure to see every sample the same amount of times (epochs)