CS 232: Artificial Intelligence

Spring 2024

Prof. Carolyn Anderson
Wellesley College

Reminders

- Buy-1-get-3 late day sale on HW 5
- Progress updates sent yesterday did not include late days for HW 5 (unless you've already submitted it)
- Mid-semester feedback

* My help hours from 3-3:45

Recap

Spot the differences

Neural Network Unit

$$z = b + \sum_i w_i x_i$$

$$y = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

σ could be sigmoid or softmax

but could also be

ReLU or tanh

Logistic Regression

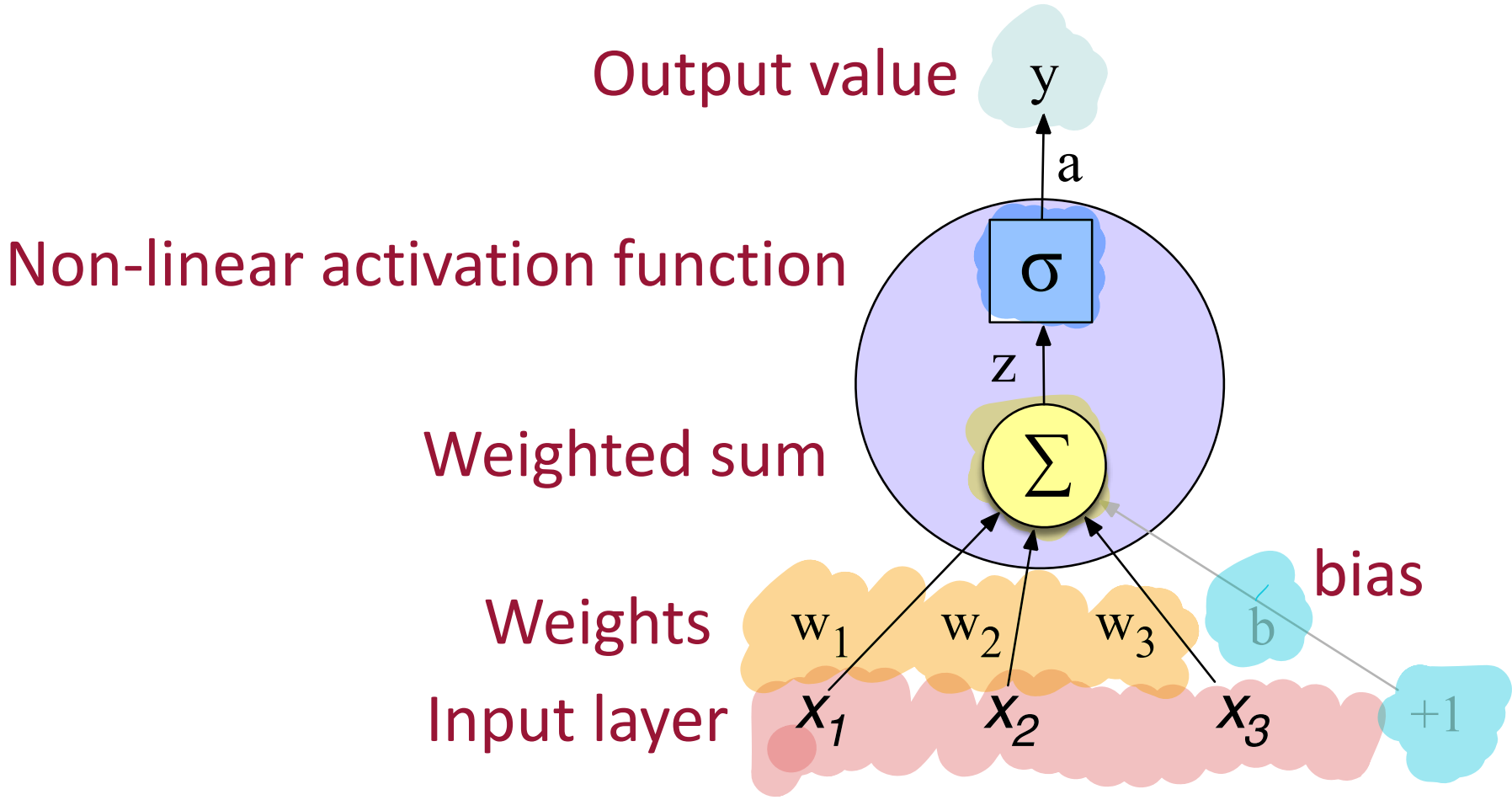
$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

$$P(y = 1) = \sigma(w \cdot x + b)$$

sigmoid for binary classification

softmax for multi-class

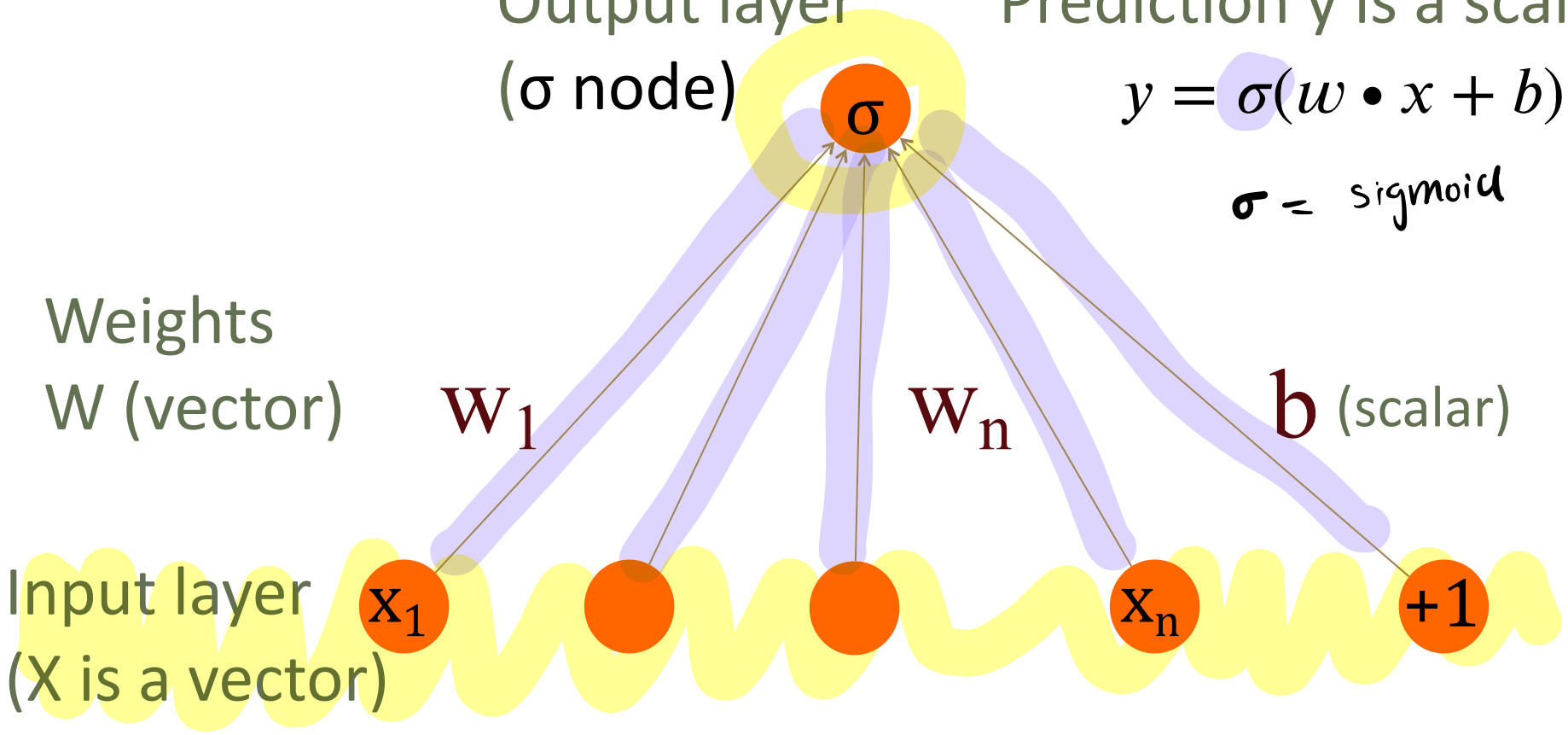
Neural Network Unit



Feedforward Networks

Binary Logistic Regression as a 1-layer Network

(we don't count the input layer when counting layers!)



Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network

$$y = \text{softmax}(Wx + b)$$

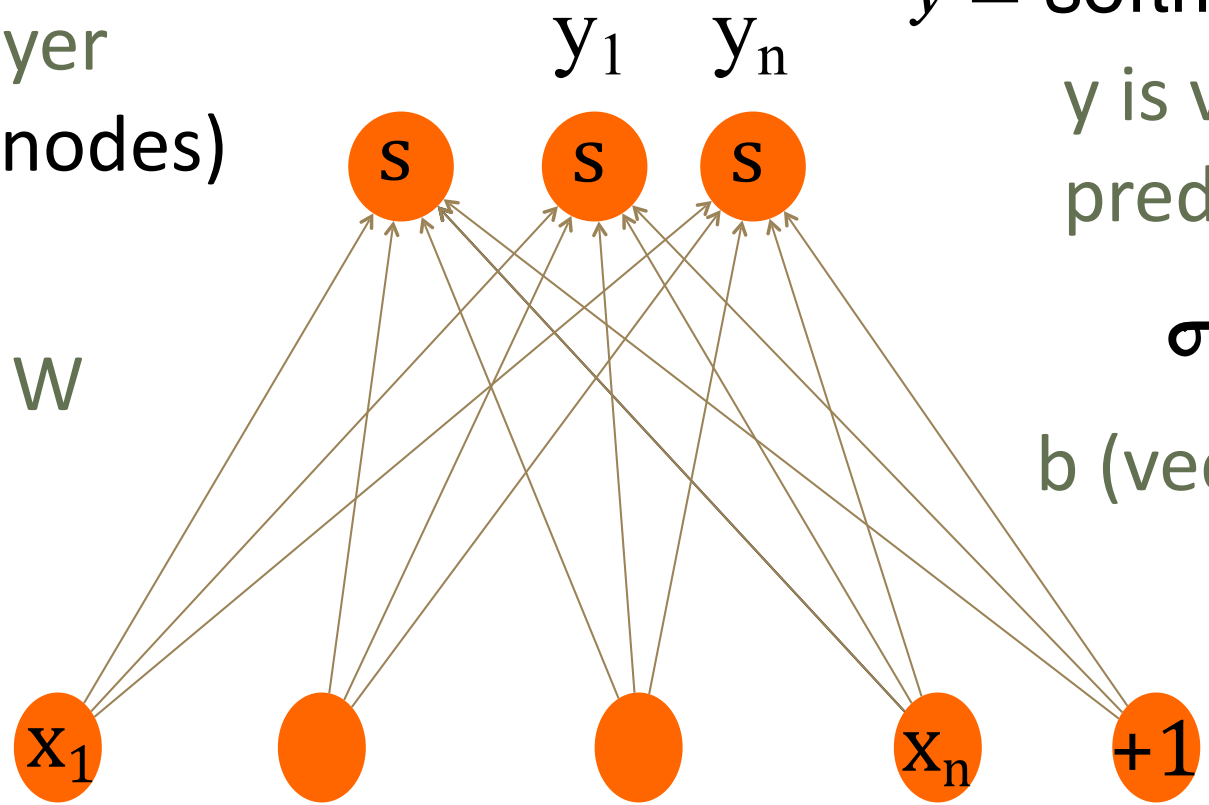
Output layer
(softmax nodes)

y is vector of predictions

σ = softmax

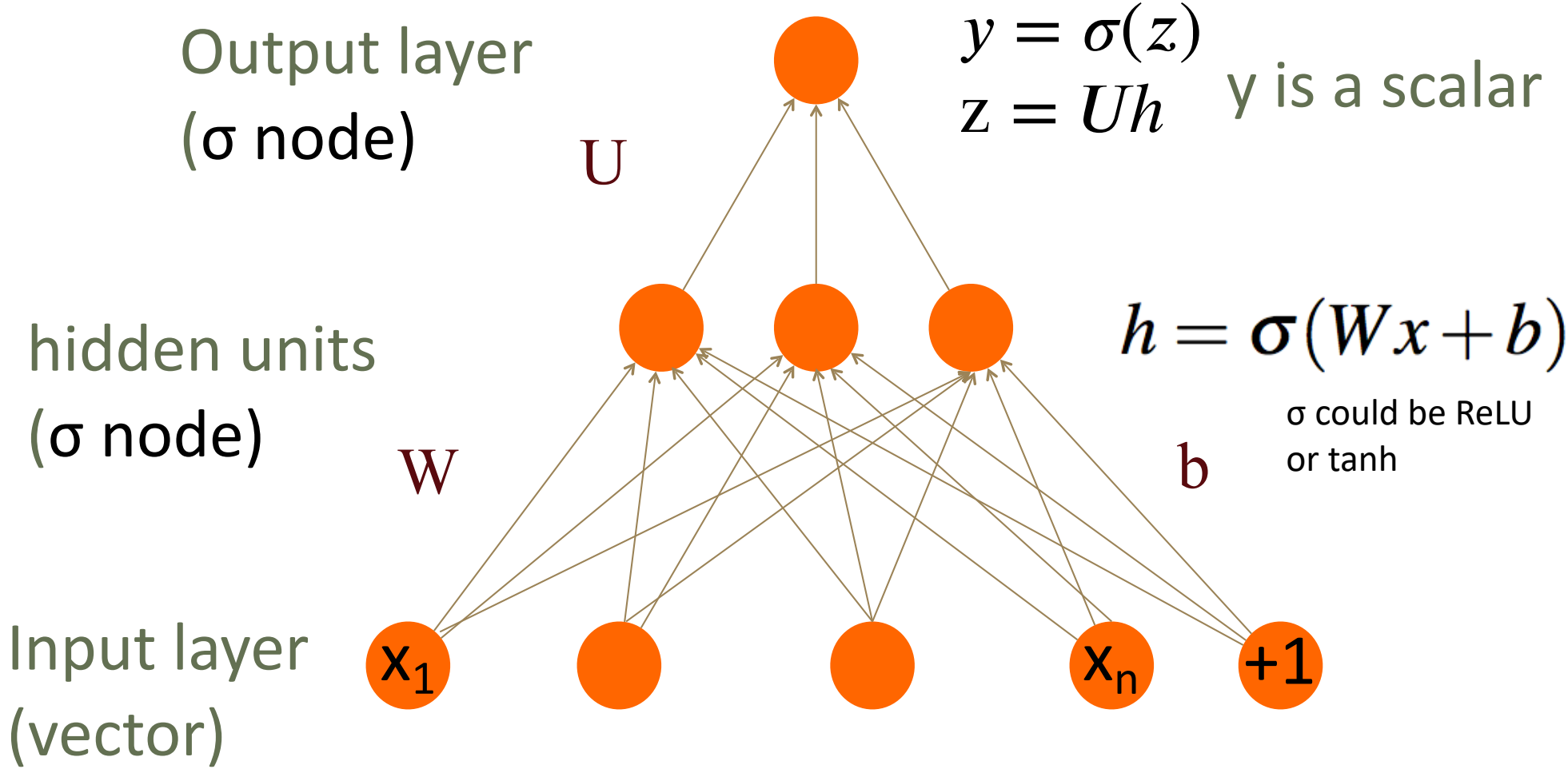
Weights W
(matrix)

b (vector)

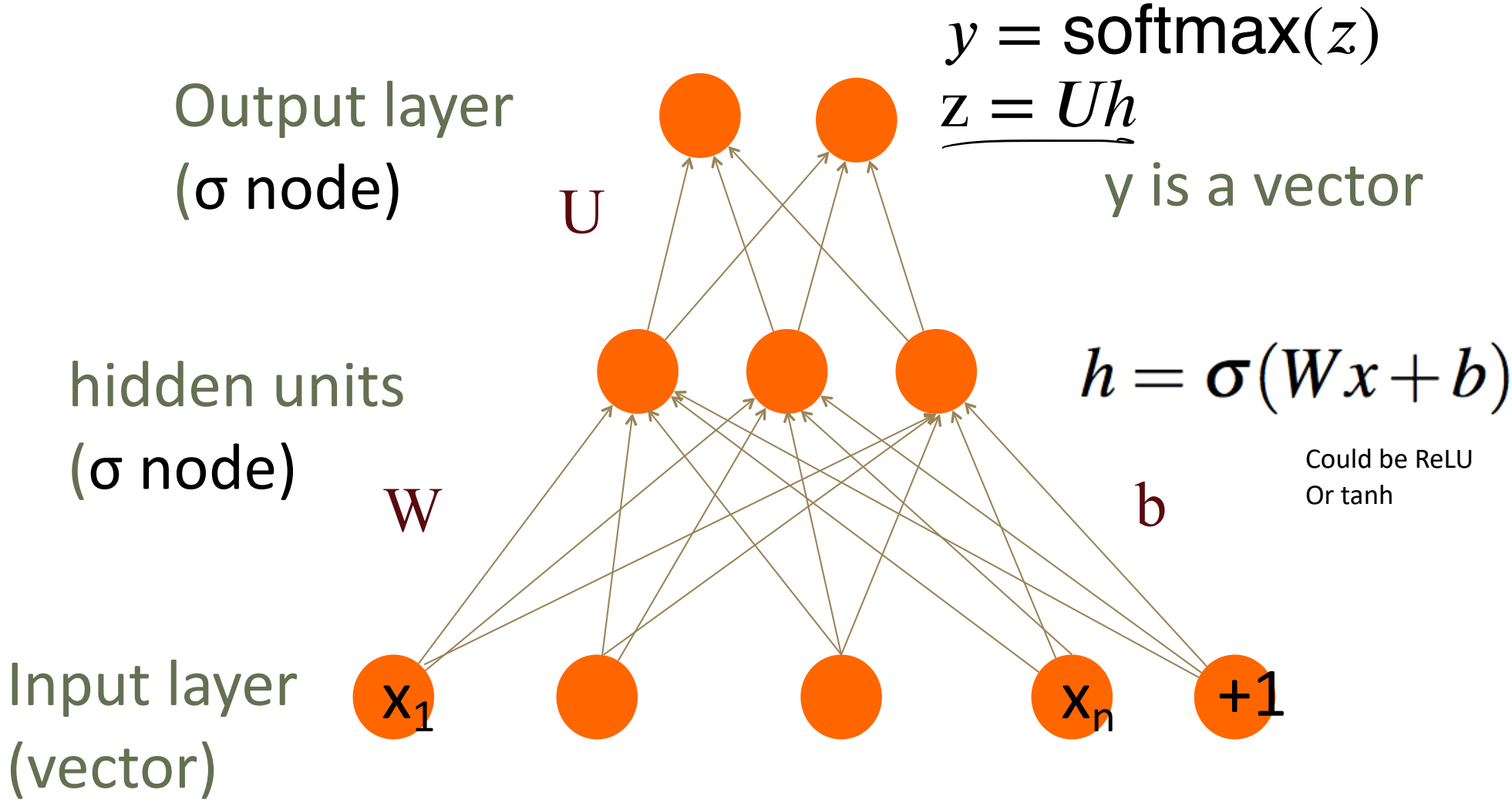


Input layer
(X is a vector)

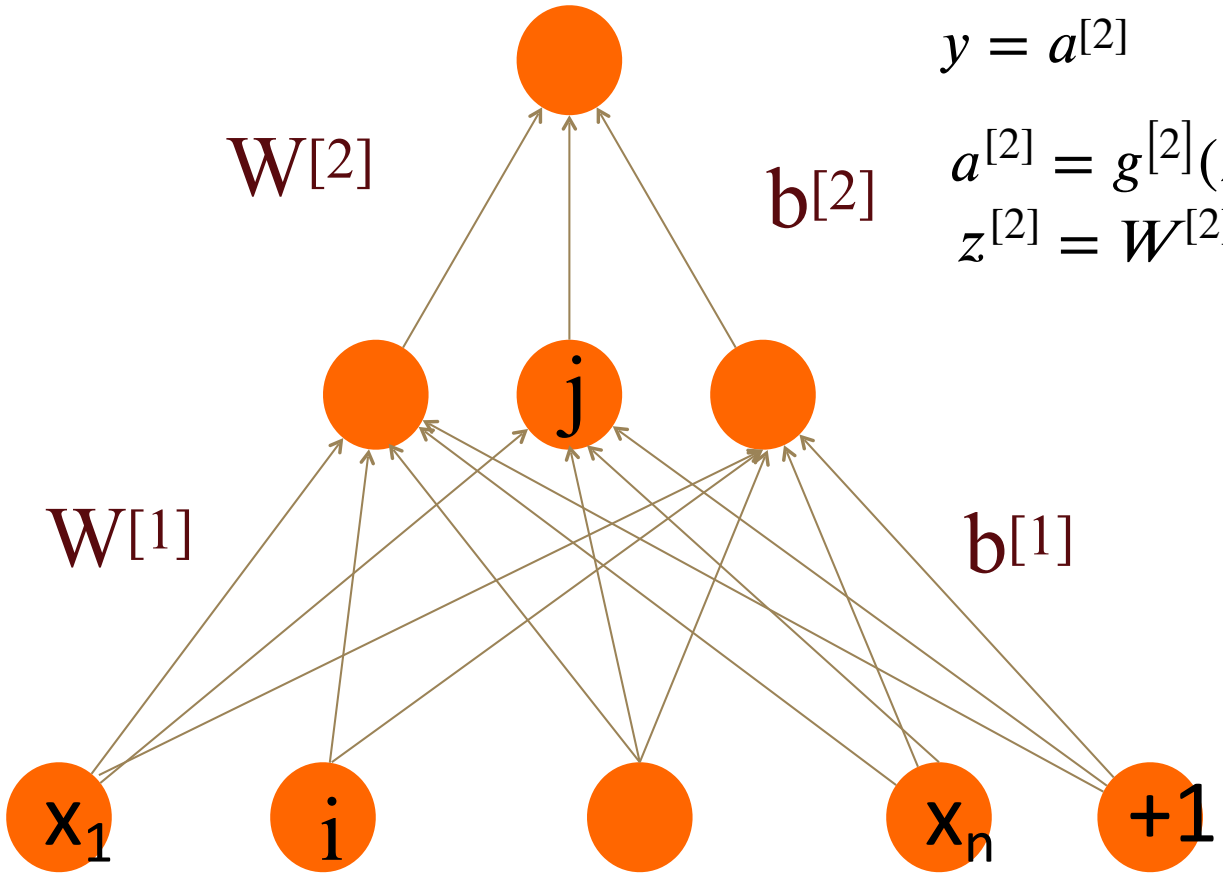
Two-Layer Network with scalar output



Two-Layer Network with softmax output



Multi-layer Notation



$y = a^{[2]}$ sigmoid or softmax

$a^{[2]} = g^{[2]}(z^{[2]})$

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[1]} = g^{[1]}(z^{[1]})$ ReLU

$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$

$a^{[0]}$

Replacing the bias unit

Let's switch to a notation without the bias unit (a notational change):

1. Add a dummy node $a_0=1$ to each layer
2. Its weight w_0 will be the bias
3. So input layer $a^{[0]}_0=1$,
   - And $a^{[1]}_0=1$, $a^{[2]}_0=1$,…

Replacing the bias unit

Instead of:

$x = x_1, x_2, \ldots, x_{n0}$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{h}_j = \sigma\left(\sum_{i=1}^{n_0} \mathbf{W}_{ji}\,\mathbf{x}_i + \mathbf{b}_j\right)$$

We'll do this:

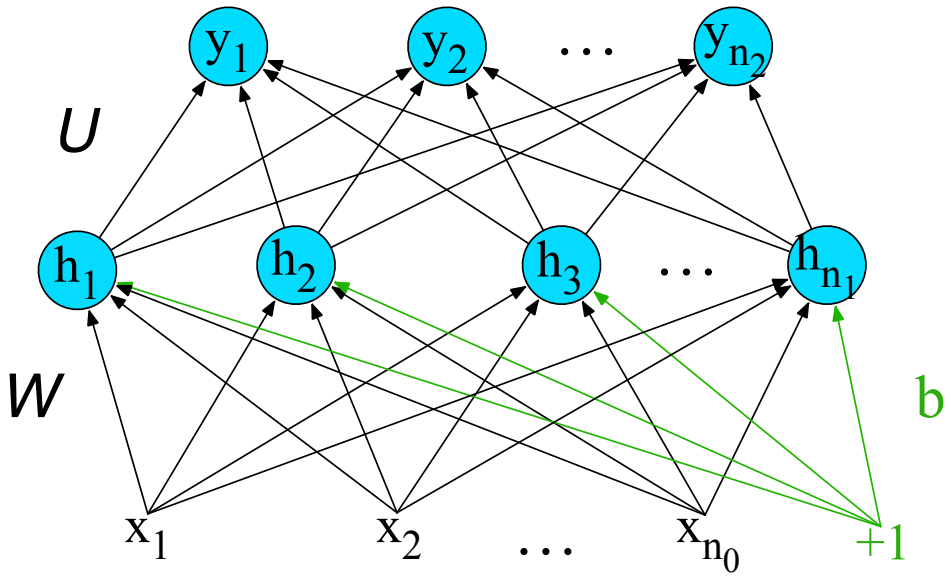
$x = x_0, x_1, x_2, \ldots, x_{n0}$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x})$$

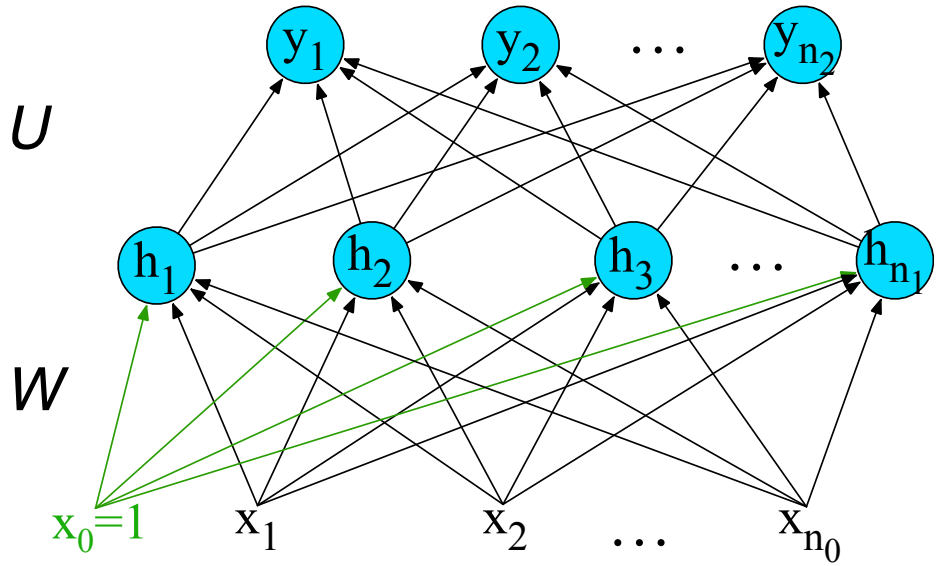
$$\sigma\left(\sum_{i=0}^{n_0} \mathbf{W}_{ji}\,\mathbf{x}_i\right)$$

Replacing the bias unit

Instead of:



We'll do this:



Using feedforward networks

Use cases for feedforward networks

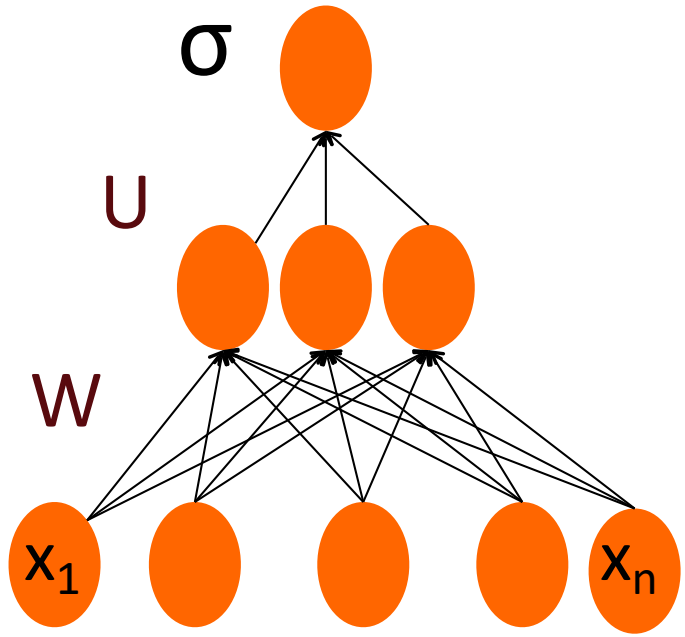
Let's revisit sentiment analysis with feedforward networks (state-of-the-art systems use more powerful architectures).

Classification: Sentiment Analysis

We could do exactly what we did with logistic regression

Input layer are binary features as before

Output layer is 0 or 1

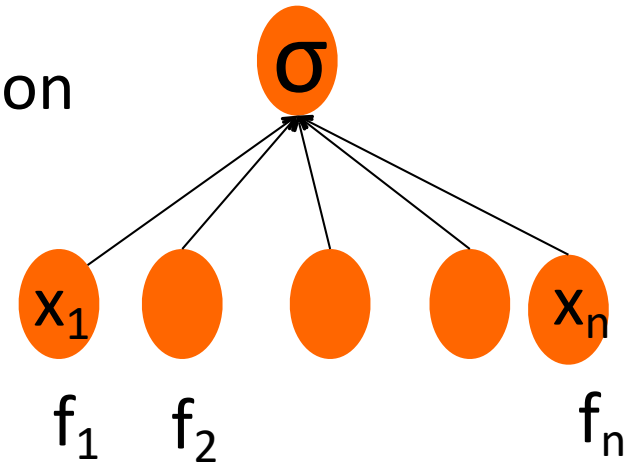


Sentiment Features

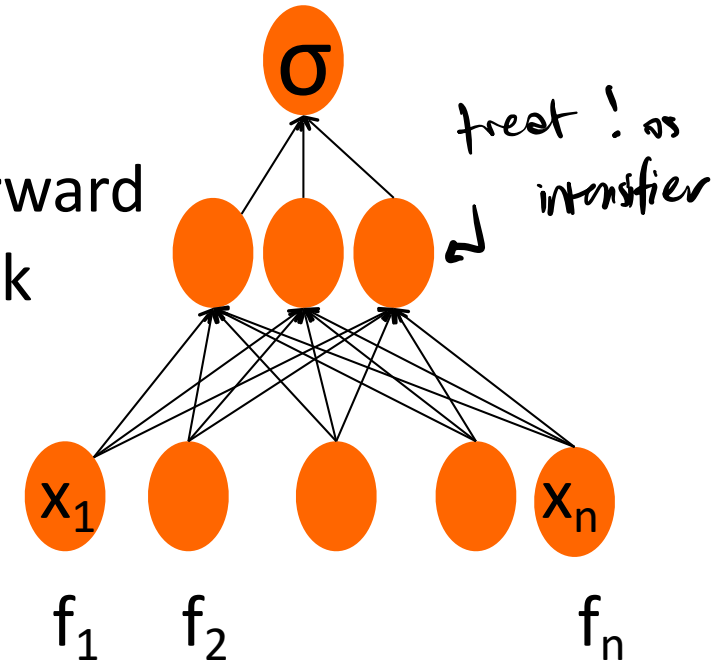
Var	Definition
x_1	count(positive lexicon words $\in$ doc)
x_2	count(negative lexicon words $\in$ doc)
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Feedforward nets for simple classification

Logistic Regression



2-layer feedforward network



Just add a hidden layer to logistic regression!

This allows the network to use non-linear interactions between features (which *hopefully* improves performance).

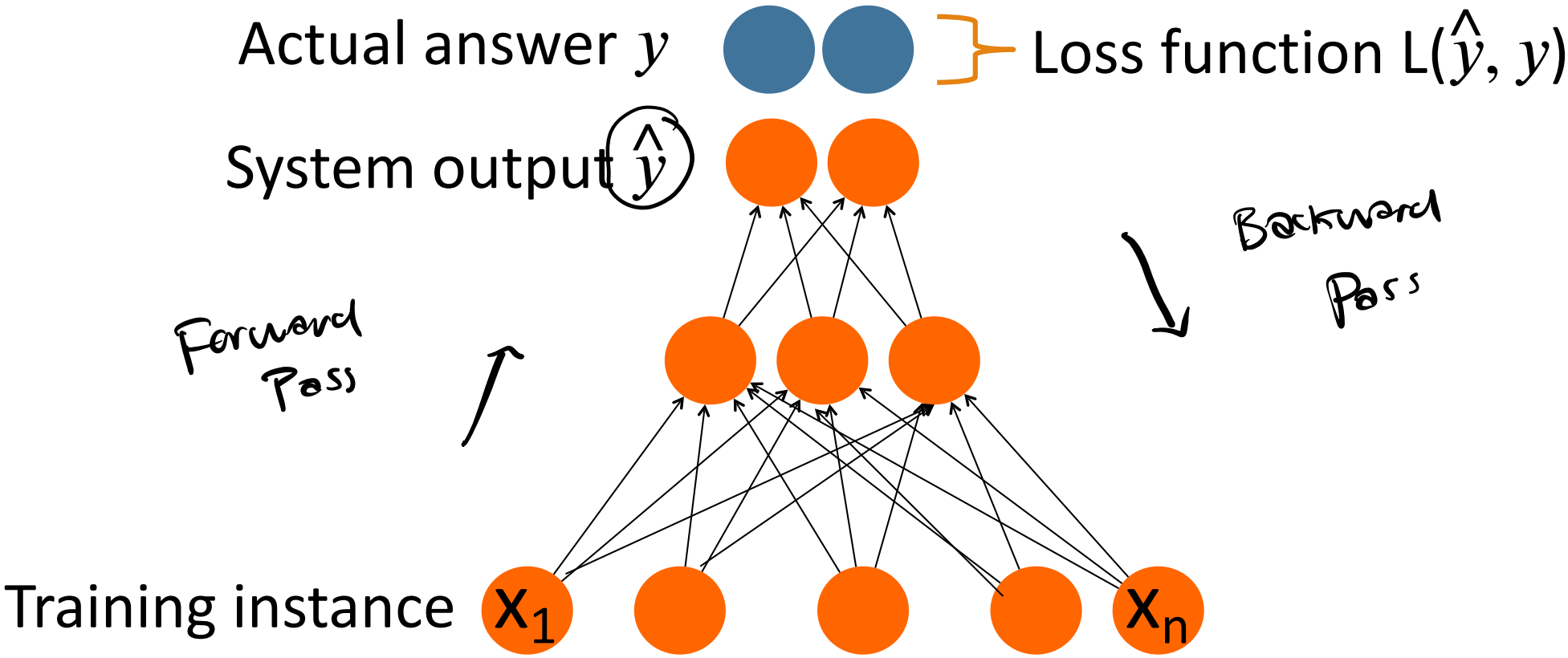
Even better: representation learning

The real power of deep learning comes from the ability to learn features from the data, instead of using hand-built human-engineered features for classification.

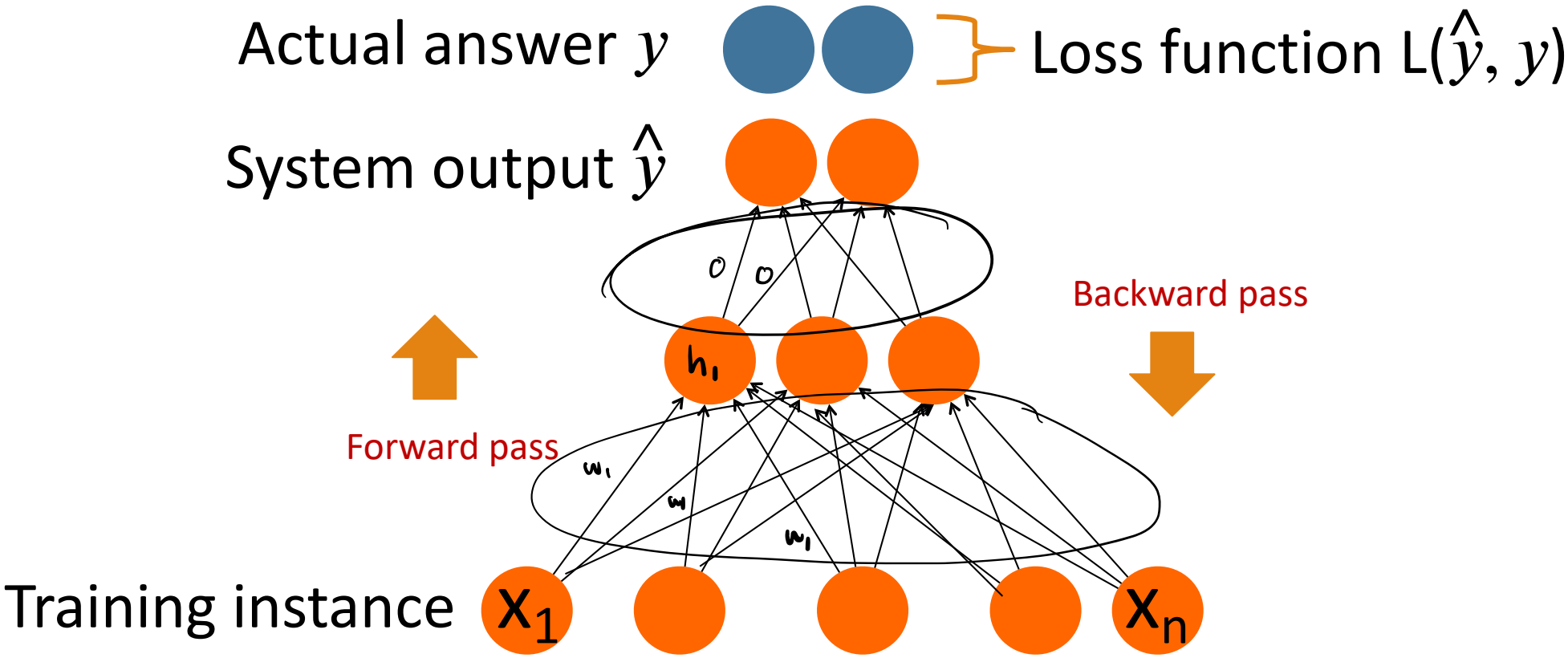
We'll pick up on this after spring break.

Training a Neural Network

Intuition: training a 2-layer Network



Intuition: training a 2-layer Network



Intuition: Training a 2-layer network

For every training tuple (x, y)

- Run *forward* computation to find our estimate $\hat{y}$
- Run *backward* computation to update weights:
  - For every output node
    - Compute loss L between true y and the estimated $\hat{y}$
    - For every weight w from hidden layer to the output layer
      - Update the weight
  - For every hidden node
    - Assess how much blame it deserves for the current answer
    - For every weight w from input layer to the hidden layer
      - Update the weight

Loss Function: a measure of how far off the current answer is from the right answer.

For binary logistic regression, we use cross entropy loss:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

$$L_{\text{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

For multinomial classification, we use cross entropy loss:

$$L_{CE}(\hat{y}, y) = -\log \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{K} \exp(\mathbf{z}_j)} \quad \text{(where } i \text{ is the correct class)}$$

Gradient descent for weight updates

The derivative of the loss function with respect to weights tells us how to adjust the weights to make better predictions.

Derivative of the loss function: $\frac{\partial L(f(x;\theta), y)}{\partial w}$

We want to move the weights in the opposite direction of the gradient:

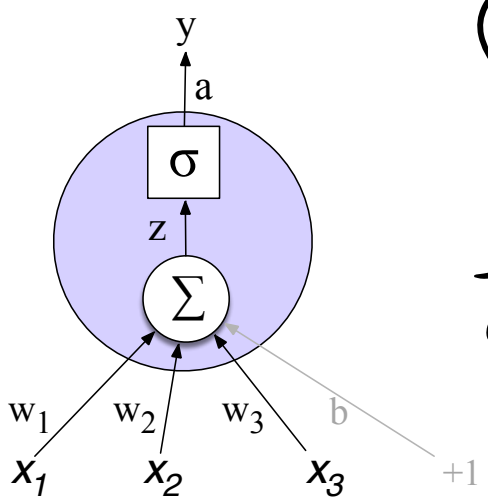
$$w_{t+1} = w_t - \gamma \frac{\partial L(f(x;\theta), y)}{\partial w_t}$$

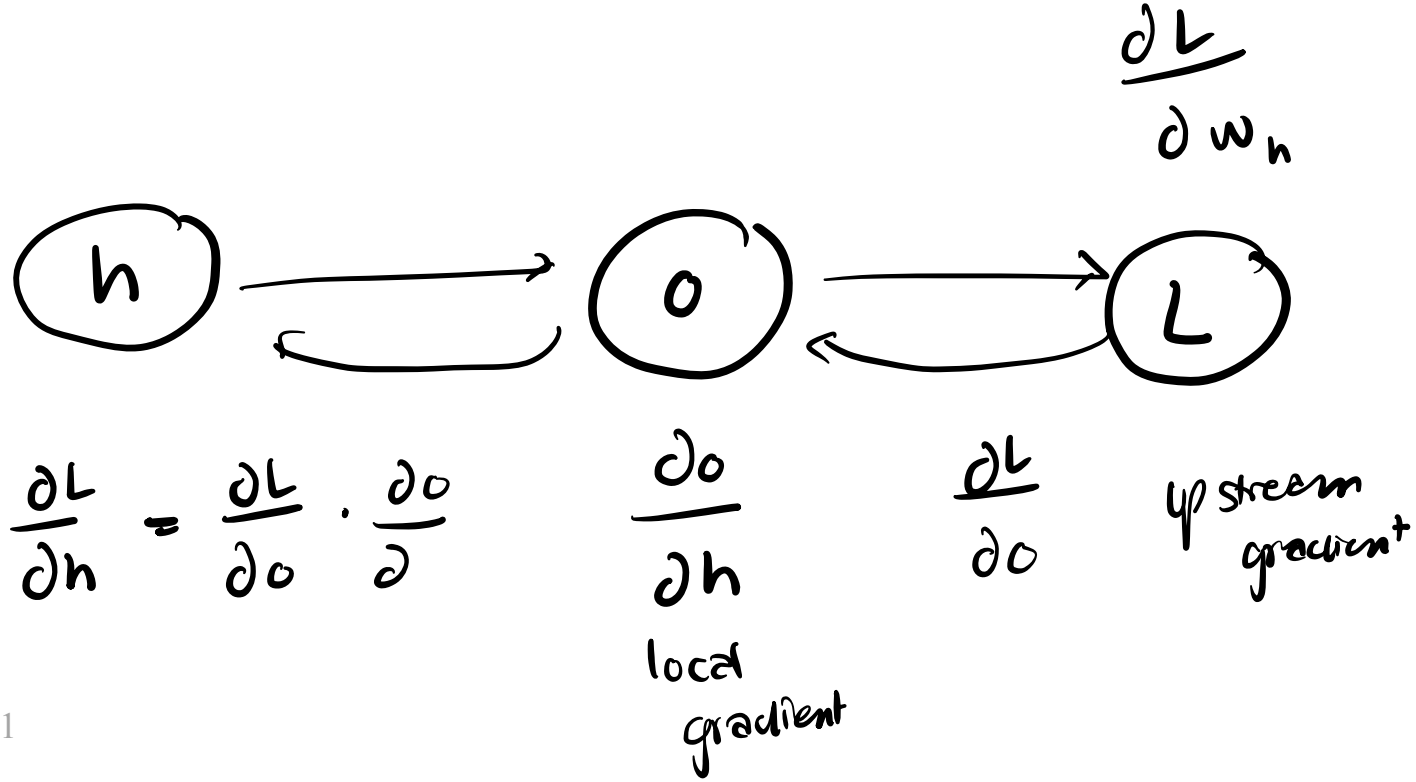
For logistic regression:

$$\frac{\partial L_{CE}(y, \hat{y}, x, \theta)}{\partial w_i} = (\hat{y} - y) x_i$$

$$= (\sigma(wx + b) - y) x_i$$

Where did that derivative come from?





Each node takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node.

A node may have multiple local gradients if it has multiple inputs.

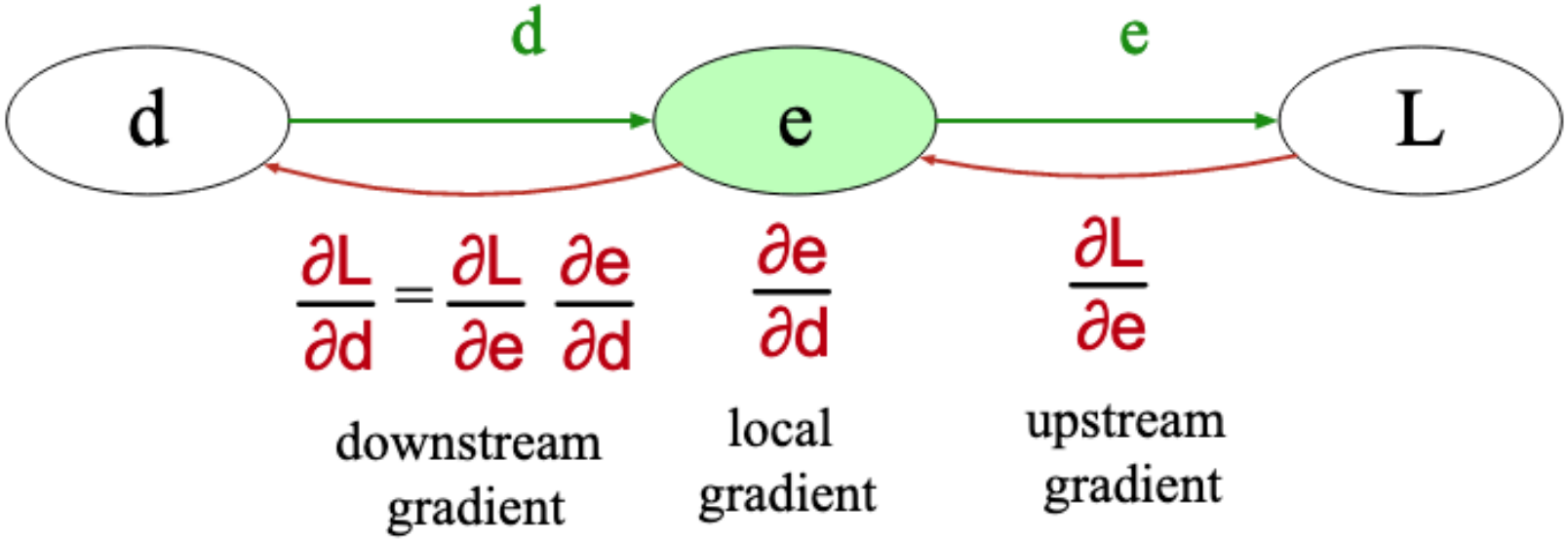
The chain rule

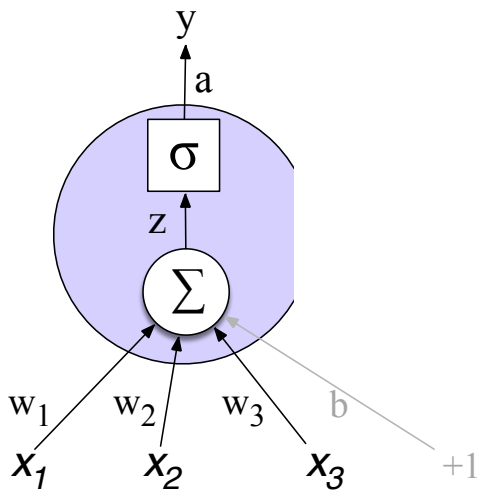
Computing the derivative of a composite function:

$f(x) = u(v(x))$ $$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$f(x) = u(v(w(x)))$ $$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

Where did that derivative come from?





Each node takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node.

A node may have multiple local gradients if it has multiple inputs.

Why Computation Graphs

For training, we need the derivative of the loss with respect to each weight in every layer of the network.

Problem: the derivatives on the prior slide only give the updates for one weight layer: the last one, since loss is computed only at the very end of the network!

Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)

- Backprop is a special case of *backward differentiation*

Can we get back to cat pics, please?

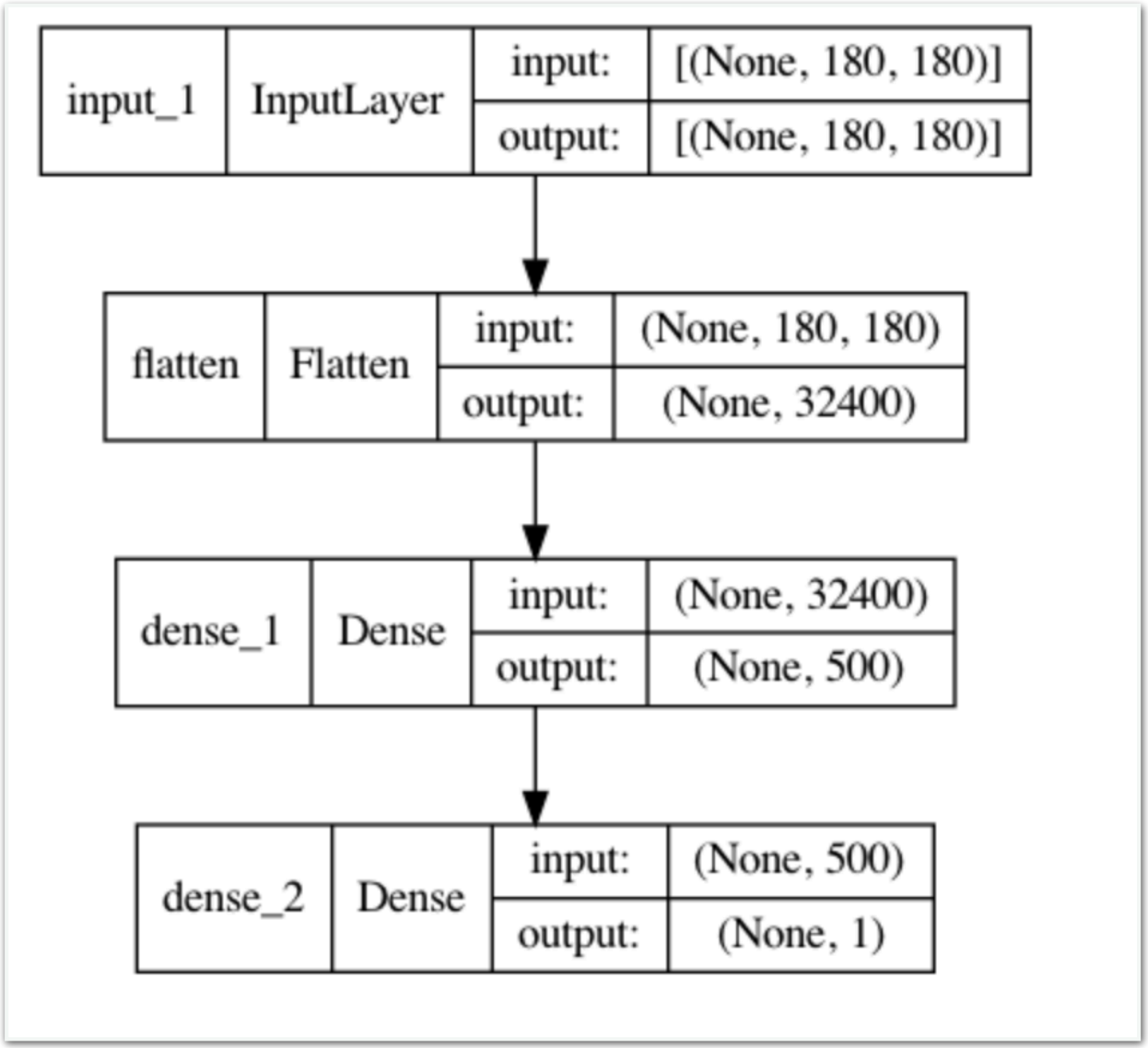
Finally, we're ready to power up our supervised cat/dog classifier by **adding more layers**. This takes it from a **regression model** to a **neural network**.

Adding More Layers

```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    x = layers.Flatten()(inputs)
    h1 = layers.Dense(1000)(x)
    h2 = layers.Dense(500, activation="relu")(h1)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes
    outputs = layers.Dense(units, activation=activation)(h2)
    return keras.Model(inputs, outputs)


model = make_model(input_shape=image_size, num_classes=2)
keras.utils.plot_model(model, show_shapes=True)
```

New Architecture



Computation Graphs

Why Computation Graphs

For training, we need the derivative of the loss with respect to each weight in every layer of the network.

Problem: the derivatives on the prior slide only give the updates for one weight layer: the last one, since loss is computed only at the very end of the network!

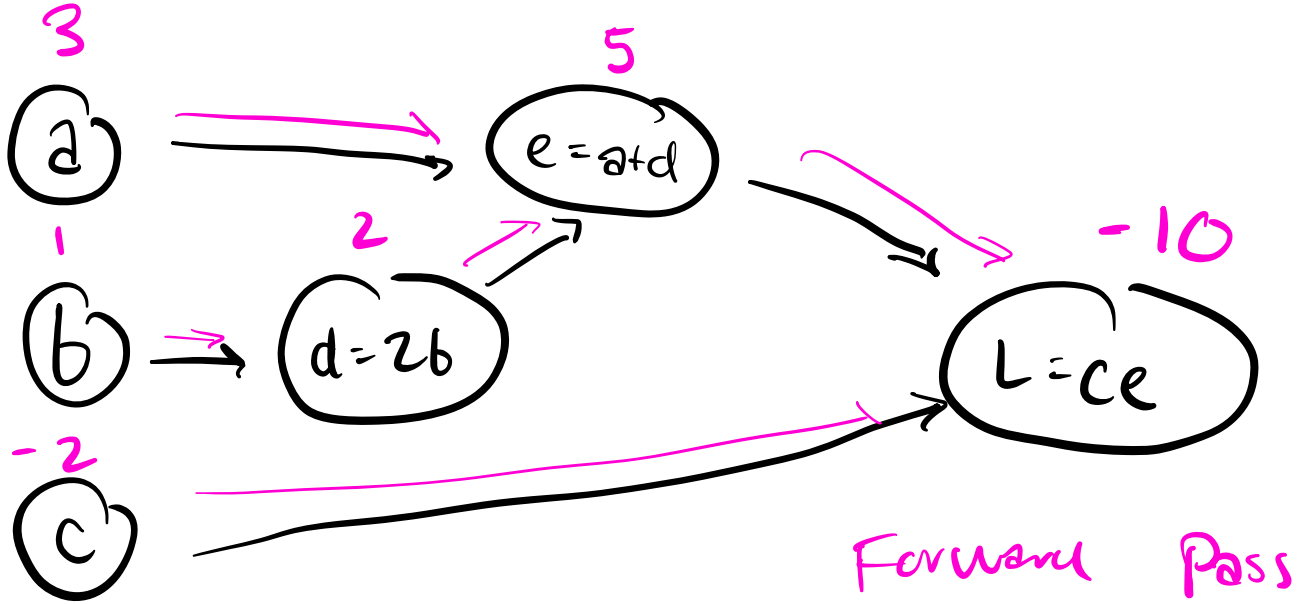
Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)

- Backprop is a special case of backward differentiation
- Which relies on **computation graphs**.

Computation Graphs

A computation graph represents the process of computing a mathematical expression

$$L(a,b,c) = c(a+2b)$$



Computations:

$d = 2b$

$e = a + d$

$L = ce$

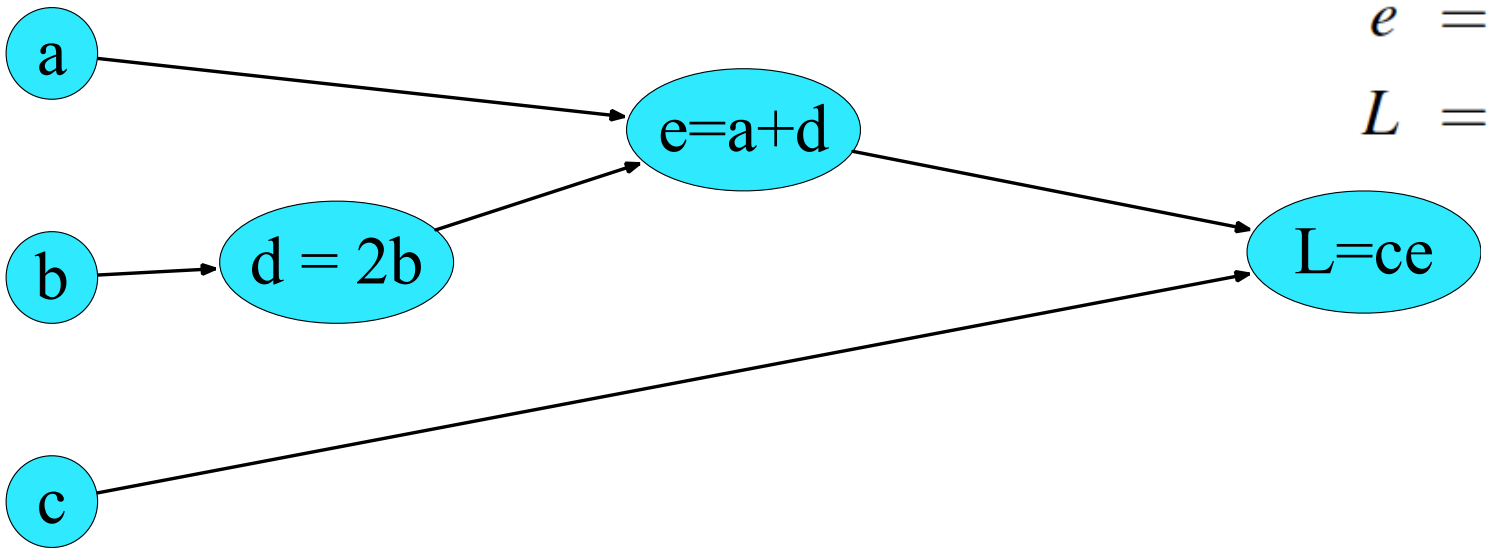
Computation Graphs

A computation graph represents the process of computing a mathematical expression

$$L(a,b,c) = c(a+2b)$$

Computations:

$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$

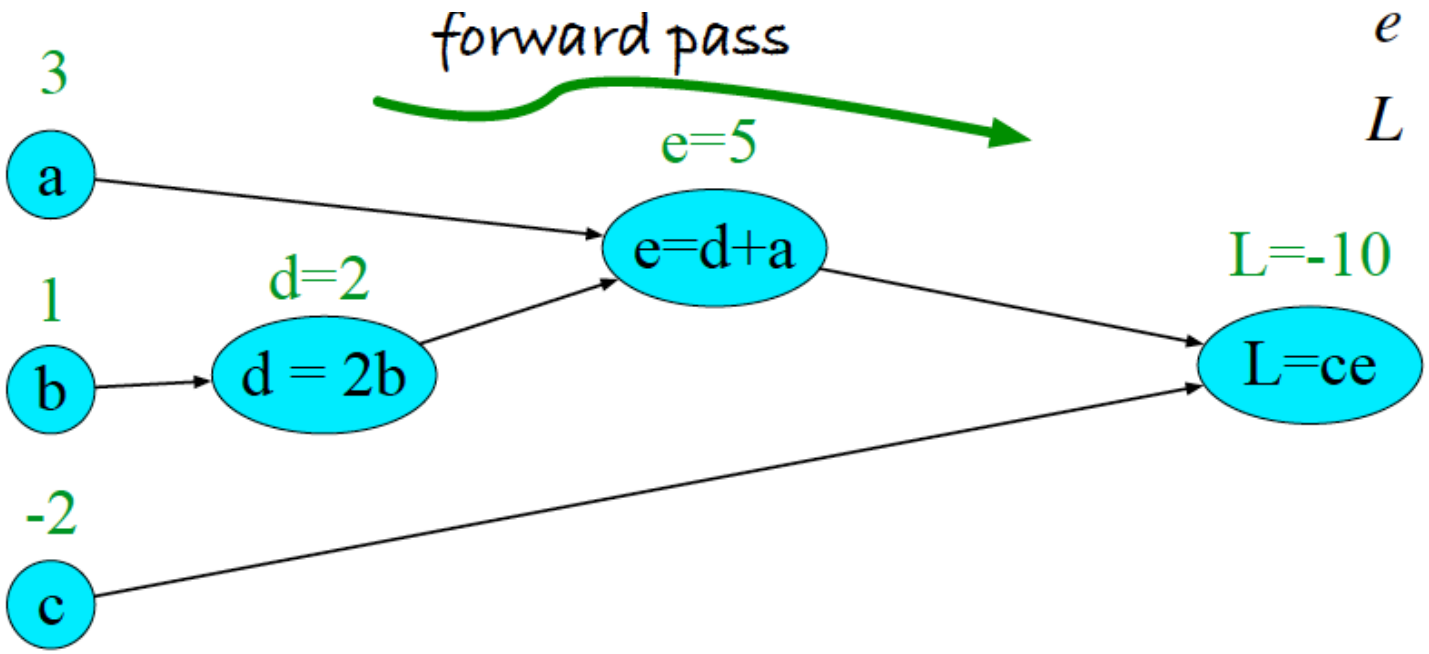


Computation Graphs

$$L(a,b,c) = c(a+2b)$$

Computations:

$$d = 2 * b$$
$$e = a + d$$
$$L = c * e$$



Backwards differentiation in computation graphs

The importance of the computation graph comes from the backward pass

This is used to compute the derivatives that we'll need for the weight update.

The chain rule

Computing the derivative of a composite function:

$f(x) = u(v(x))$ $\qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$

$f(x) = u(v(w(x)))$ $\qquad \frac{\partial f}{\partial x} = \frac{\partial u}{\partial v} \cdot \frac{\partial v}{\partial w} \cdot \frac{\partial w}{\partial x}$

The chain rule

Computing the derivative of a composite function:

$f(x) = u(v(x))$ $$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$f(x) = u(v(w(x)))$ $$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

Example

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \cdot \frac{\partial e}{\partial a} \qquad L = ce$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \cdot \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}$$

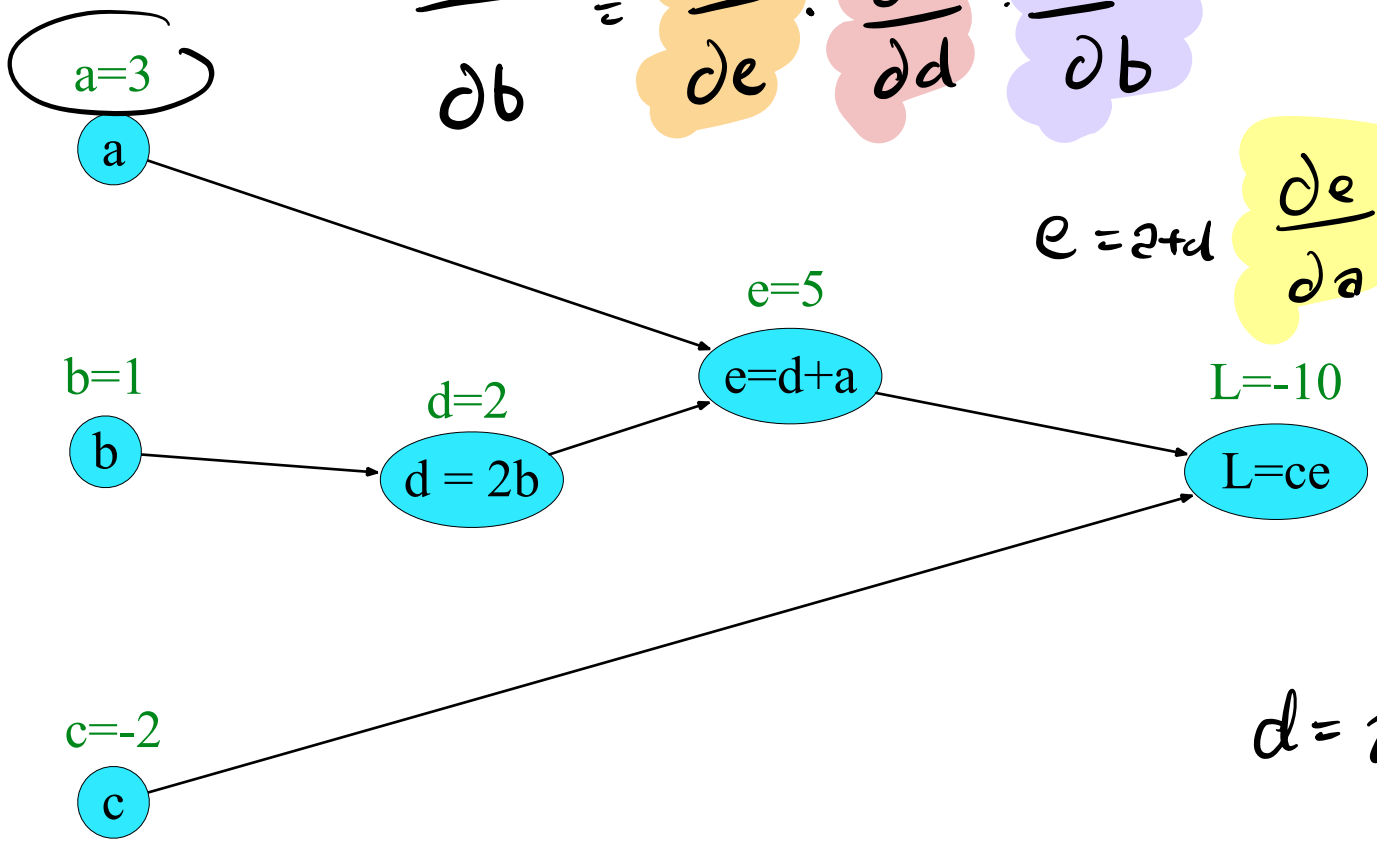
$$\frac{\partial L}{\partial e} = c$$

$$\frac{\partial L}{\partial c} = e$$

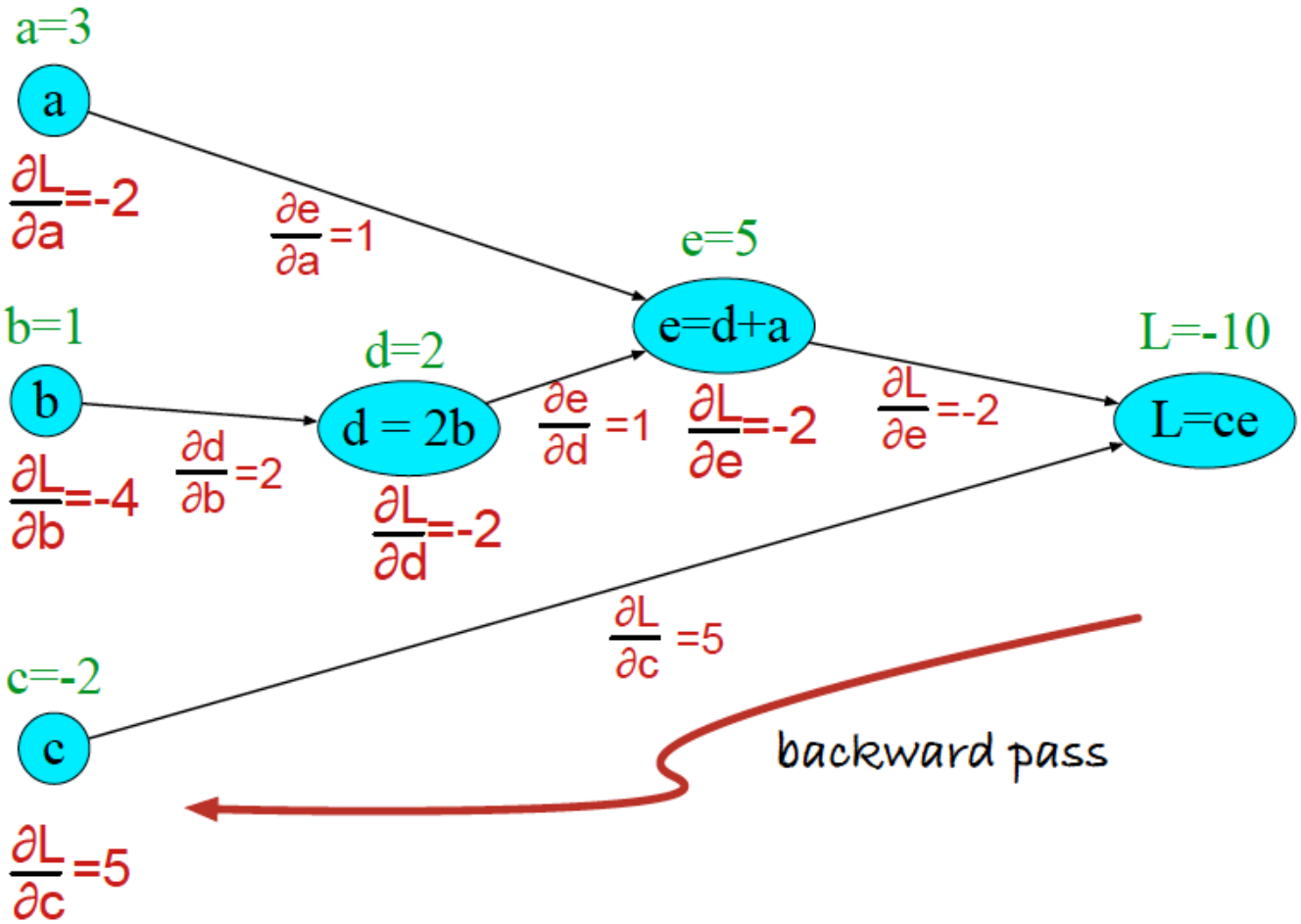
$$e = a + d \qquad \frac{\partial e}{\partial a} = 1$$

$$\frac{\partial e}{\partial d} = 1$$

$$d = 2b \qquad \frac{\partial d}{\partial b} = 2$$



Example


a=3
a
∂L/∂a=-2
∂e/∂a=1
e=5
e=d+a
b=1
b
d=2
d = 2b
∂e/∂d=1
∂L/∂e=-2
∂L/∂e=-2
L=-10
L=ce
∂d/∂b=2
∂L/∂b=-4
∂L/∂d=-2
∂L/∂c=5
c=-2
c
backward pass
∂L/∂c=5

Summary

For training, we need the derivative of the loss with respect to weights in early layers of the network

- But loss is computed only at the very end of the network!

Solution: **backward differentiation**

Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.