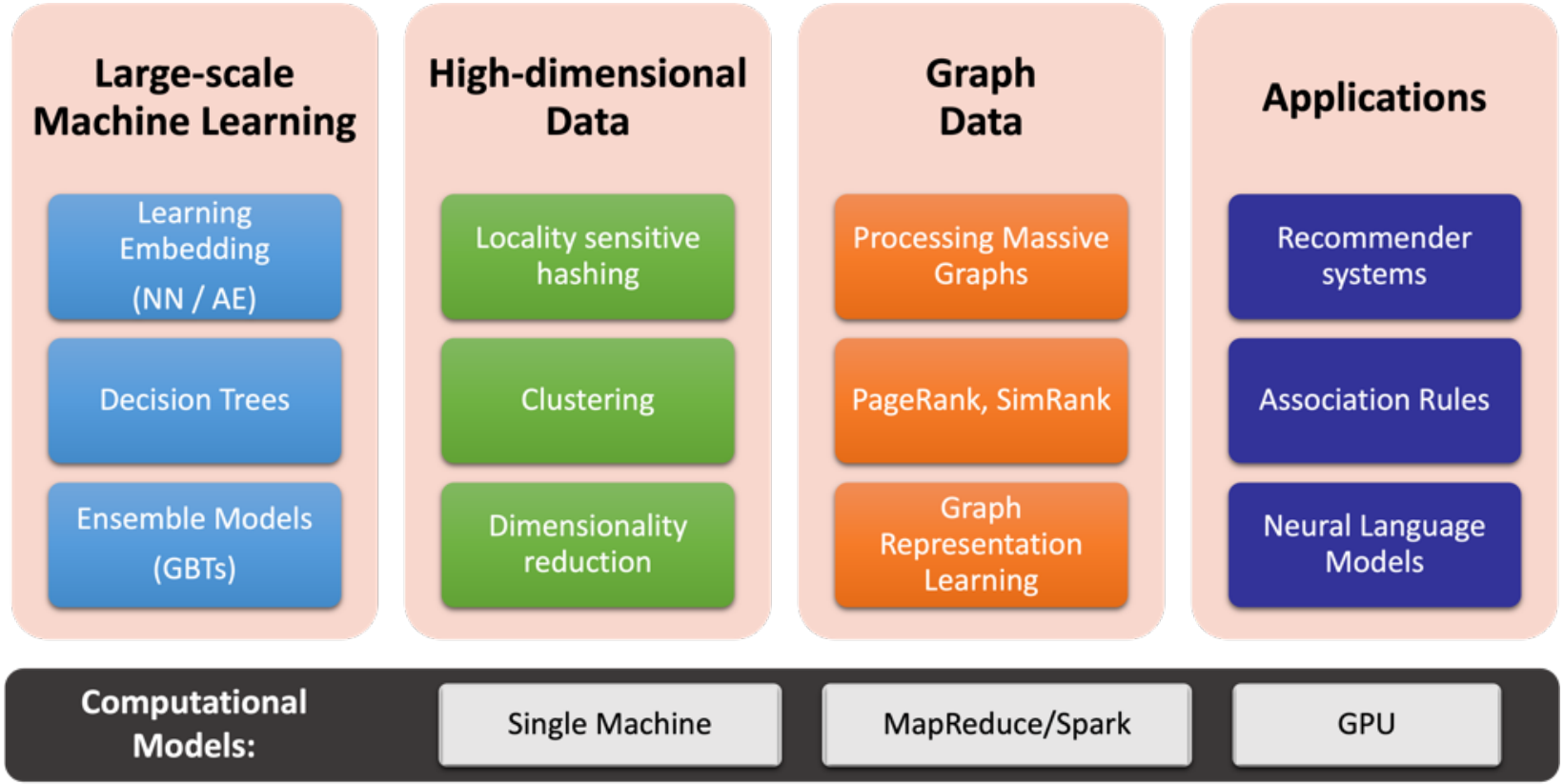
MIE524 Data Mining

Neural Networks

Slides Credits:
Slides from Leskovec, Rajaraman, Ullman (http://www.mmds.org),
Dan Jurafsky & James H. Martin (https://web.stanford.edu/~jurafsky/slp3/),
Roger Grosse, Stephen Scott, Geoffrey Hinton

MIE524: Course Topics (Tentative)



Agenda

- **Motivation: why embeddings?**
- Neural network fundamentals
- Training Neural Networks
- Computation Graphs and Backward Differentiation
- Why is this scalable?
- The XOR problem
- Learning embeddings: Autoencoders

Using slides by:
Jure Leskovec & Mina Ghashami, Dan Jurafsky & James H. Martin,
Roger Grosse, Stephen Scott, Geoffrey Hinton

What is Machine Learning?

- Machine learning is about *Optimization*
- Three key components:
  1. Training Data $\mathrm{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
  2. Loss function $\mathcal{L}$
  3. Model $f_\theta(x)$
- Optimize $f_\theta(x)$ on D w.r.t loss function $\mathcal{L}$:
  - find the parameter θ that minimizes the expected loss on the training data

$$\min_f J(f) = \min_f \frac{1}{\mathrm{n}} \sum_{i=1}^{\mathrm{n}} \mathcal{L}(f_\theta(x_i), y_i)$$

Two Dominant ML Paradigms

- **Supervised learning**:
  - Given "labeled data" $\{x, y\}$, learn $f(x) = y$
  - Ex: classification, regression
    - In linear regression, the model $f_\theta(x) = Wx + b$
    - Parameters are $\theta = \{W, b\}$
    - The loss function is mean square error (MSE)

- **Unsupervised learning**:
  - Given only "unlabeled data" $\{x\}$, learn $f(x)$
  - Ex: Dimensionality reduction, clustering

Will be covered in future lecture

Input Feature Vectors

- All ML methods work with the input feature vectors $\{x_1, x_2, \ldots, x_n\}$ and almost all of them require input features to be numerical

- From ML perspective, there are four types of features:
  - Numerical (continues or discrete)
    - Continues: height
    - Discrete: age
  - **Categorical** (ordinal or nominal)
    - Ordinal: level={beginner, intermediate, advanced}
    - Nominal: gender={male, female}, color={red, blue, green}
  - Time series:
    - Average of home sale price over years
  - Text
    - Bag of words

Categorical Features

- There are two ways to encode categorical var:
  - Integer encoding
  - One-hot encoding (and multi-hot encoding)
- Consider the following movie dataset:

Title	provider	IMDB genres	Release year	IMDB rating
Stranger Things	Netflix	drama, fantasy, horror	2016	8.7
Cocomelon	Prime Video	animation, comedy, family	2019	4.7
100 Foot Waves	HBO Max	documentary, sport	2021	8.1
I, Tonya	Hulu	biography, drama, comedy	2017	7.5

Integer Encoding

- Assigns each category value with an integer
  - *provider :=[Netflix, Prime Video, HBO Max, Hulu],* we assign them integers 1, 2, 3 and 4 respectively.

Title	provider	IMDB genres	Release year	IMDB rating
Stranger Things	1	drama, fantasy, horror	2016	8.7
Cocomelon	2	animation, comedy, family	2019	4.7
100 Foot Waves	3	documentary, sport	2021	8.1
I, Tonya	4	biography, drama, comedy	2017	7.5

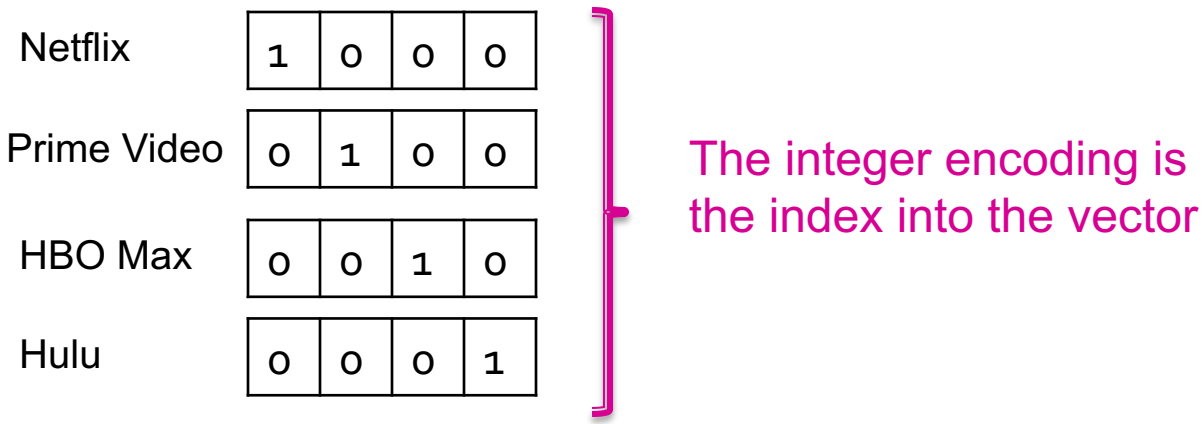
- Pros: dense representation
- Cons: It implies ordering between different categories:
  Netflix < Prime Video < HBO Max < Hulu

- Makes more sense to use it for ordinal variables:
  - Such as "Education"= {Diploma, Undergrad, Masters, PhD }
  - But still it implies values are equally spaced out

One-hot Encoding

- First do integer encoding, then create a binary vector that represents the numerical values
  - Ex: following integer encoding on provider:

    Netflix -> 1, Prime Video -> 2, HBO Max ->3 , Hulu -> 4
  - create a binary vector of length 4 for each value:



Multi-hot Encoding

- An extension of one-hot encoding when categorical variable can take multiple values at the same time
  - Ex: There are 28 distinct IMDB genres

a movie can take multiple genres, e.g. *stranger things is drama, fantasy, horror.*

IMDB genres

```
[(1, 'Action'),
 (2, 'Comedy'),
 (3, 'Short'),
 (4, 'Western'),
 (5, 'Drama'),
 (6, 'Horror'),
 (7, 'Music'),
 (8, 'Thriller'),
 (9, 'Animation'),
 (10, 'Adventure'),
 (11, 'Family'),
 (12, 'Fantasy'),
 (13, 'Sport'),
 (14, 'Romance'),
 (15, 'Crime'),
 (16, 'Sci-Fi'),
 (17, 'Biography'),
 (18, 'Musical'),
 (19, 'Mystery'),
 (20, 'History'),
 (21, 'Documentary'),
 (22, 'Film-Noir'),
 (23, 'News'),
 (24, 'Game-Show'),
 (25, 'Reality-TV'),
 (26, 'War'),
 (27, 'Talk-Show'),
 (28, 'Adult')]
```

Stranger things	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cocomelon	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
100 foot wave	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
I, Tonya	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Applying encodings on Movies dataset

	provider				IMDB genres																																	
Stranger things	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2016	8.7
cocomelon	0	1	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2019	4.7
100 foot waves	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2021	8.1
I, Tonya	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2017	7.5

- Data dimensions increased from 5 to 35. It will blow up to thousands or a million if we multi-hot encode title!

- One-hot and multi-hot encodings are not practical for features with large value sets.

From encoding to embedding

- One-hot/multi-hot encodings:
  - pros: simple, robust, when trained on huge amounts of data they outperform complex systems trained on fewer data
  - cons: sparse and high dimensional, don't capture semantic similarity
- In a corpus of documents with one million distinct words:
  - high dimensional: multi-hot encodings are 1-million dimensional
  - Sparse: an average document contains 500 words therefore the multi-hot encodings are > 99.95% sparse
  - lack of semantic: encoding of two words 'good' and 'great' are as different as encoding of 'good' and 'bad'!
- An embedding is a translation of a high-dim vector into a low-dim space. An embedding is a:
  - Dense representation (floating-point value)
  - Low-dimensional vector
  - Captures semantic similarity

Word representation

V = [a, aaron, …, zulu, <UNK>]

$|V| = 10{,}000$

1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
O_{5391}	O_{9853}				

I want a glass of orange juice.

I want a glass of apple ?.

Andrew Ng

Featurized representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
	-1	1	-0.95	0.97	0.00	0.01
	0.01	0.02	0.93	0.95	-0.01	0.00
	0.03	0.02	0.7	0.69	0.03	-0.02
	0.04	0.01	0.02	0.01	0.95	0.97
	⋮	⋮				

e_{5391} e_{9853}

I want a glass of orange juice.

I want a glass of apple juice.

Featurized representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮				
Size						
Cost						
Alive						
Verb						

300

e_{5391} e_{9853}

I want a glass of orange juice.

I want a glass of apple juice.

Andrew Ng

From encoding to embedding

- State of the art embedders are among neural networks
- Can we use neural networks to create non-linear embedding?

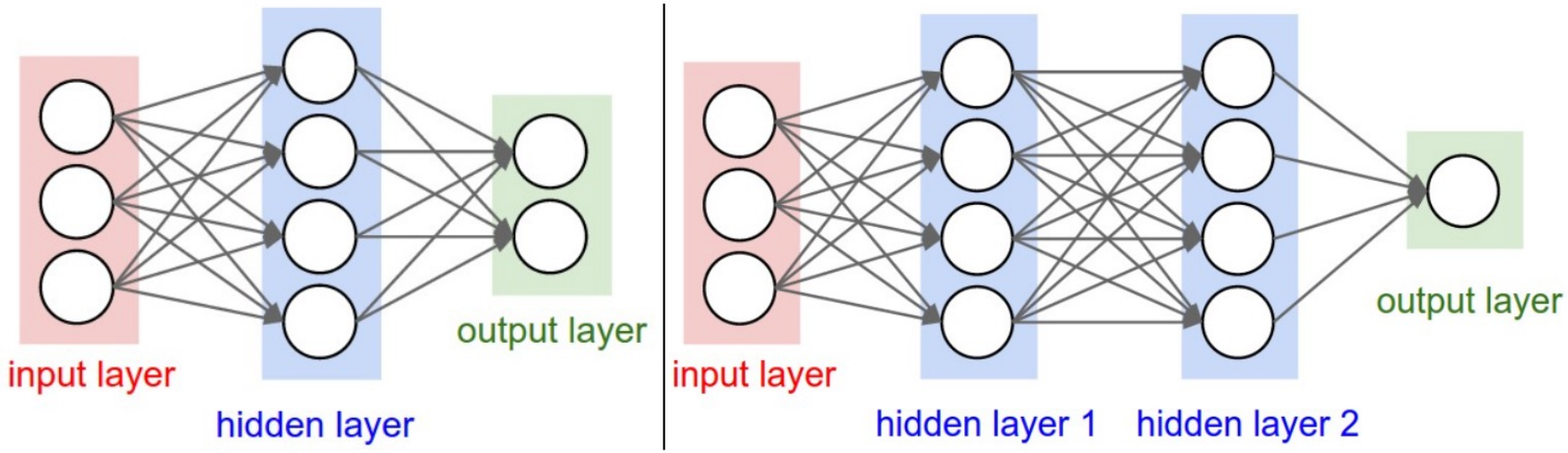
Agenda

- Motivation: why embeddings?
- **Neural network fundamentals**
- Training Neural Networks
- Computation Graphs and Backward Differentiation
- Why is this scalable?
- The XOR problem
- Learning embeddings: Autoencoders

Using slides by:
Jure Leskovec & Mina Ghashami, Dan Jurafsky & James H. Martin,
Roger Grosse, Stephen Scott, Geoffrey Hinton

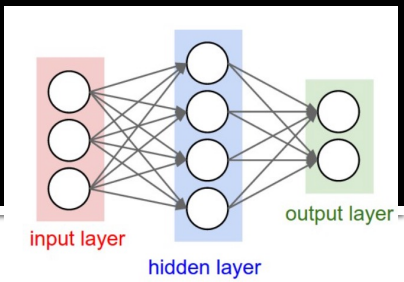
Neural Network: Architecture

- A neural network is a collection of neurons that are connected in an *acyclic graph*
- Outputs of some neurons are inputs to other neurons, and they are organized into layers



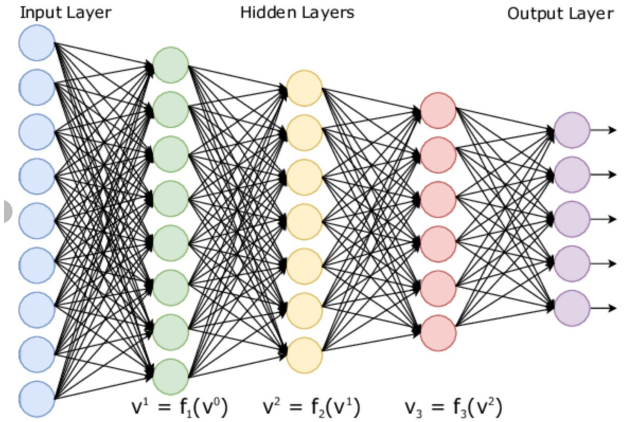
credit: cs231

Neural Network



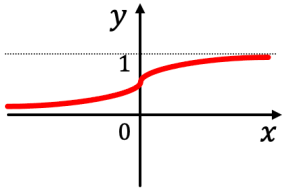
- **Fully-connected layer** is the most common layer type:
  - neurons between two adjacent layers are fully pairwise connected
  - neurons within a single layer share no connections

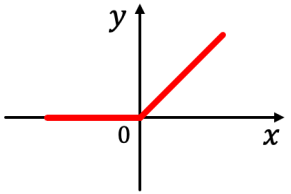
- Number of hidden layers and neurons in each hidden layer are hyperparameters of the network



Neural Network: A Neuron

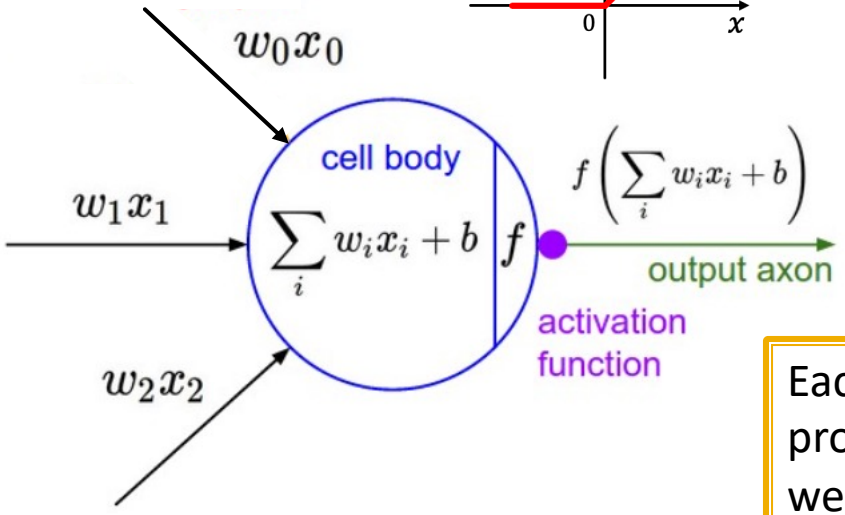
- A neuron
  - Input: $[x_0, x_1, x_2]$
  - Output = $f(\sum w_i x_i + b)$





f is the activation function, It takes a single number and performs an operation on it. Some choices are:

1. Sigmoid $\sigma(x) = 1/(1 + e^{-x})$
2. Tanh $\tanh(x) = 2\sigma(2x) - 1$
3. Relu $f(x) = \max(0, x)$

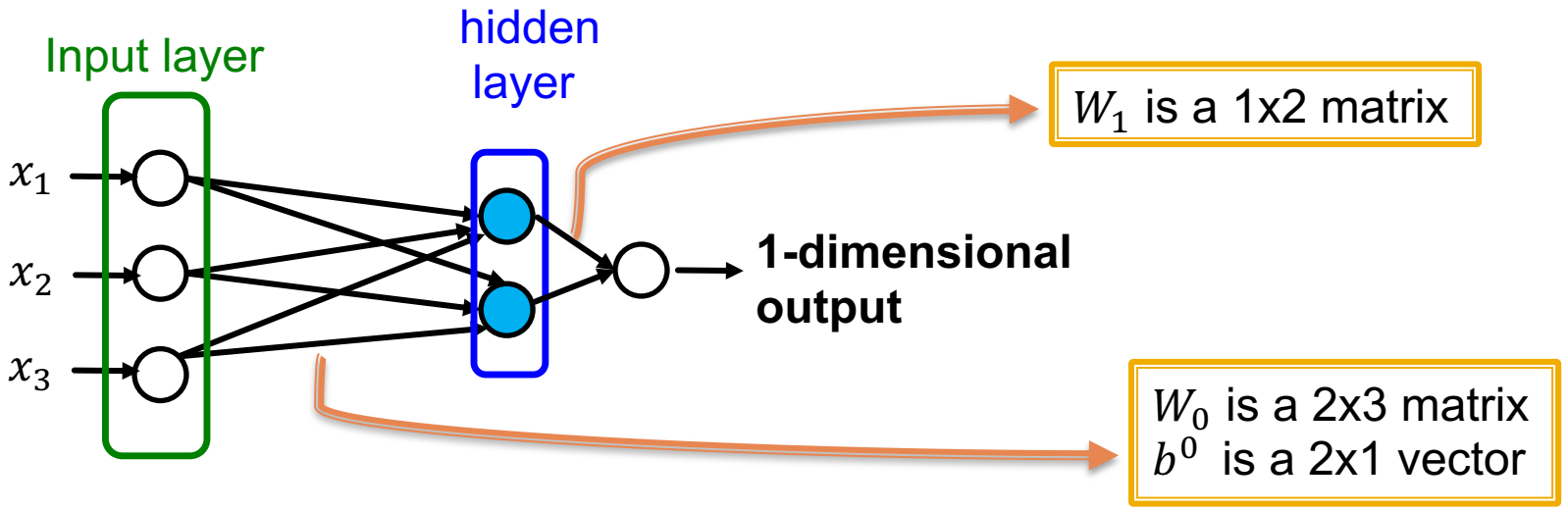


Each neuron performs a dot product with the input and its weights, adds the bias and applies the activation function

credit: cs231

Neural Network: A Layer

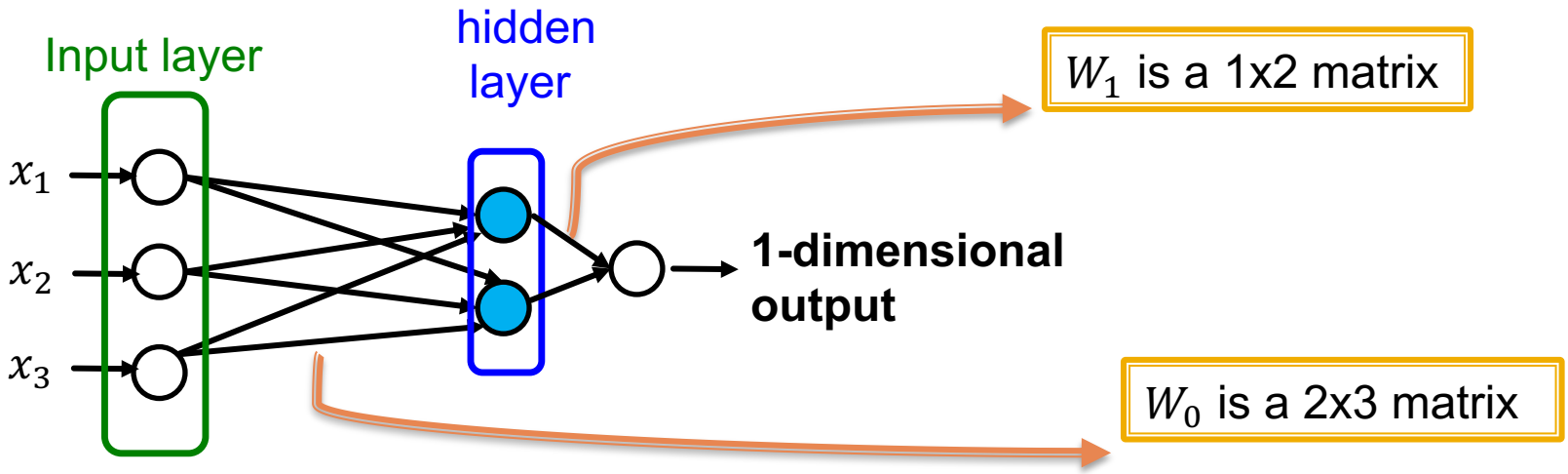
- Consider two neurons in a hidden layer



- Each layer computes $\boldsymbol{x}^{(l+1)} = \sigma(W_l \boldsymbol{x}^{(l)} + b^l)$
- W_l is weight matrix that transforms representation at layer l to layer $l+1$
- b^l is bias at layer l, and is added to the linear transformation of $\boldsymbol{x}$
- σ is sigmoid activation function

Neural Network: A Layer

- This network computes $f(x) = W_1(\sigma(W_0\boldsymbol{x}^{(0)} + b^0) + b^1)$



- Notice without activation functions, *f(x) will be linear in x* !!

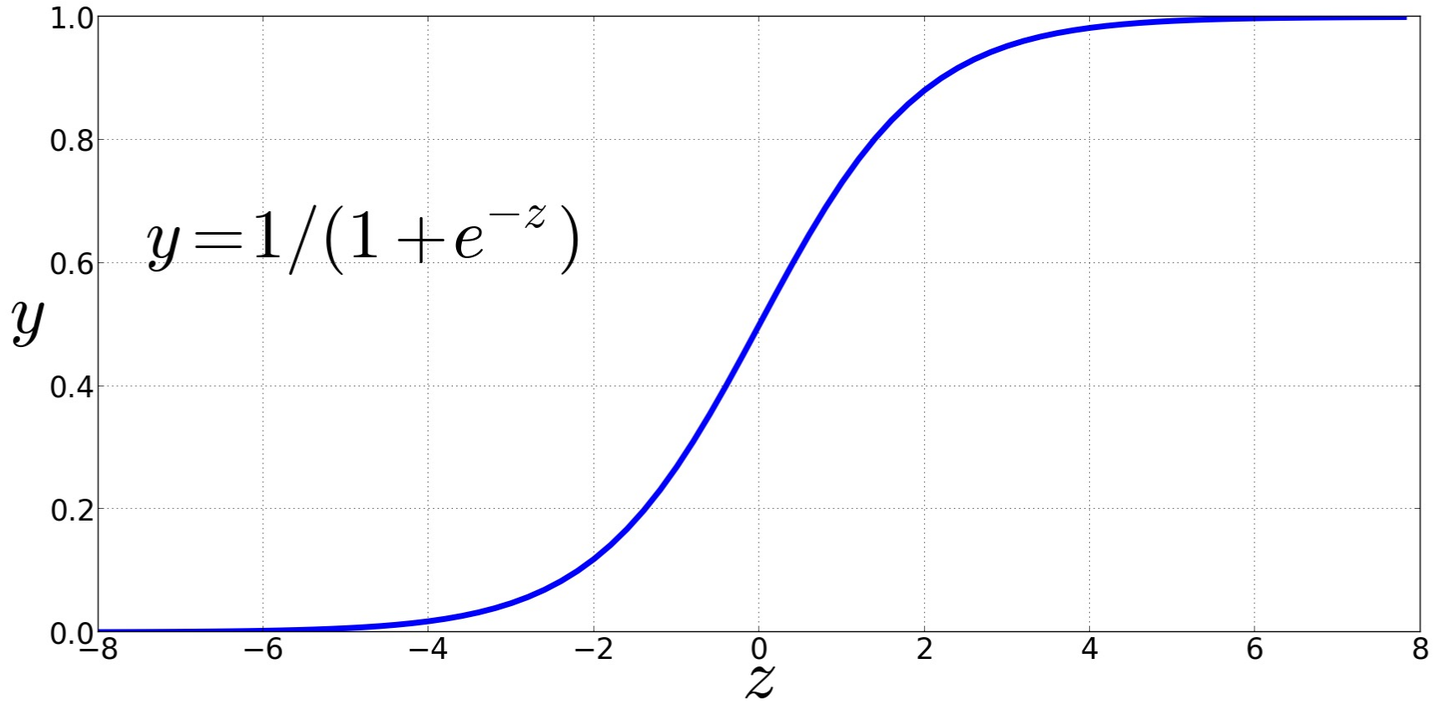
$$f(x) = W_1W_0\boldsymbol{x}^{(0)} + W_1b^0 + b^1$$

Non-Linear Activation Functions

We're already seen the sigmoid for logistic regression:

Sigmoid

$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$



Final function the unit is computing

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))}$$

Final unit again

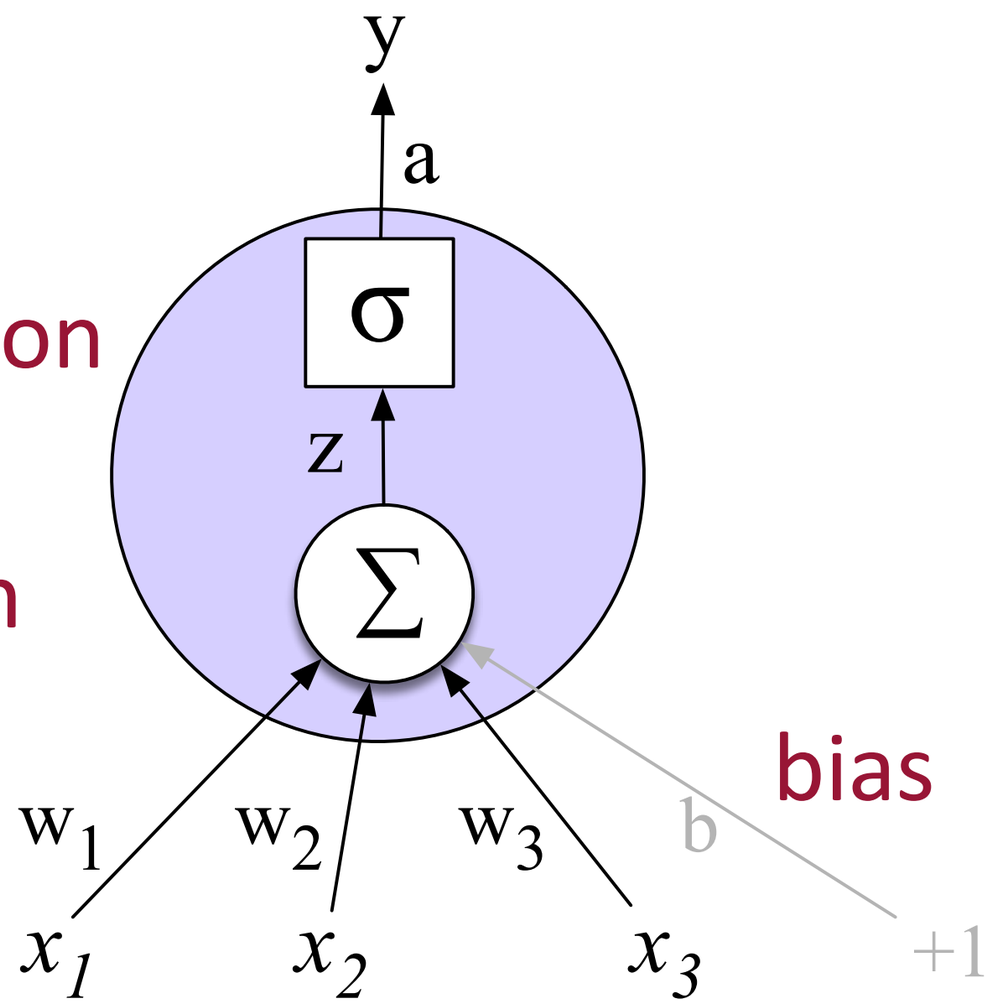
Output value

Non-linear activation function

Weighted sum

Weights

Input layer



An example

Suppose a unit has:

`w = [0.2,0.3,0.9]`

`b = 0.5`

What happens with input x:

`x = [0.5,0.6,0.1]`

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

An example

Suppose a unit has:

w = [0.2,0.3,0.9]

b = 0.5

What happens with the following input x?

x = [0.5,0.6,0.1]

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

An example

Suppose a unit has:

w = [0.2,0.3,0.9]

b = 0.5

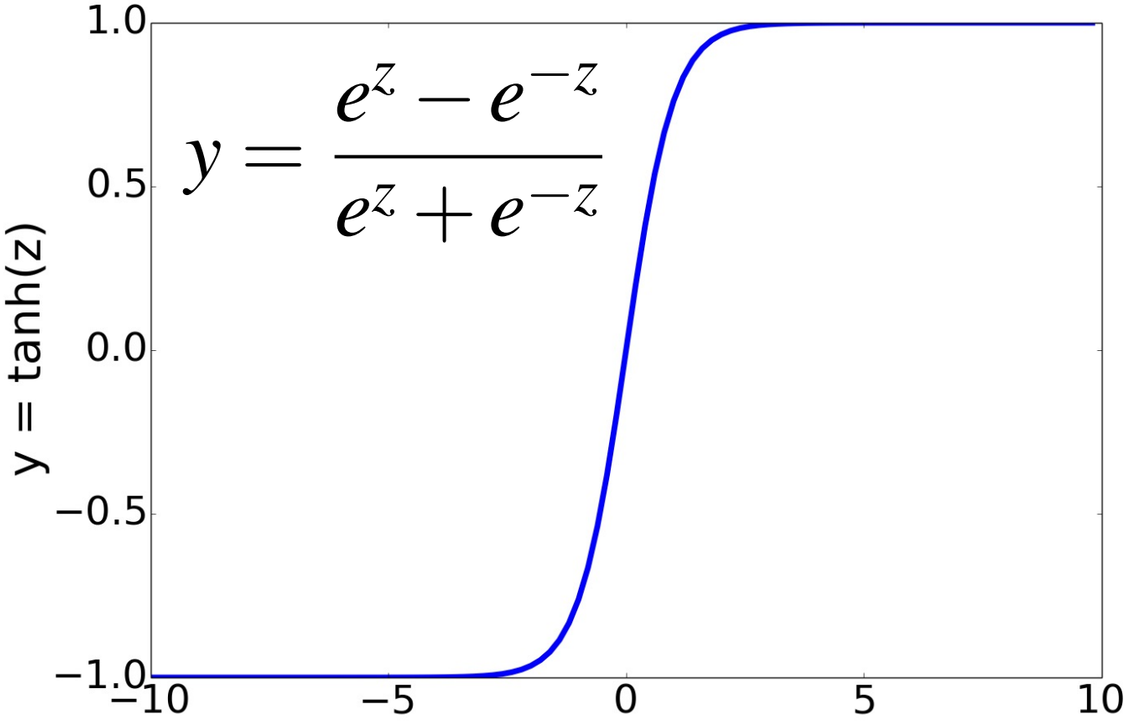
What happens with input x:

x = [0.5,0.6,0.1]

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

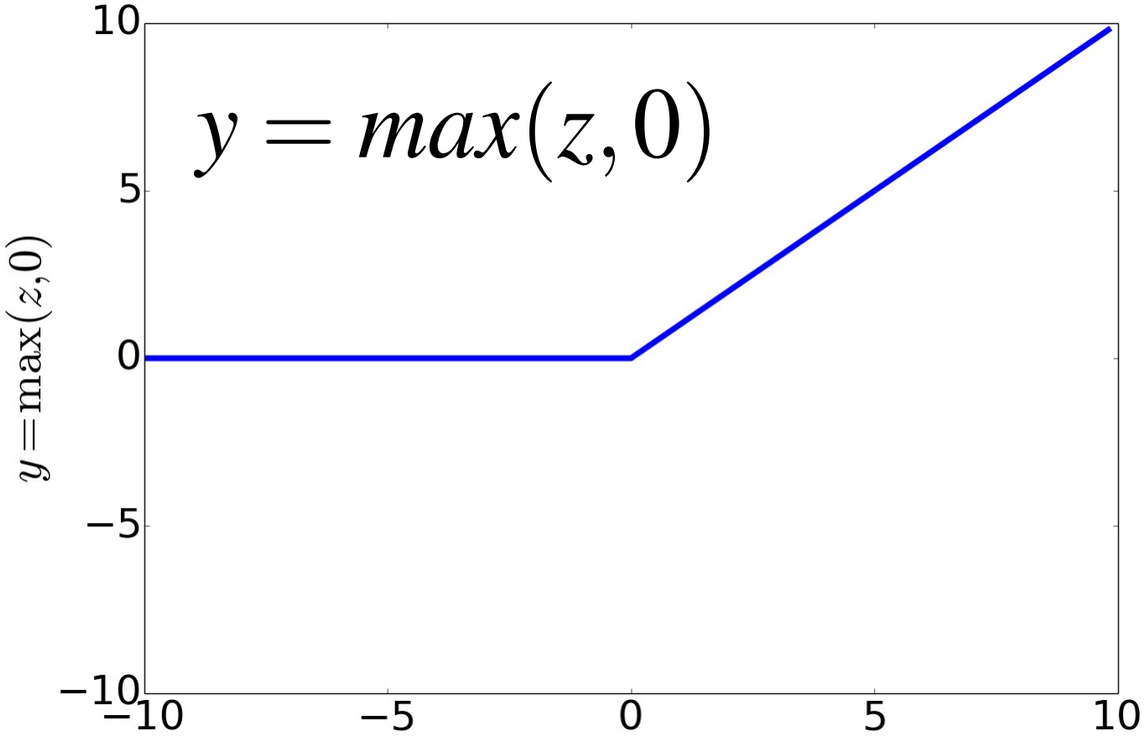
$$\frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

Non-Linear Activation Functions besides sigmoid



tanh

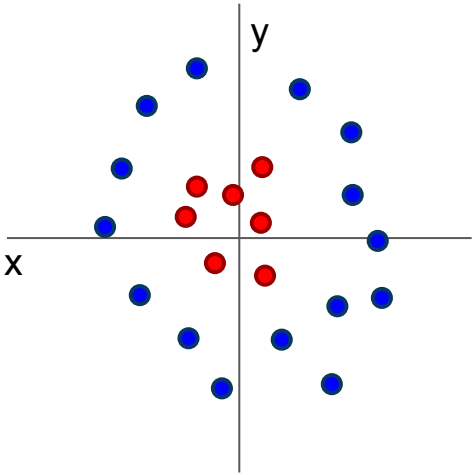
Most Common:



ReLU

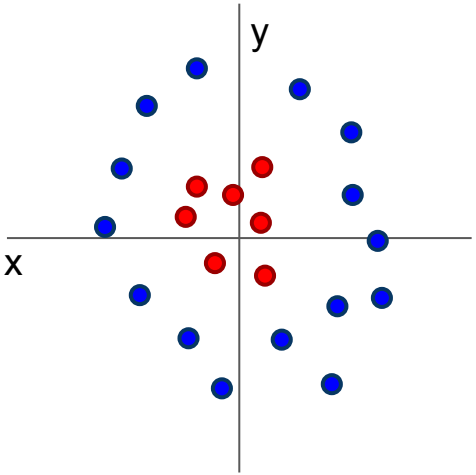
Rectified Linear Unit

Why do we want non-linearity?



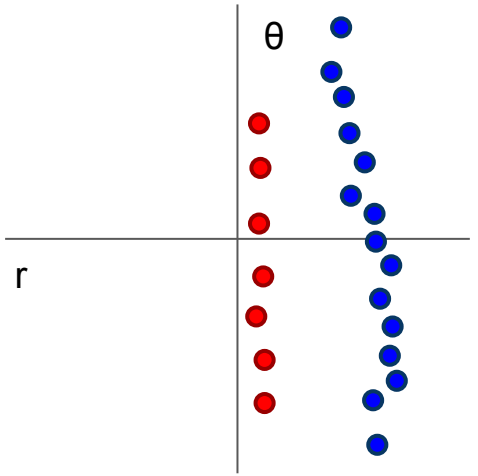
Cannot separate red and blue points with linear classifier

Why do we want non-linearity?



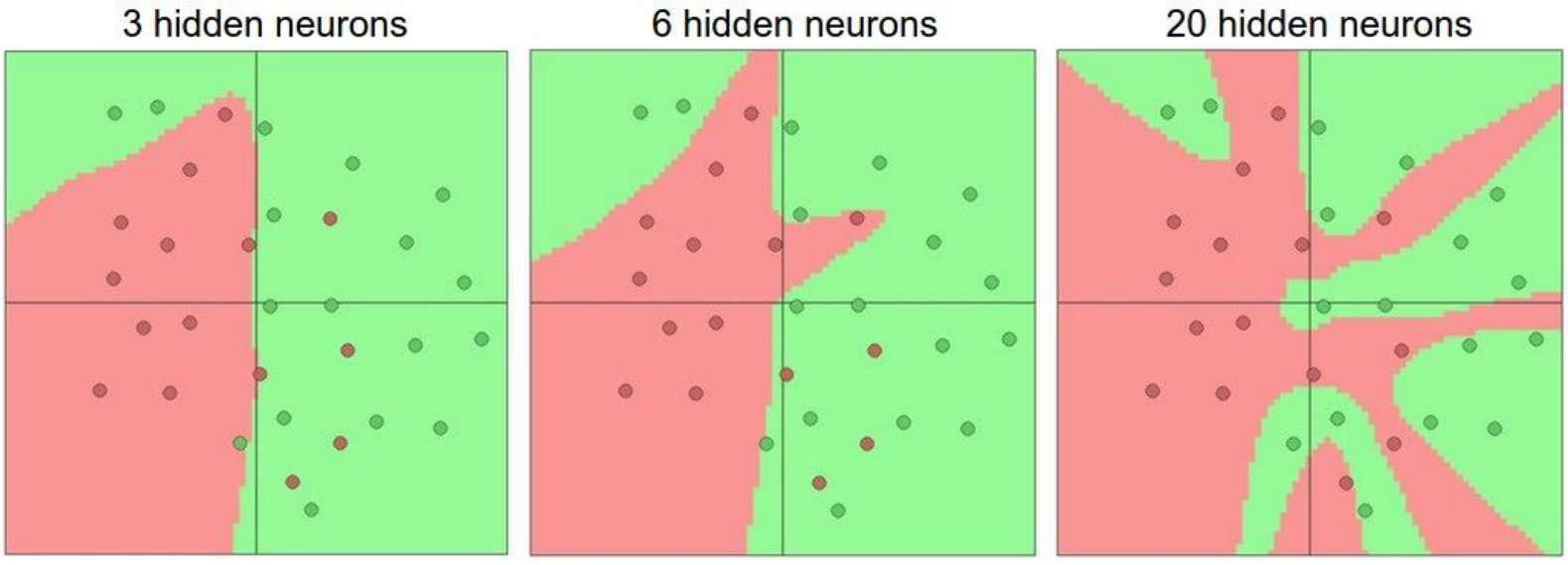
Cannot separate red and blue points with linear classifier

$$f(x, y) = (r(x, y), \theta(x, y))$$



After applying feature transform, points can be separated by linear classifier

Setting the number of layers and their sizes



more neurons = more capacity

Neural Network: Loss Function

- A loss function $\mathcal{L}$ is required to train the NN.
  - Example: L2 loss

$$\mathcal{L}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = \|y - f(x)\|_2$$

- Common loss functions for **regression**:
  - L2 loss, L1 loss, huber loss, …
- Common loss functions for **classification**:
  - Cross entropy, max margin (hinge loss), …
- Example
  - See https://pytorch.org/docs/stable/nn.html#loss-functions

Cross Entropy Loss

- Common loss for classification tasks
  - Defined between one-hot of true label and the predicted probability distribution over classes

- Ex: Task = multi-class classification with 5 classes
  - True label y belongs to class 3, so one-hot of y = | 0 | 0 | 1 | 0 | 0 |
  - Predicted probability distribution $\hat{y} = f(x)$ = | 0.1 | 0.3 | 0.4 | 0.1 | 0.1 |

  - $\mathrm{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{C}(y_i \log \hat{y}_i) = -\log(\hat{y}_{\text{correct class}})$

  - y_i, $\hat{y}_i$ are the **actual** and **predicted** value of the i-th class.

- **Intuition:** the lower the loss, the closer the prediction is to one-hot $\boldsymbol{y}$

Cross Entropy Loss

- Often model's output is a score for each class, not a probability distribution
- To convert to probability distribution:

$$f(\boldsymbol{x}) = \text{Softmax}\big(g(\boldsymbol{x})\big)$$

Probability distribution over classes

Output score for each class

- $f(\boldsymbol{x})_i = \dfrac{e^{g(x)_i}}{\sum_{j=1}^{C} e^{g(x)_j}}$

Cross entropy loss sometimes is referred to as softmax loss

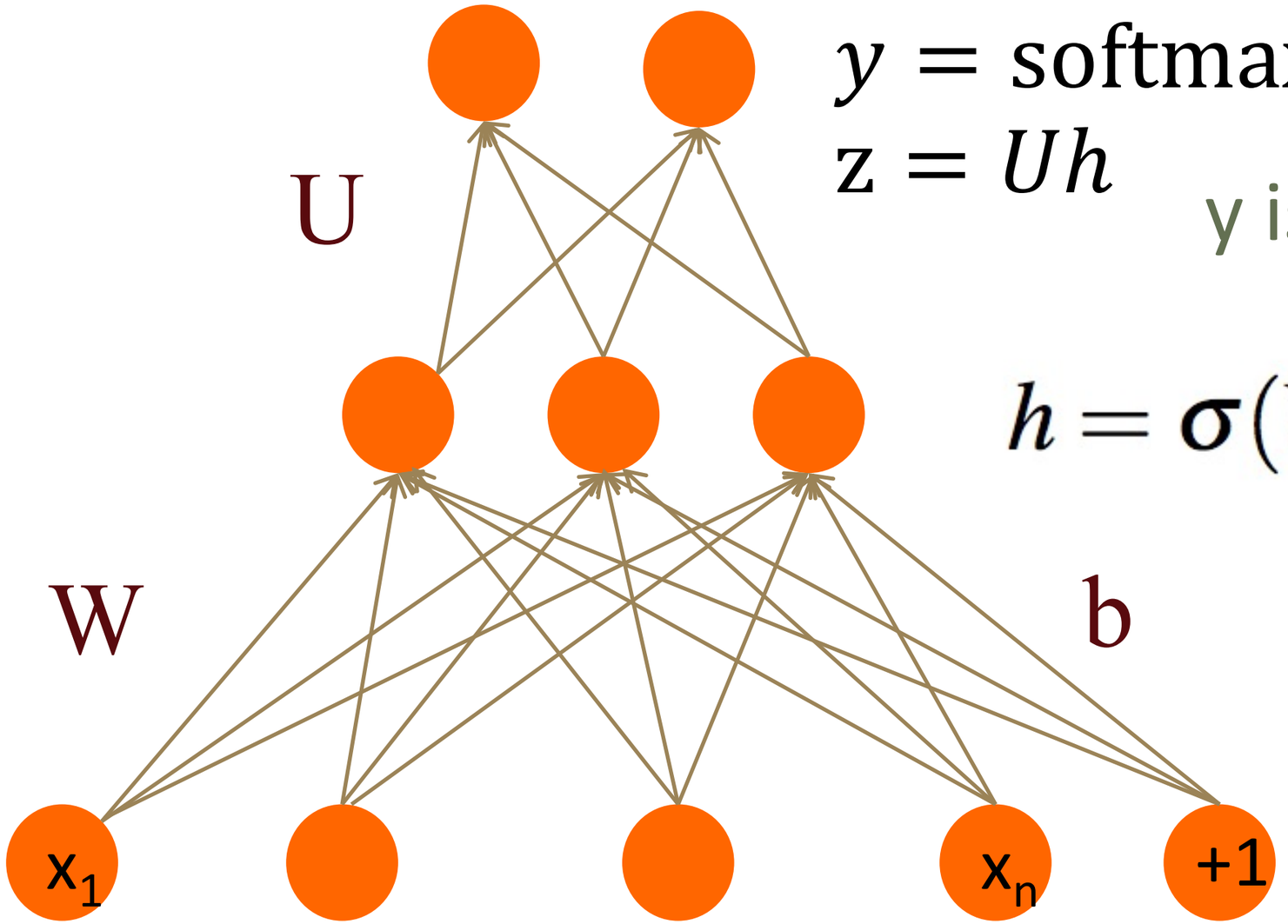
- It normalizes a vector into a probability distribution that sums to 1

Two-Layer Network with softmax output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$$y = \text{softmax}(z)$$
$$z = Uh$$

y is a vector

$$h = \sigma(Wx + b)$$

Could be ReLU
Or tanh

Reminder: softmax: a generalization of sigmoid

For a vector *z* of dimensionality *k*, the softmax is:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

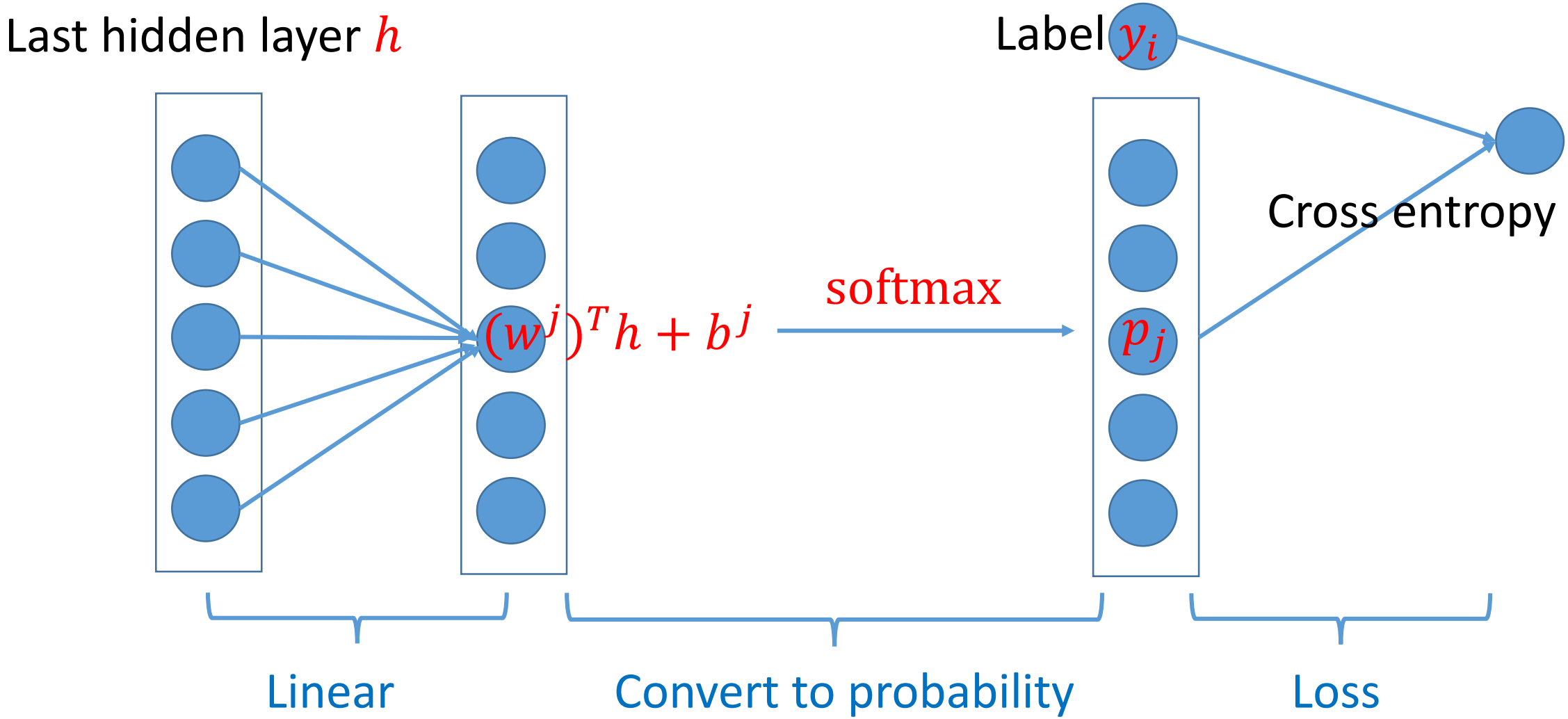
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

Example:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Multiclass logistic regression: summary



Agenda

- Motivation: why embeddings?
- Neural network fundamentals
- **Training Neural Networks**
- Computation Graphs and Backward Differentiation
- Why is this scalable?
- The XOR problem
- Learning embeddings: Autoencoders

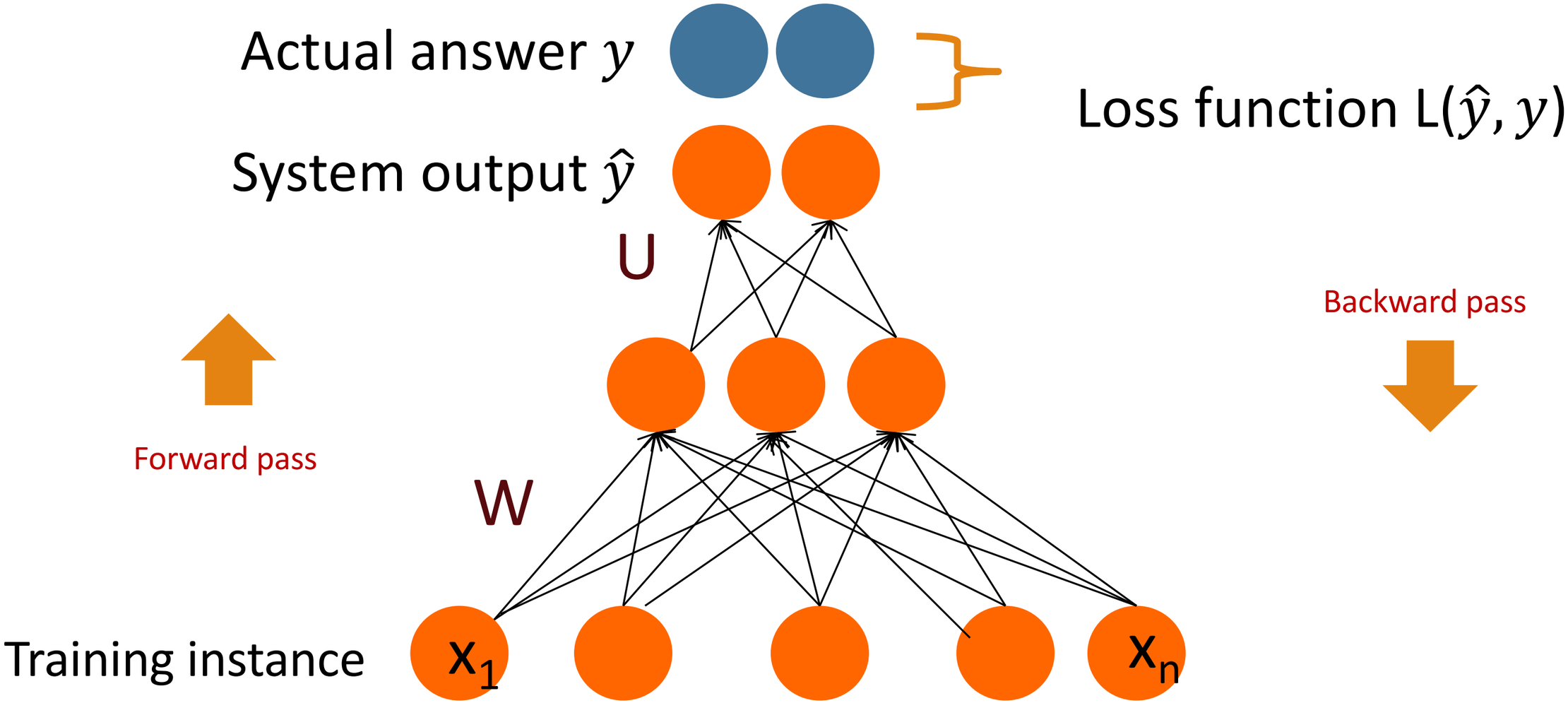
Using slides by:
Jure Leskovec & Mina Ghashami, Dan Jurafsky & James H. Martin,
Roger Grosse, Stephen Scott, Geoffrey Hinton

Simple Neural Networks and Neural Language Models

Training Neural Nets: Overview

Slides by Dan Jurafsky and James H. Martin

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

Reminder: gradient descent for weight updates

Use the derivative of the loss function with respect to weights $\frac{d}{dw} L(f(x;w), y)$

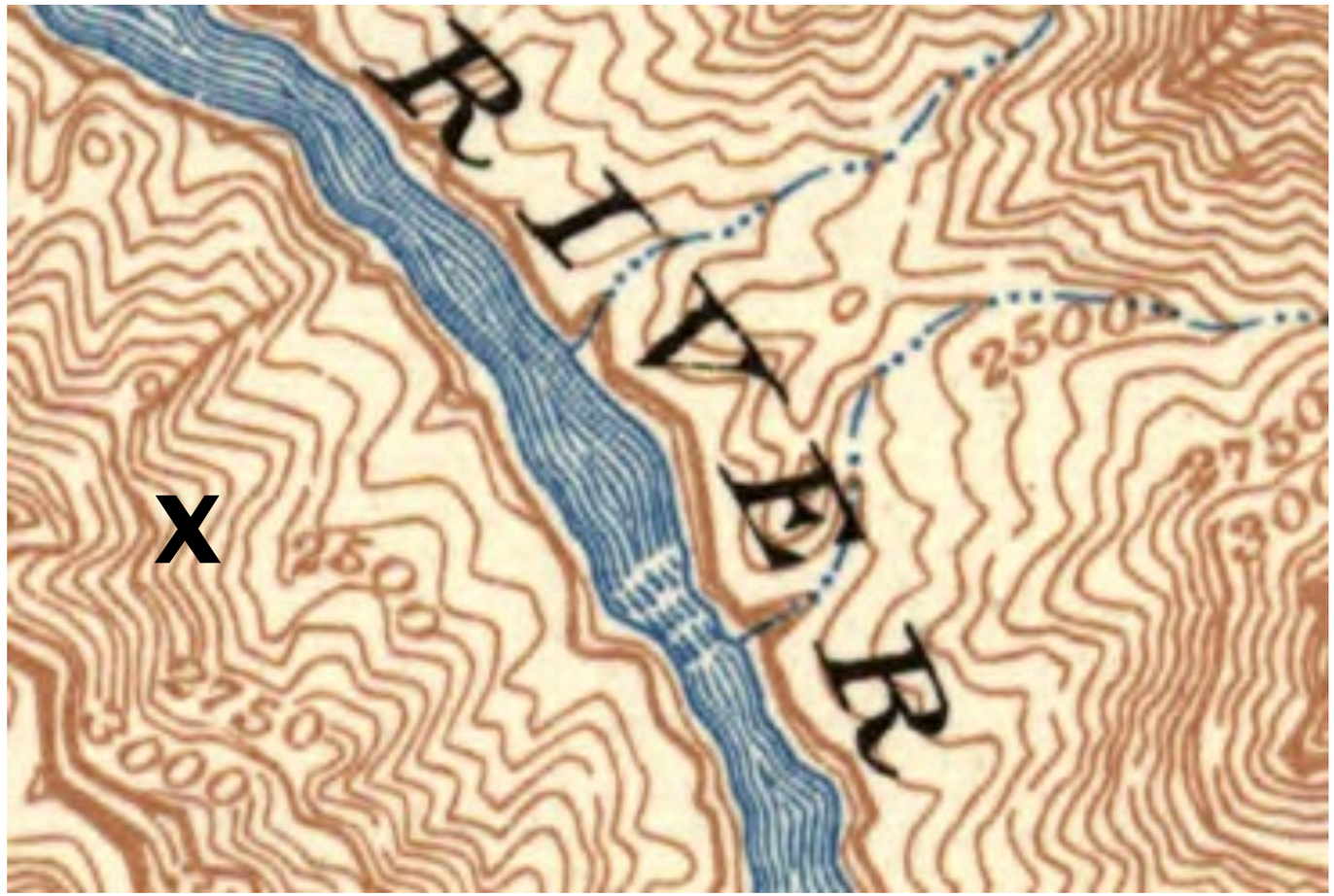
To tell us how to adjust weights for each training item

- Move them in the opposite direction of the gradient

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

Intuition of gradient descent

How do I get to the bottom of this river canyon?



Look around me 360°

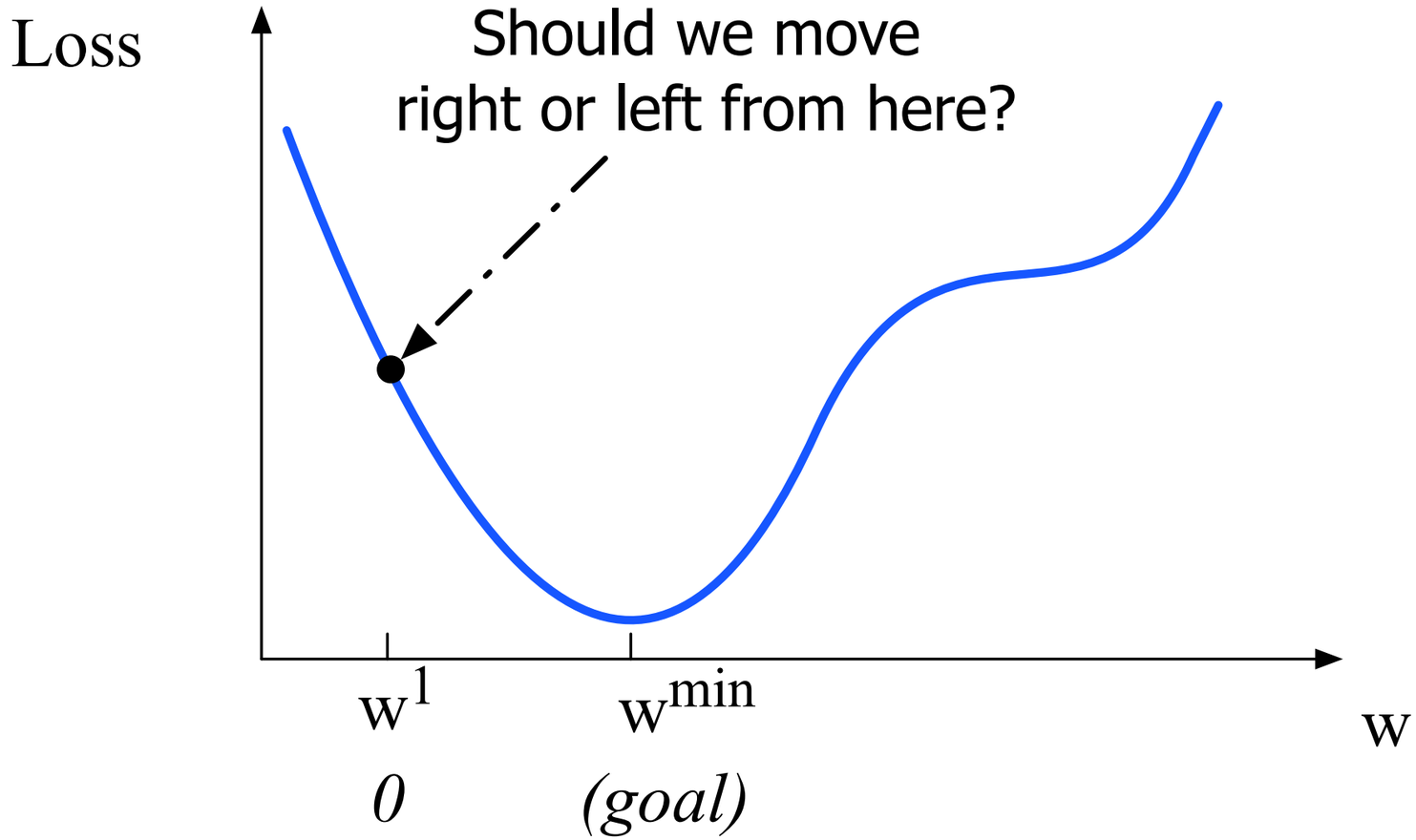
Find the direction of steepest slope down

Go that way

Let's first visualize for a single scalar w

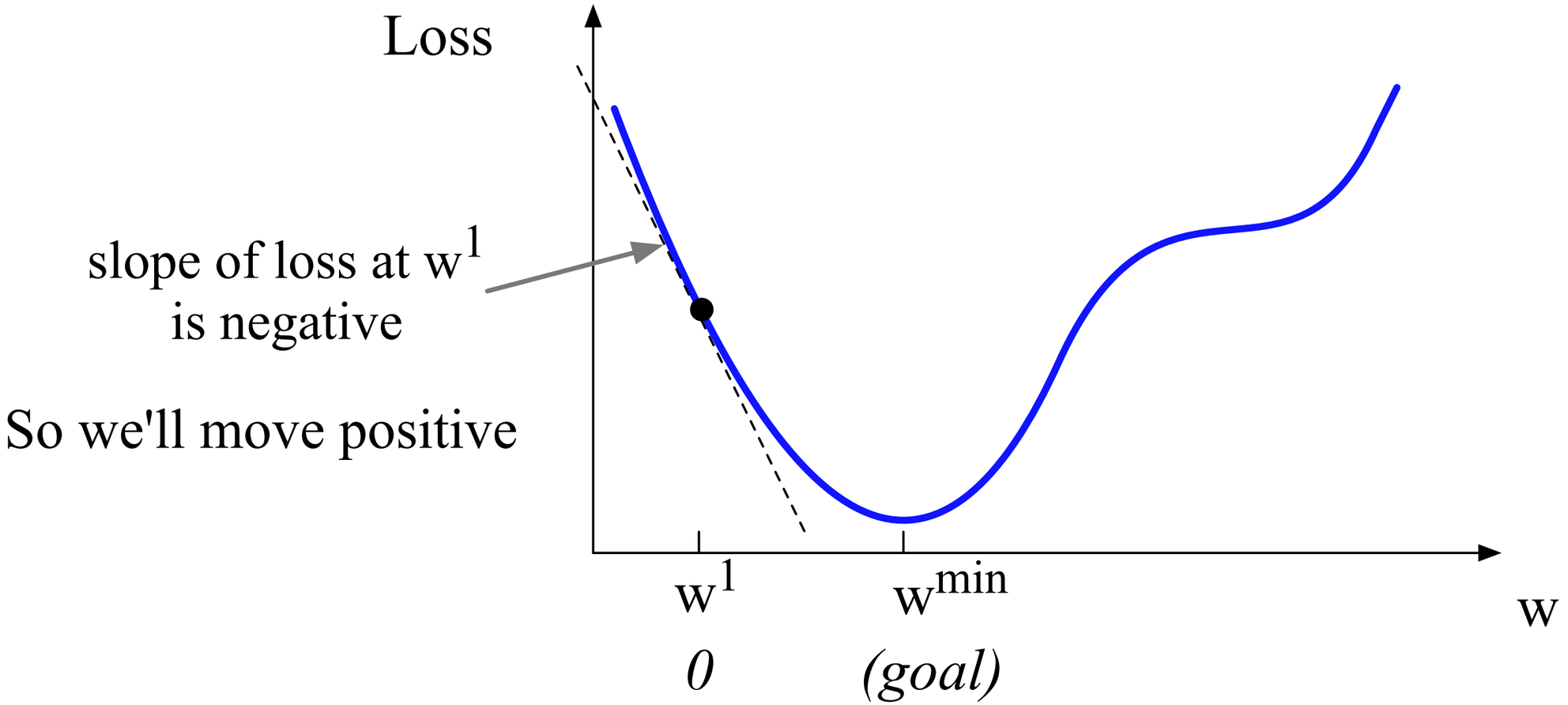
Q: Given current w, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function



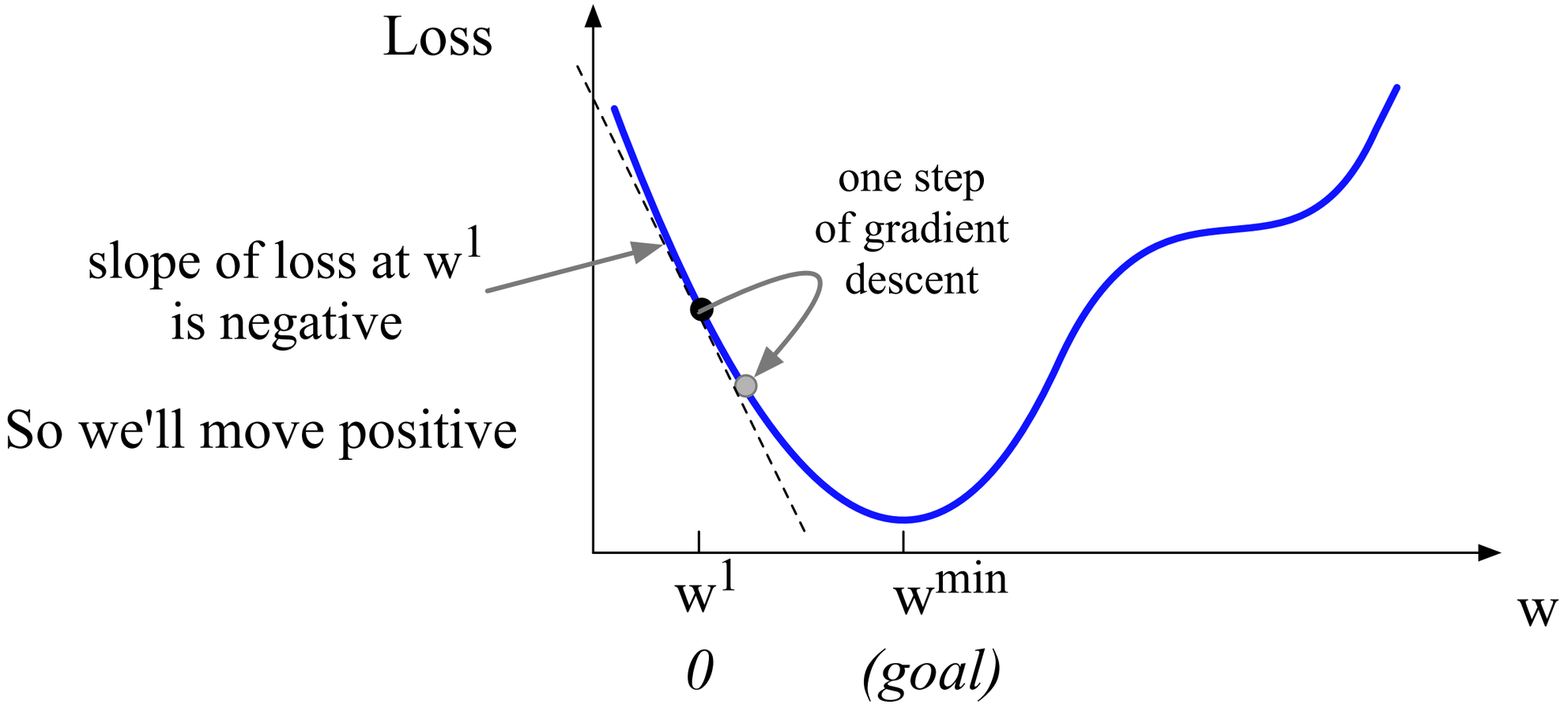
Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?
A: Move *w* in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?
A: Move *w* in the reverse direction from the slope of the function



Gradients

The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

Gradient Descent: Find the gradient of the loss function at the current point and move in the **opposite** direction.

How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x;w), y)$ weighted by a **learning rate** η
- Higher learning rate means move *w* faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

Real gradients

Are much longer; lots and lots of weights

For each dimension w_i the gradient component i tells us the slope with respect to that variable.

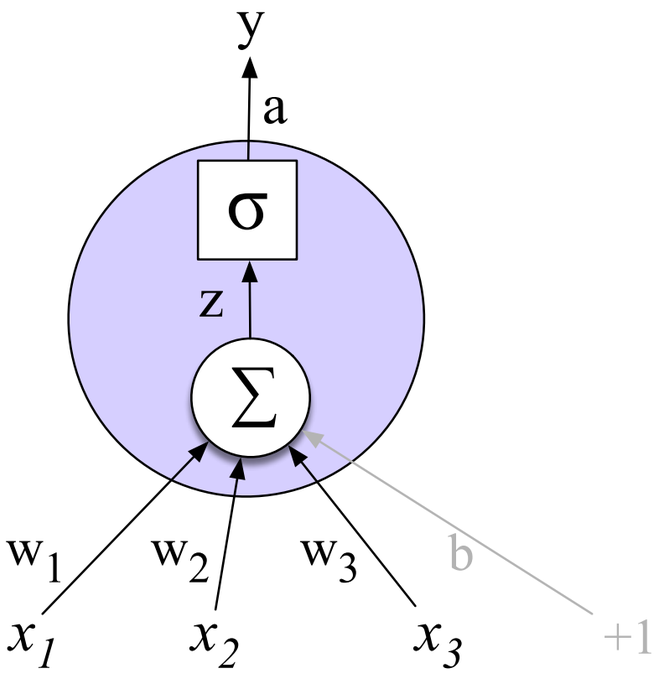
- "How much would a small change in w_i influence the total loss function L?"
- We express the slope as a partial derivative ∂ of the loss ∂w_i

The gradient is then defined as a vector of these partials.

Where did that derivative come from?

Using the chain rule! $f(x) = u(v(x))$ $\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$

Intuition (see the text for details)



Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}$$

How can I find that gradient for every weight in the network?

These derivatives on the prior slide only give the updates for one weight layer: the last one!

What about deeper networks?

- Lots of layers, different activation functions?

Solution:

- Even more use of the chain rule!!
- Computation graphs and backward differentiation!

Agenda

- Motivation: why embeddings?
- Neural network fundamentals
- Training Neural Networks
- **Computation Graphs and Backward Differentiation**
- Why is this scalable?
- The XOR problem
- Learning embeddings: Autoencoders

Using slides by:
Jure Leskovec & Mina Ghashami, Dan Jurafsky & James H. Martin,
Roger Grosse, Stephen Scott, Geoffrey Hinton

Simple Neural Networks and Neural Language Models

Computation Graphs and Backward Differentiation

Slides by Dan Jurafsky and James H. Martin

Why Computation Graphs

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

- But the 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

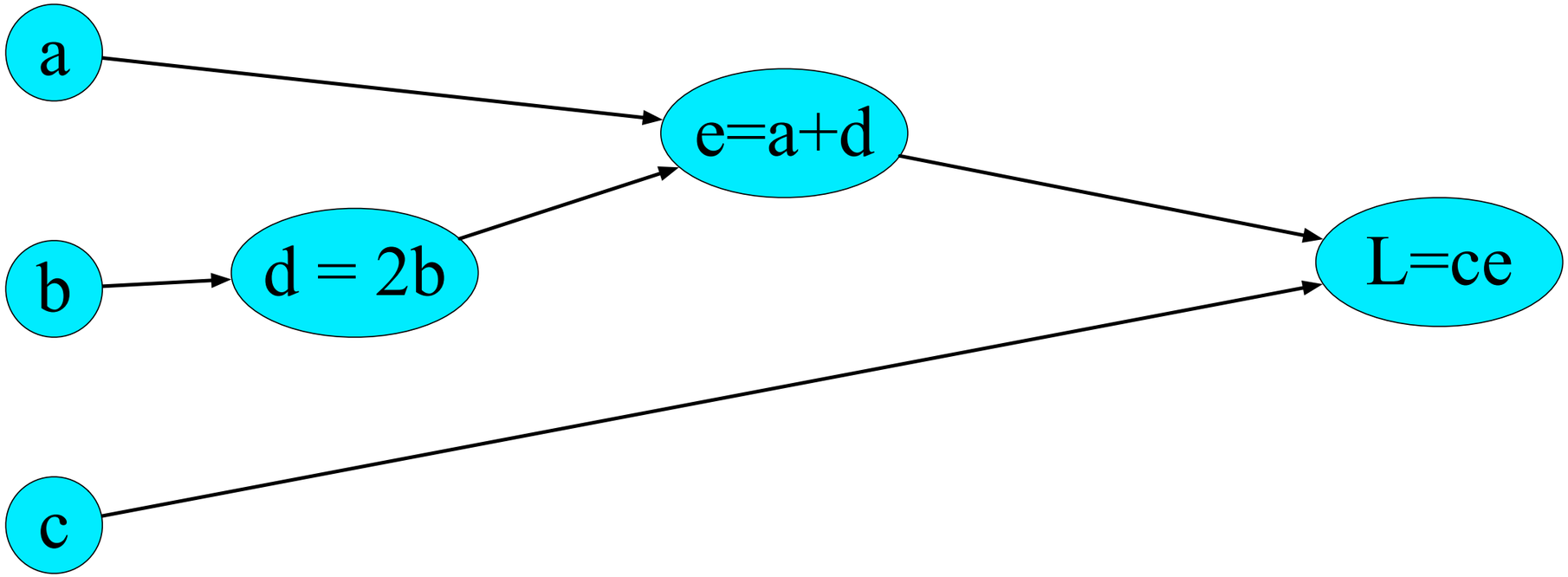
A computation graph represents the process of computing a mathematical expression

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

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

Computations:

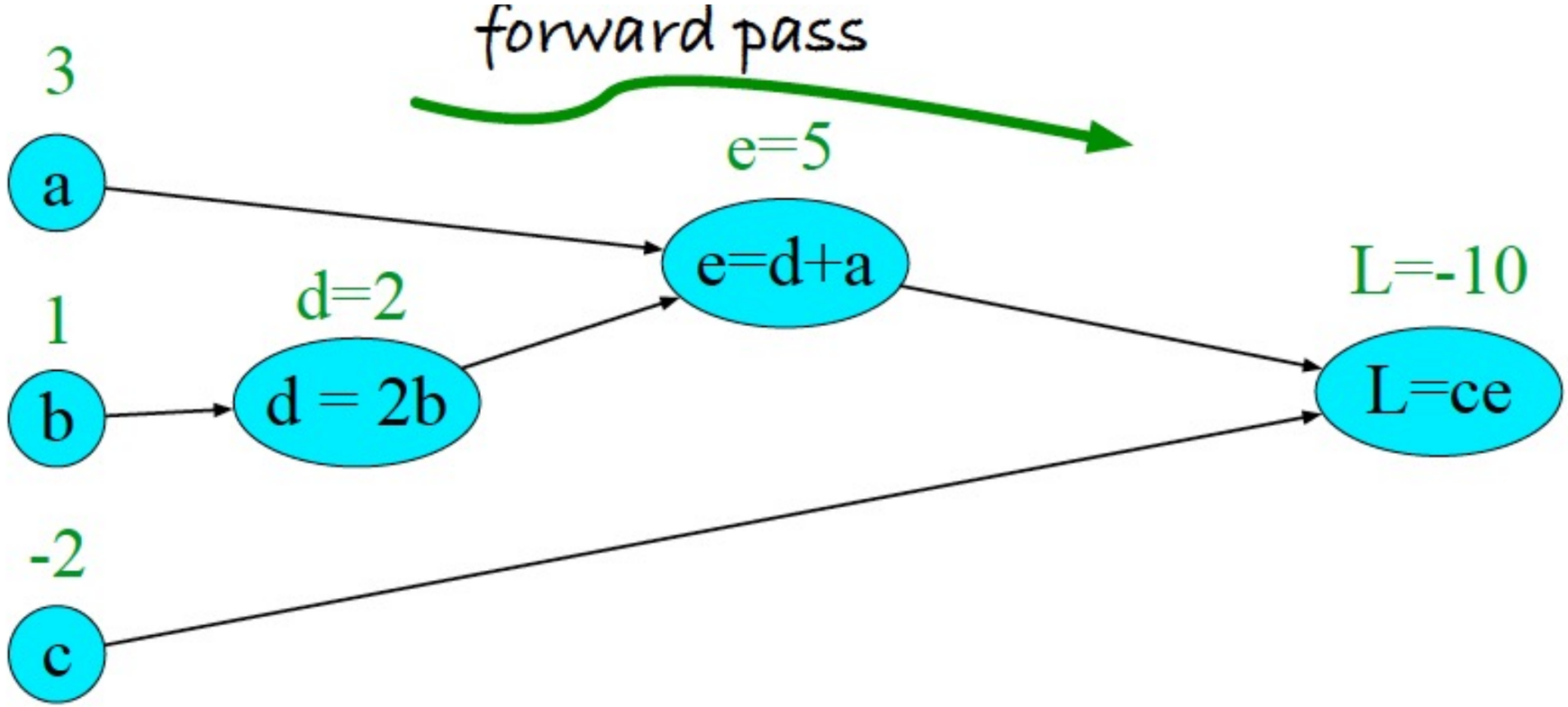
$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



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

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.

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

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

We want: $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$

The derivative $\frac{\partial L}{\partial a}$ tells us how much a small change in *a* affects *L*.

The chain rule

Computing the derivative of a composite function:

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

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

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

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial c} &= e \\
\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}
\end{aligned}
$$

Example

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

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

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

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

$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a+d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$

Example

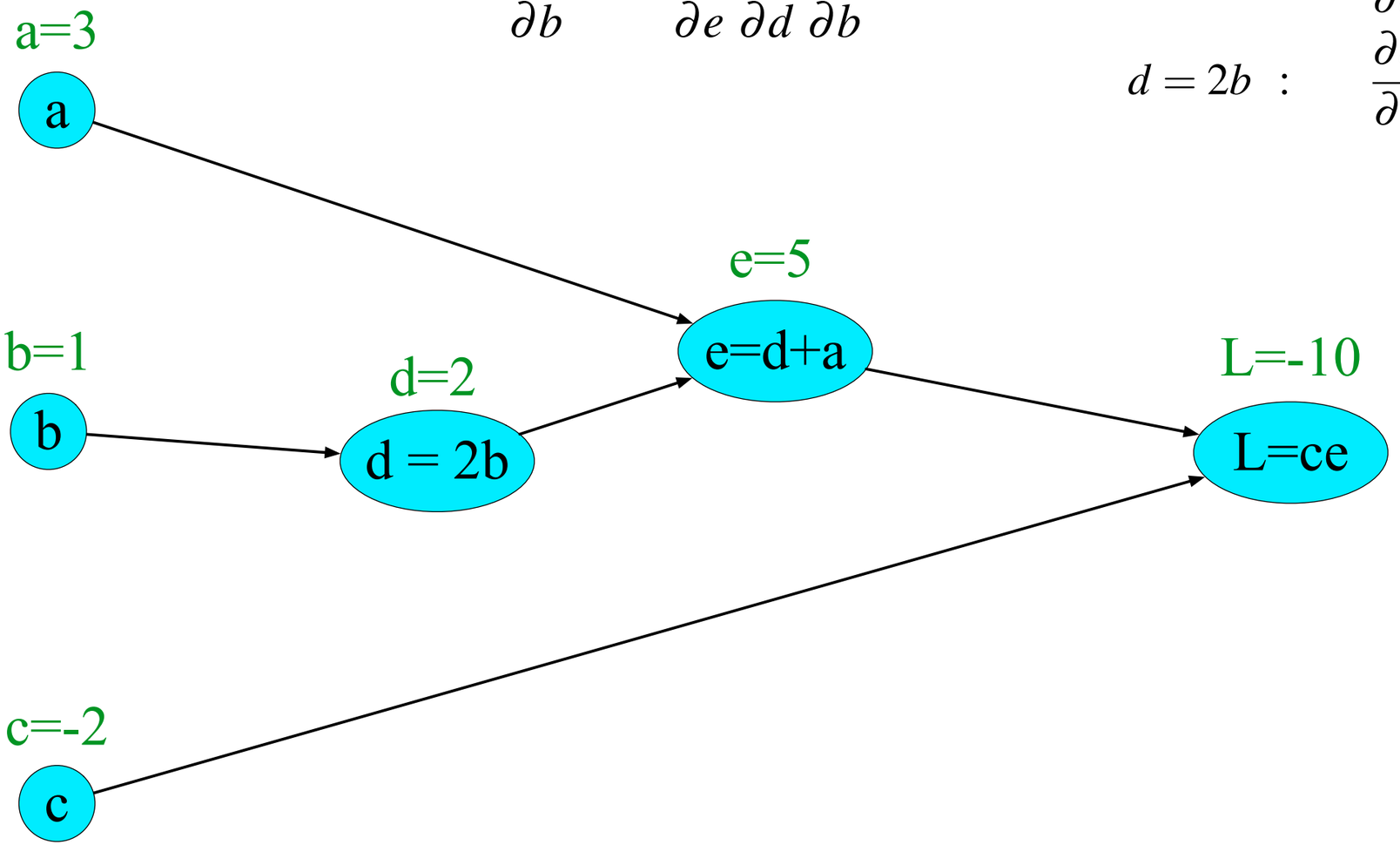
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

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

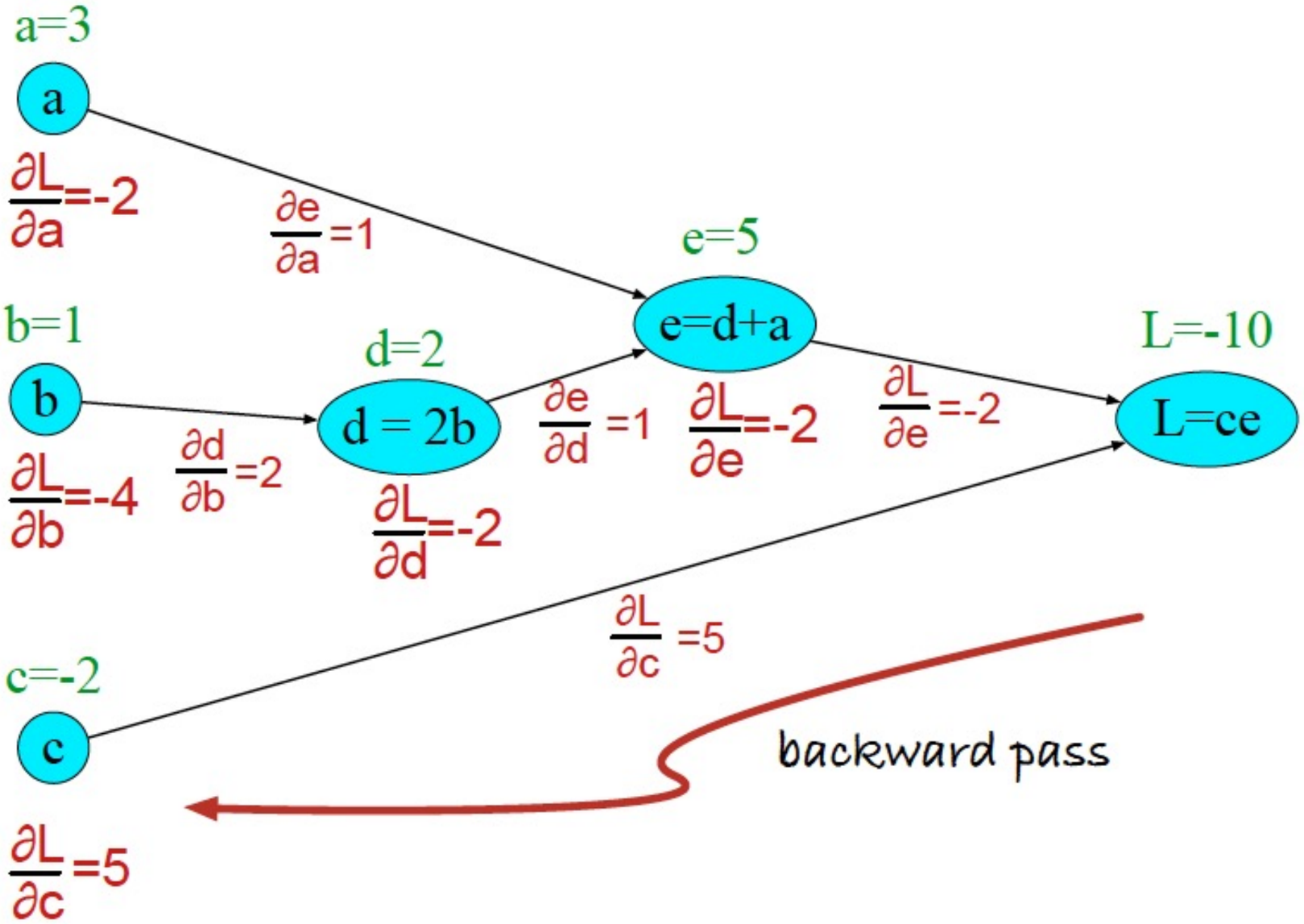
$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a + d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

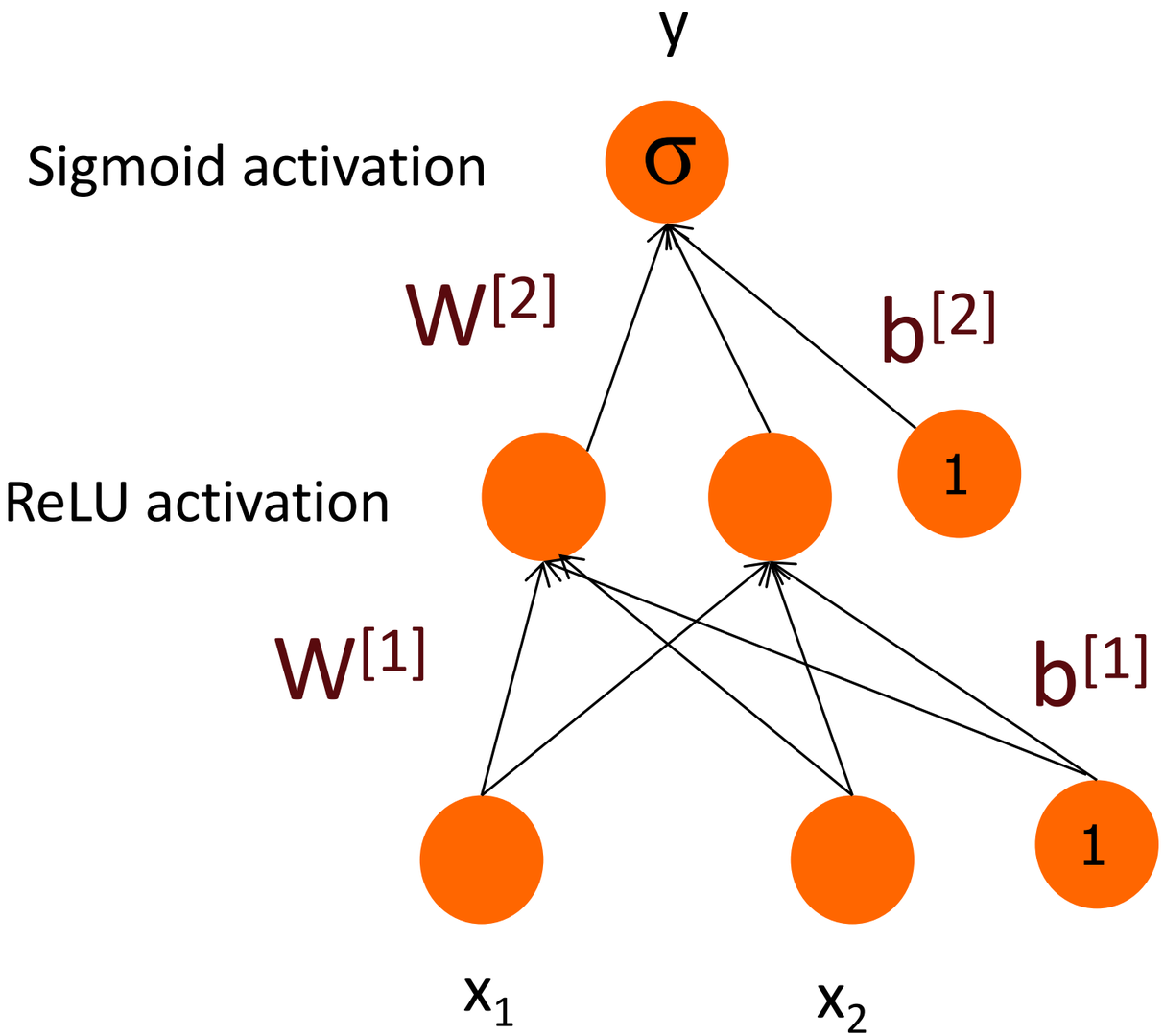
$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$



Example

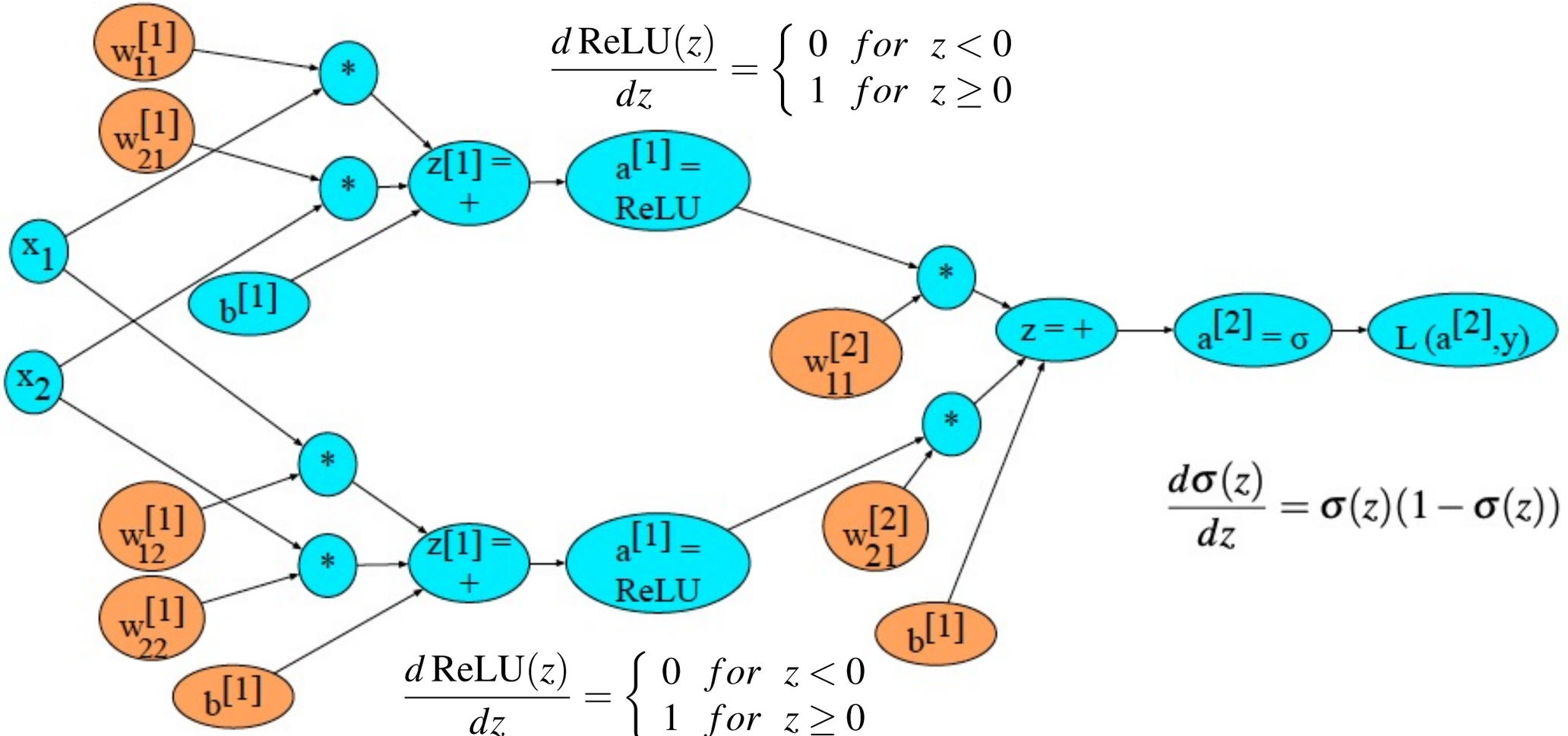

a=3
a
∂L/∂a=-2
∂e/∂a=1
e=5
e=d+a
b=1
b
∂L/∂b=-4
∂d/∂b=2
d=2
d = 2b
∂e/∂d=1
∂L/∂d=-2
∂L/∂e=-2
∂L/∂e=-2
L=-10
L=ce
∂L/∂c=5
c=-2
c
backward pass
∂L/∂c=5

Backward differentiation on a two layer network



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathbf{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \boldsymbol{\sigma}(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

Backward differentiation on a 2-layer network



Neural Network

- There are much more about NN including:
  - Minibatch Stochastic gradient descent
  - Batch size, Epoch
  - Learning rate scheduling
  - Optimizers to improve over SGD

Agenda

- Motivation: why embeddings?
- Neural network fundamentals
- Training Neural Networks
- Computation Graphs and Backward Differentiation
- **Why is this scalable?**
- The XOR problem
- Learning embeddings: Autoencoders

Using slides by:
Jure Leskovec & Mina Ghashami, Dan Jurafsky & James H. Martin,
Roger Grosse, Stephen Scott, Geoffrey Hinton