University of Colorado **Boulder**

Department of Computer Science

CSCI 5622: Machine Learning

Chenhao Tan

Lecture 11: Neural Networks I

Slides adapted from Chris Ketelsen, Jordan Boyd-Graber, and Noah Smith

Administrivia

- Final project team formation
- Proposal ideas
- HW3 released

Learning Objectives

- Understanding feed-forward neural networks

Outline

- Revisiting logistic regression
- Feed-forward neural networks

Outline

- Revisiting logistic regression
- Feed-forward neural networks

Revisiting Logistic Regression

$$P(\mathrm{Y}=0 \mid \boldsymbol{x}, \beta) = \frac{1}{1+\exp\left[\beta_0 + \sum_j \beta_j \boldsymbol{x}_j\right]}$$

$$P(\mathrm{Y}=1 \mid \boldsymbol{x}, \beta) = \frac{\exp\left[\beta_0 + \sum_j \beta_j \boldsymbol{x}_j\right]}{1+\exp\left[\beta_0 + \sum_j \beta_j \boldsymbol{x}_j\right]}$$

$$\mathscr{L} = -\sum_i \log P(y^{(i)} \mid \boldsymbol{x}^{(i)}, \beta)$$

$$\frac{\exp(0)}{\exp(0) + \exp(\beta_0 + \sum_j \beta_j x_j)}$$

$0 = \beta^{(0)} \cdot x$

$\beta \cdot x$

$$\begin{pmatrix} \beta^{(0)} \\ \beta \end{pmatrix} x = \begin{pmatrix} 0 \\ \beta_0 + \sum_j \beta_j x_j \end{pmatrix}$$

softmax — $\dfrac{\exp 0}{\exp(0) + \exp(\beta_0 + \sum_j \beta_j x_j)}$, $\dfrac{\exp(\beta_0 + \sum_j \beta_j x_j)}{\exp(0) + \exp(\beta_0 + \sum_j \beta_j x_j)}$

Next slide $\begin{cases} \beta_1 = [0, \ldots, 0] \\ \beta_2 = [\beta_0, \ldots, \beta_d] \end{cases}$

$$x \xrightarrow{\text{linear}} \begin{bmatrix} 0 \\ \beta_0 + \sum_j \beta_j x_j \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} P(y=0 \mid x) \\ P(y=1 \mid x) \end{bmatrix}$$

Revisiting Logistic Regression

- Transformation on $\boldsymbol{x}$ (we map class labels from $\{0, 1\}$ to $\{1, 2\}$):

$$l_k = \beta_k^T \boldsymbol{x}, k = 1, 2$$

$$o_k = \frac{\exp l_k}{\sum_{c \in \{1,2\}} \exp l_c}, k = 1, 2$$

Revisiting Logistic Regression

- Transformation on $\boldsymbol{x}$ (we map class labels from $\{0, 1\}$ to $\{1, 2\}$):

$$l_k = \beta_k^T \boldsymbol{x}, k = 1, 2 \quad \text{linear layer}$$

$$o_k = \frac{\exp l_k}{\sum_{c \in \{1,2\}} \exp l_c}, k = 1, 2 \quad \text{softmax layer}$$

Revisiting Logistic Regression

- Transformation on $\boldsymbol{x}$ (we map class labels from $\{0,1\}$ to $\{1,2\}$):

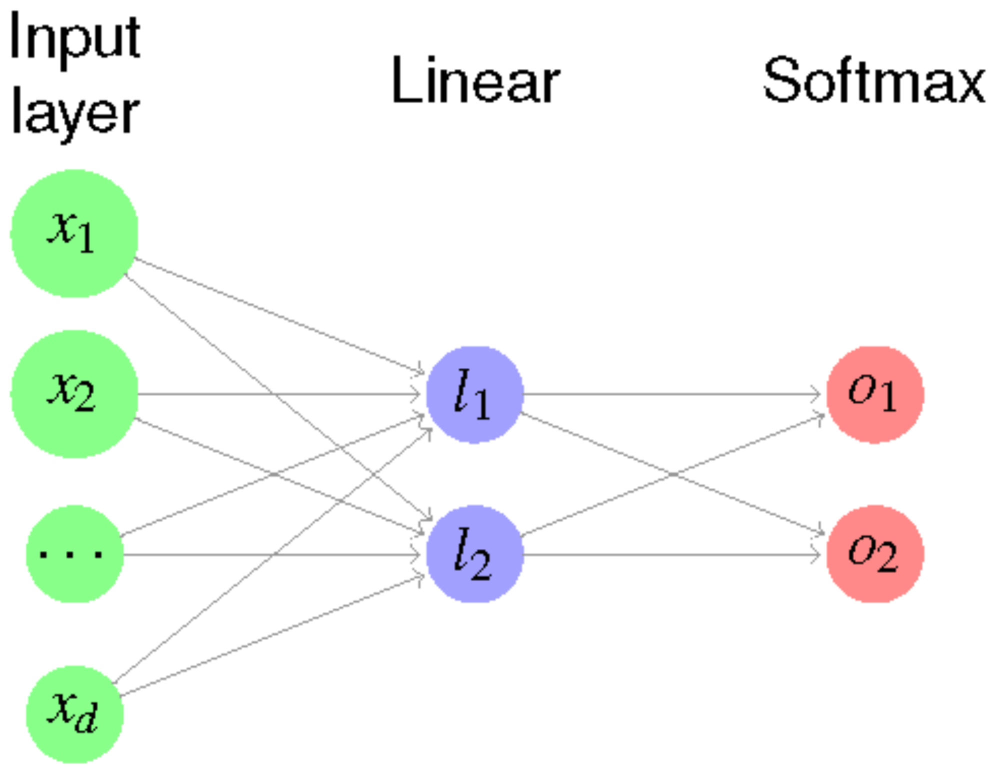
$$l_k = \beta_k^T \boldsymbol{x}, k = 1, 2 \quad \text{linear layer}$$

$$o_k = \frac{\exp l_k}{\sum_{c \in \{1,2\}} \exp l_c}, k = 1, 2 \quad \text{softmax layer}$$

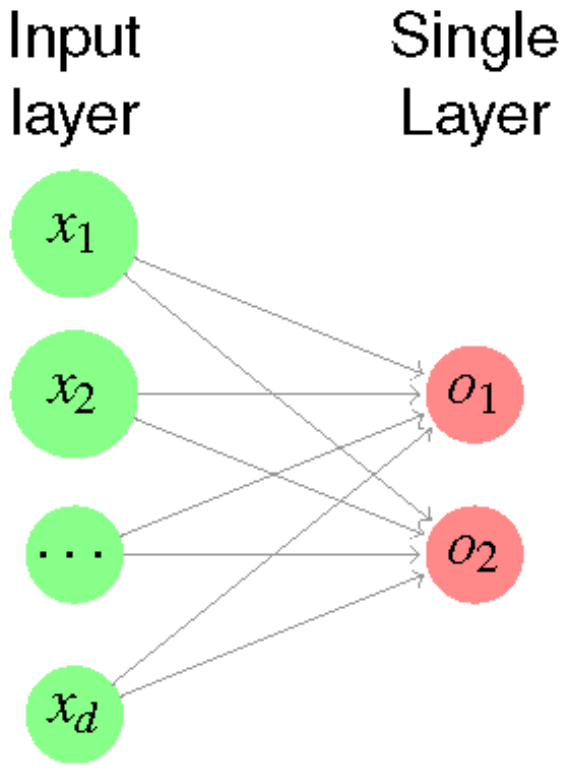
- Objective function (using cross entropy $-\sum_i p_i \log q_i$):

$$\mathscr{L}(Y, \hat{Y}) = -\sum_i \left[P(y^{(i)} = 1) \log P(\hat{y}_i = 1 \mid \boldsymbol{x}^{(i)}, \beta) + P(y^{(i)} = 0) \log \hat{P}(y_i = 0 \mid \boldsymbol{x}^{(i)}, \beta) \right]$$

Logistic Regression as a Single-layer Neural Network



Logistic Regression as a Single-layer Neural Network



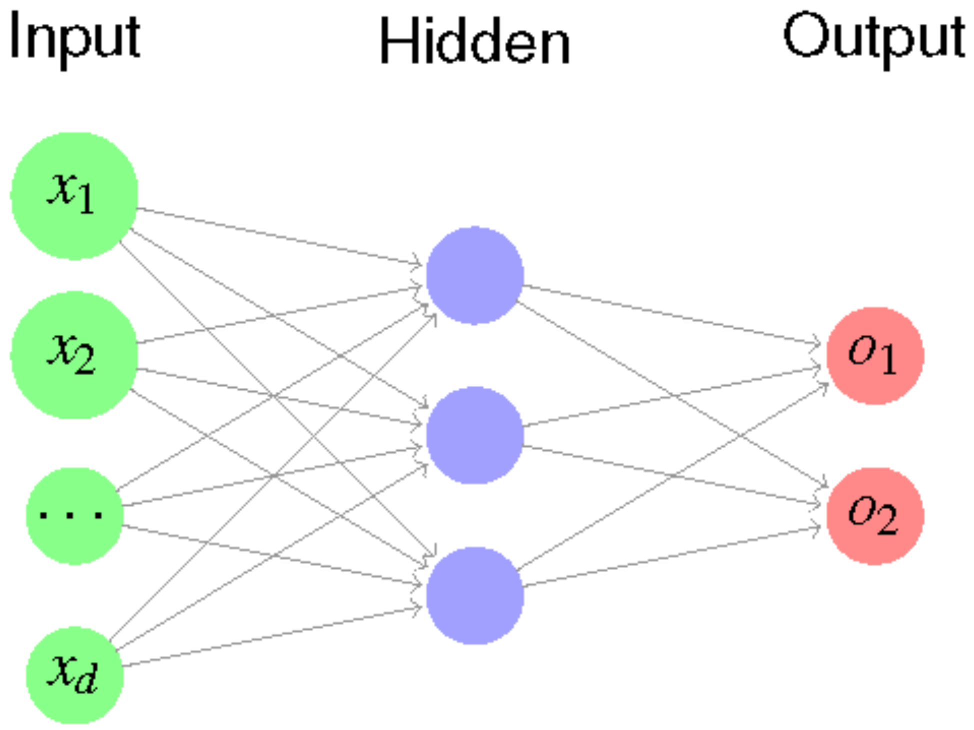
k

$$\frac{\exp(x_i)}{\sum_i \exp(x_i)}$$

Outline

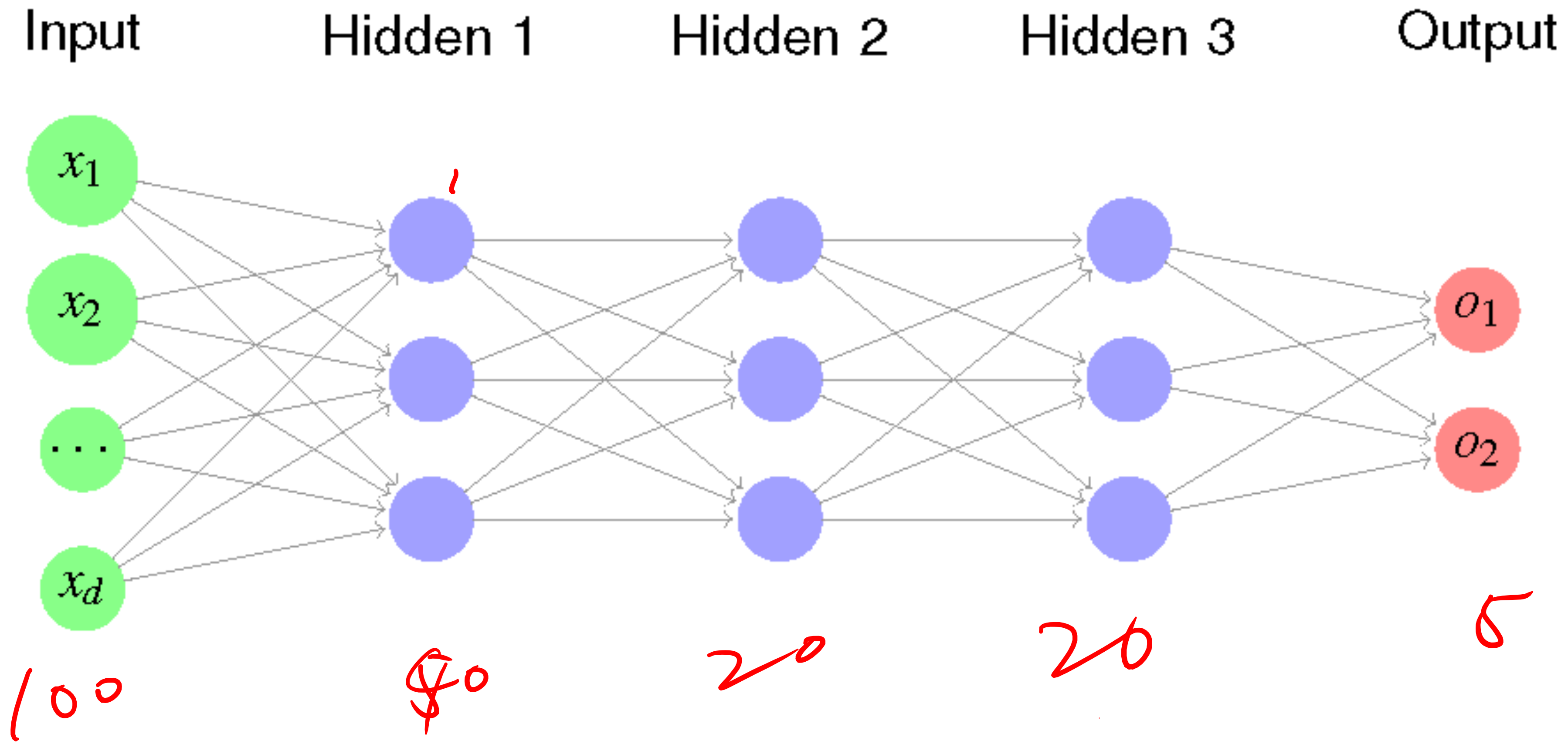
- Revisiting logistic regression
- Feed-forward neural networks

A two-layer example (one hidden layer)



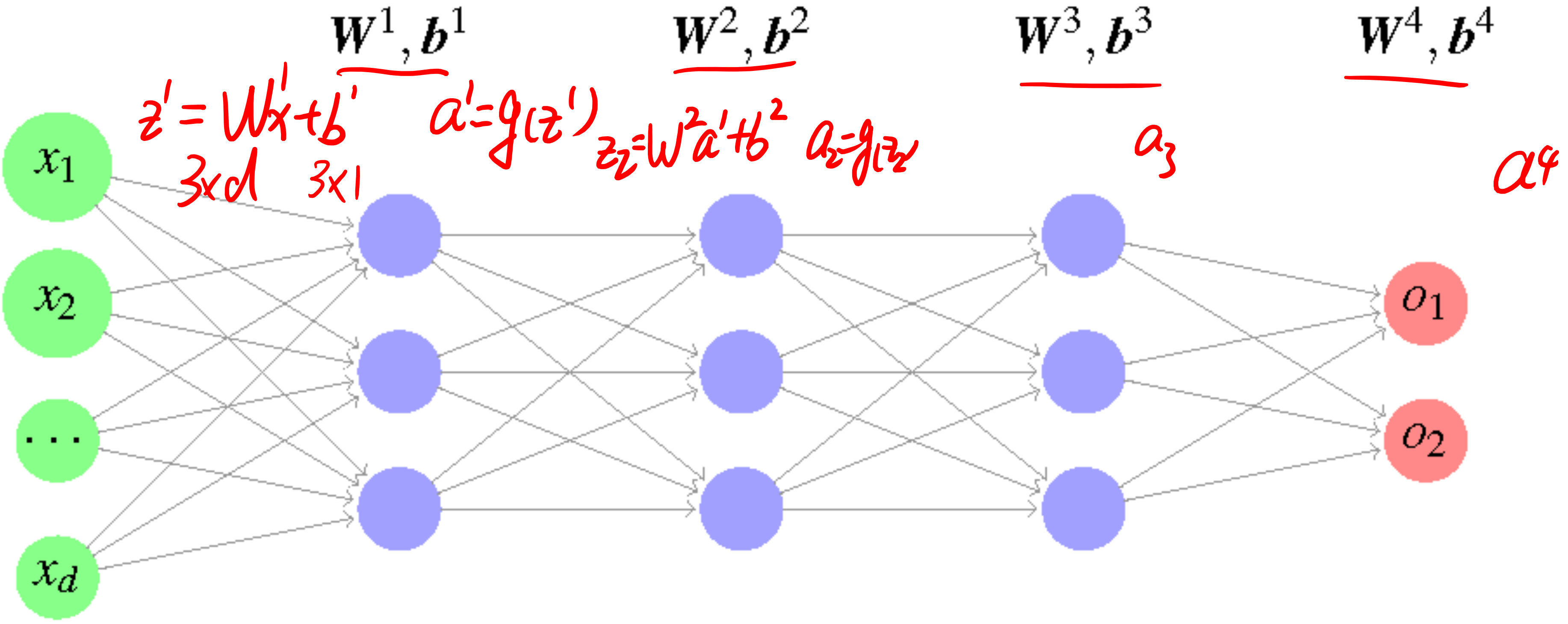
Deep Neural networks

More layers:



Forward propagation algorithm

How do we make predictions based on a multi-layer neural network?
Store the biases for layer l in $\boldsymbol{b}^l$, weight matrix in $\boldsymbol{W}^l$



Forward propagation algorithm

Suppose your network has L layers
Make prediction for an instance $\boldsymbol{x}$

```
Initialize a^0 = x
for l = 1 to L do
    z^l = W^l a^{l-1} + b^l
    a^l = g(z^l)
end for
The prediction ŷ is simply a^L
```

Nonlinearity

What happens if there is no nonlinearity?

$$W^2(W^1x+b^1)+b^2$$

$$= \underbrace{W^2W^1}_{W}x+\underbrace{W^2b^1+b^2}_{b}$$

What happens if there is no nonlinearity?
Linear combinations of linear combinations are still linear combinations.

Neural networks in a nutshell

- Training data $S_{\text{train}} = \{(\boldsymbol{x}, y)\}$
- Network architecture (model)

$$\hat{y} = f_w(\boldsymbol{x})$$

g

- Loss function (objective function)

$$\mathscr{L}(y, \hat{y})$$

a<

- Learning (next week)

Nonlinearity Options

- Sigmoid

$$f(x) = \frac{1}{1 + \exp(x)}$$

- tanh

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- ReLU (rectified linear unit)

$$f(x) = \max(0, x)$$

- softmax

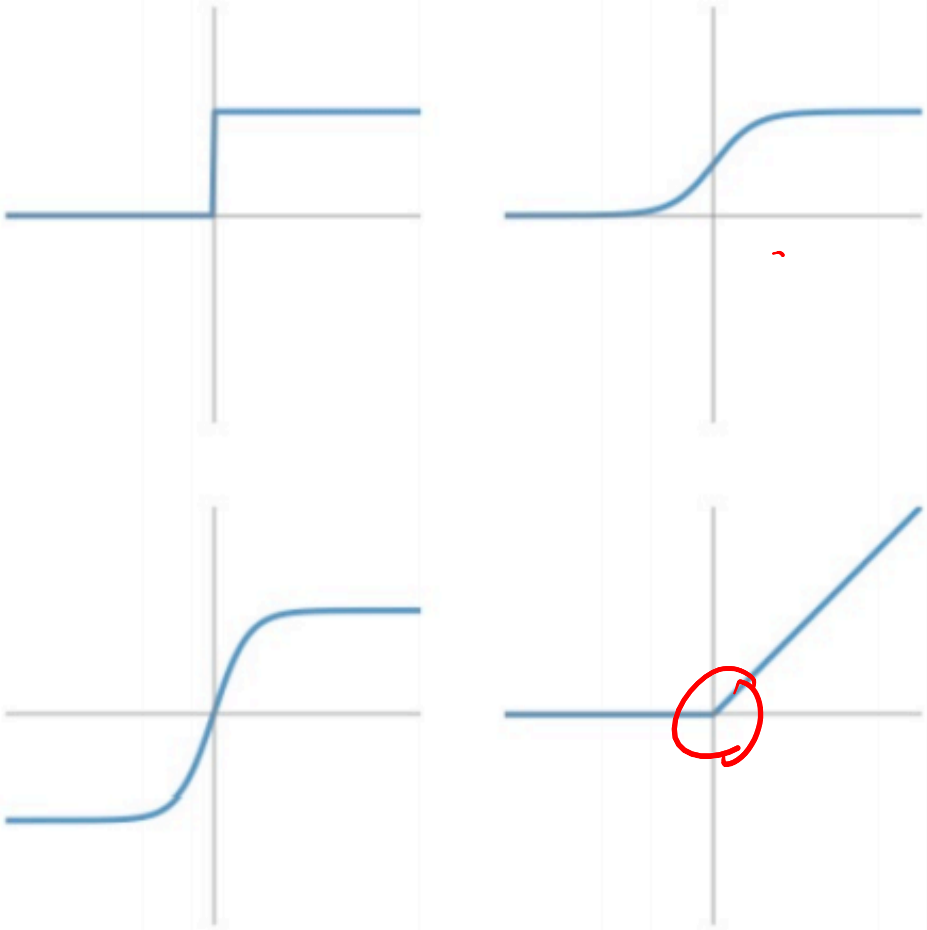
$$\boldsymbol{x} = \frac{\exp(\boldsymbol{x})}{\sum_{x_i} \exp(x_i)}$$

https://keras.io/activations/

Nonlinearity Options

$I(a>0)$

Sigmoid

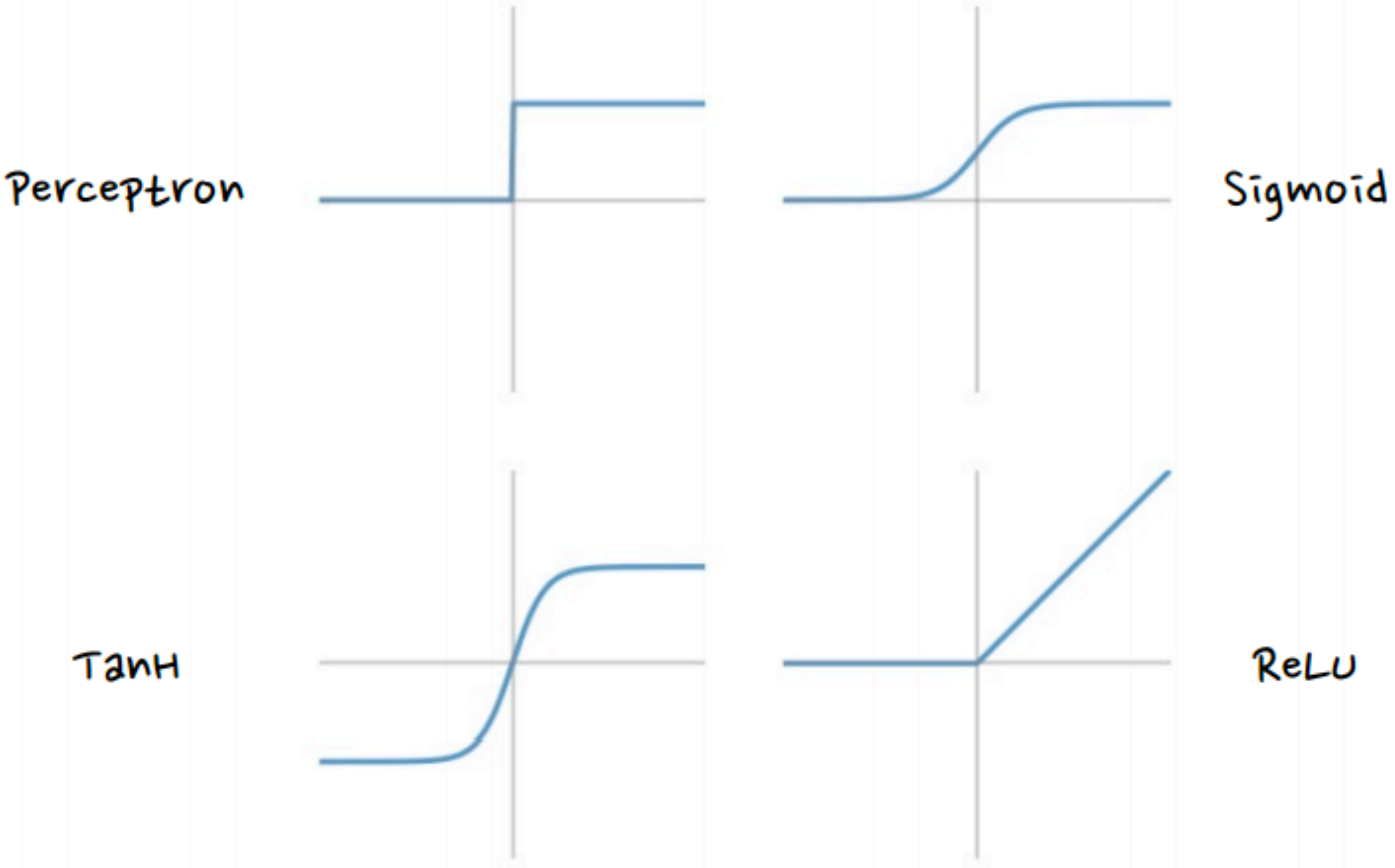


tanh

ReLU

$\max(0,x)$

Nonlinearity Options



Loss Function Options

- ℓ_2 loss

$$\sum_i (y_i - \hat{y}_i)^2$$

- ℓ_1 loss

$$\sum_i |y_i - \hat{y}_i|$$

- Cross entropy (logistic regression)

$$-\sum_i y_i \log \hat{y}_i$$

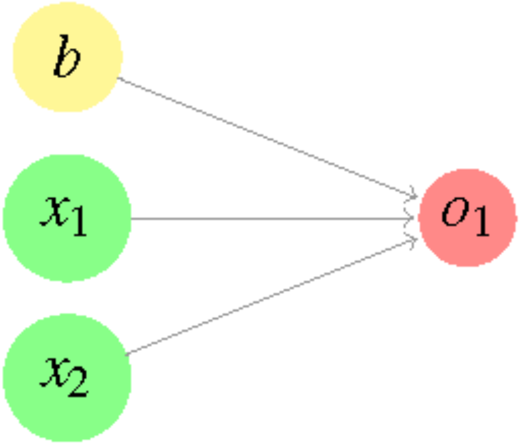
- Hinge loss (more on this during SVM)

$$\max(0, 1 - y\hat{y})$$

`https://keras.io/losses/`

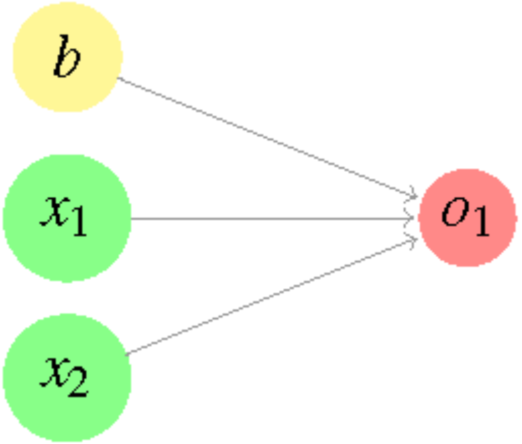
A Perceptron Example

$$\boldsymbol{x} = (x_1, x_2), y = f(x_1, x_2)$$



A Perceptron Example

$\boldsymbol{x} = (x_1, x_2), y = f(x_1, x_2)$



We consider a simple activation function

$$f(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

A Perceptron Example

$W_{1\times 2}$ b_1

Simple Example: Can we learn OR?

	0	1	1	2
x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \vee x_2$	0	1	1	1

W b

$$\hat{y} = \begin{cases} W\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b \geq 0 & \text{if } (x_1 \text{ OR } x_2) = 1 \\ < 0 & \text{if } (x_1 \text{ OR } x_2) = 0 \end{cases}$$

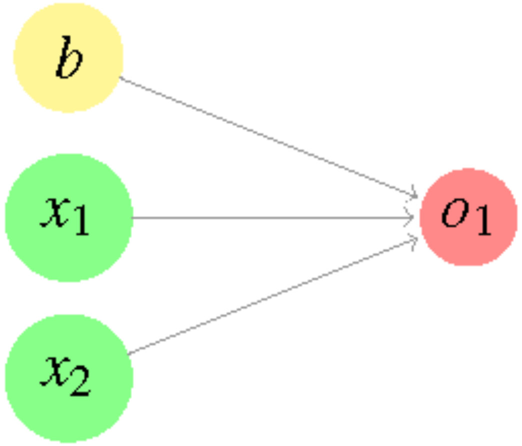
$$\begin{array}{cc} w_1 & w_2 \\ [1, & 1] \end{array} \quad -\frac{1}{2}$$

A Perceptron Example

Simple Example: Can we learn OR?

x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \vee x_2$	0	1	1	1

$$w = (1, 1), b = -0.5$$



A Perceptron Example

Simple Example: Can we learn AND?

0 1 1 | 2

x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \wedge x_2$	0	0	0	1

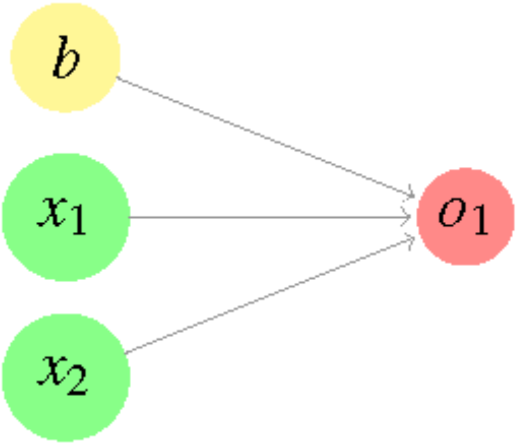
(1, 1)   -1,5

A Perceptron Example

Simple Example: Can we learn AND?

x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \wedge x_2$	0	0	0	1

$$w = (1, 1), b = -1.5$$



A Perceptron Example

Simple Example: Can we learn NAND?

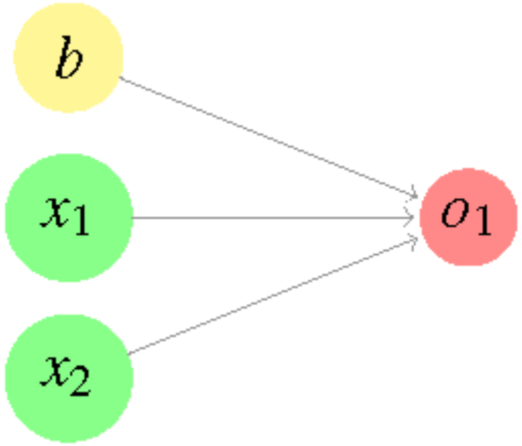
x_1	0	1	0	1
x_2	0	0	1	1
$y = \neg(x_1 \wedge x_2)$	1	1	1	0

A Perceptron Example

Simple Example: Can we learn NAND?

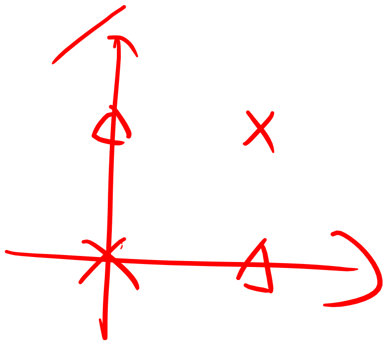
x_1	0	1	0	1
x_2	0	0	1	1
$y = \neg(x_1 \wedge x_2)$	1	1	1	0

$$\boldsymbol{w} = (-1, -1), b = 1.5$$



Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0



Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!

A Perceptron Example

Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!
But why?

A Perceptron Example

Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!

But why?

The single-layer perceptron is just a linear classifier, and can only learn things that are linearly separable.

A Perceptron Example

Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!

But why?

The single-layer perceptron is just a linear classifier, and can only learn things that are linearly separable.

How can we fix this?

A Perceptron Example

Increase the number of layers.

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

A Perceptron Example

Increase the number of layers.

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

$$\mathrm{XOR} = \mathrm{AND}\,(\underline{\mathrm{OR}\;,\;\mathrm{NAND}})$$

2x2

$W^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$ OR NAND

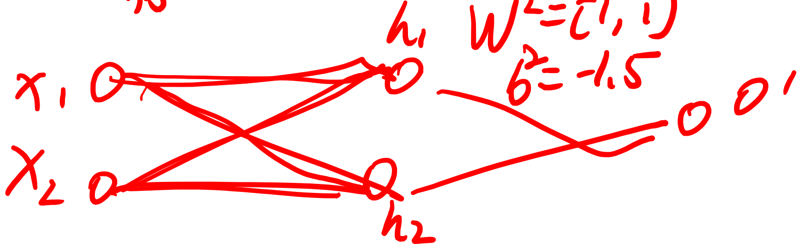
$b^1 = \begin{bmatrix} -0.5 \\ 1.5 \end{bmatrix}$

2

OR(x_1, x_2)

$W^2 = [1, 1)$

$b^2 = -1.5$



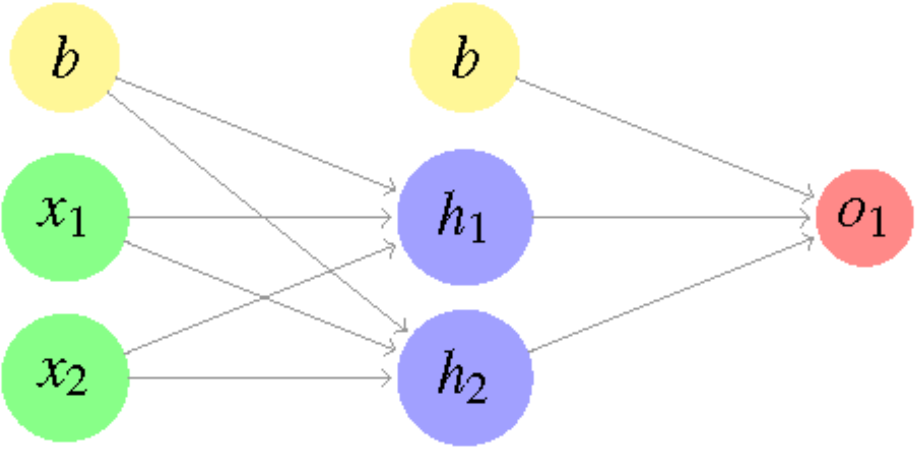
NAND(x_1, x_2)

A Perceptron Example

Increase the number of layers.

Non-linear activations are step functions.

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0



$$W^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, b^1 = \begin{bmatrix} -0.5 \\ 1.5 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b^2 = -1.5$$

General Expressiveness of Neural Networks

Neural networks with a single hidden layer can approximate any measurable functions [Hornik et al., 1989, Cybenko, 1989].

Recap

- Logistic regression and perceptron can be seen as special cases of neural networks
- Feed-forward algorithm (forward propagation)