

# Machine learning advanced Methods in AI research

Dong Nguyen  
24 Sept 2019



Utrecht University

# Practicalities

## **Literature for today:**

- Jurafsky & Martin: Chapter 5 (Logistic Regression, skip 5.8)
- Jurafsky & Martin, Chapter 7 (Neural Networks and Neural Language Models, skip 7.5)

# So far

- **ML concepts:**
  - Supervised learning
  - Inductive bias
  - Overfitting and underfitting
  - Decision boundaries
  - Evaluation of supervised learning systems
  - Vectors
  - Distance measures
- **Methods**
  - Decision trees
  - Nearest-neighbors

## Today:

Logistic regression  
Neural networks (basics)

# Logistic regression

# Why?

- It's very often used (also in the social sciences)
- It's a very strong baseline
- Fundamental to understanding neural networks

But let's start with linear  
*regression* first

## RECAP !

# Supervised learning

Learn a machine learning model using **labeled example instances**:

features                  target  
    ↘                  ↘  
 $\{<\mathbf{x}^{(1)}, \mathbf{y}^{(1)}>, \dots, <\mathbf{x}^{(N)}, \mathbf{y}^{(N)}>\}$

**Goal:** Predict the target using the features

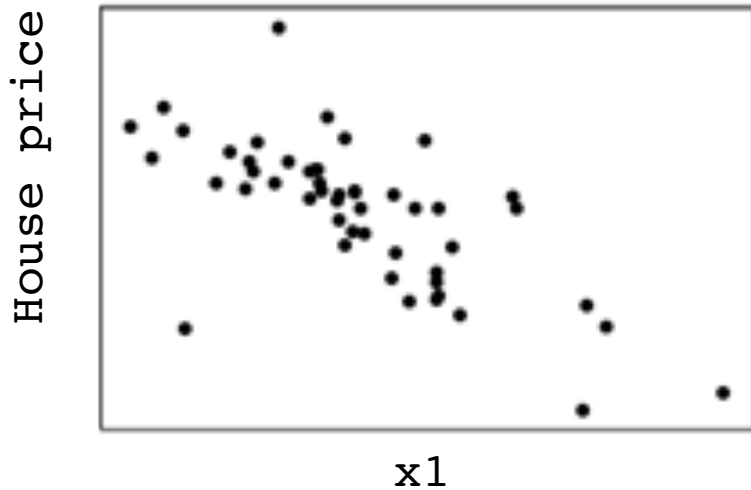
Need to define **features**, characteristics of the instances that the model uses for predictions (words in a document, movie ratings, etc..)

## Features for house price prediction:

- Neighborhood
- Number of bedrooms
- First floor square meters
- Number of schools within 2 km
- Police Label Safe Housing
- ..

This is a **regression** problem:  
predict continuous output

# Regression



features

target

$$\{ \langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \dots, \langle \mathbf{x}^{(N)}, \mathbf{y}^{(N)} \rangle \}$$

**Goal:** Predict the target using the features

**Regression task:**

Output is a continuous value ( $\mathbf{y} \in \mathbb{R}$ )

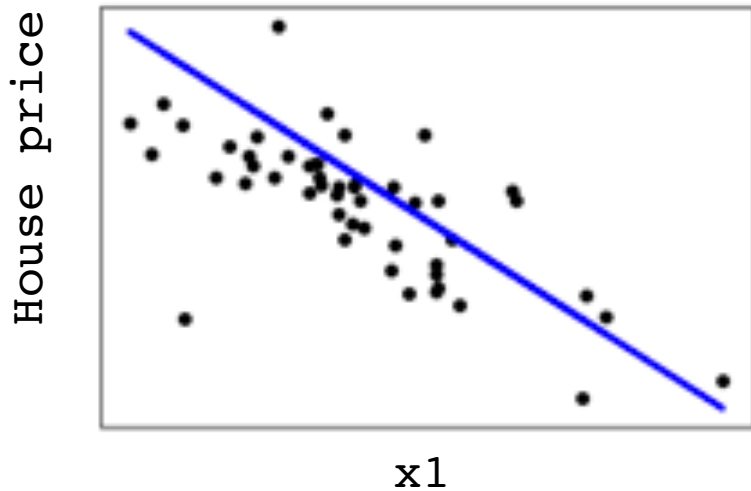
**Notation:**

Each instance  $\mathbf{x}^{(i)}$  has  $d$  features:

$$[x_1, \dots, x_d]$$

$x_j^{(i)}$ : the  $j^{\text{th}}$  feature of instance  $i$

# Linear regression



features

target

$$\{ \langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \dots, \langle \mathbf{x}^{(N)}, \mathbf{y}^{(N)} \rangle \}$$

**Goal:** Predict the target using the features

**Regression task:**

Output is a continuous value ( $\mathbf{y} \in \mathbb{R}$ )

**Notation:**

Each instance  $\mathbf{x}^{(i)}$  has  $d$  features:

$$[x_1, \dots, x_d]$$

$x_j^{(i)}$ : the  $j^{\text{th}}$  feature of instance  $i$



# Linear regression

For each feature  $x_j$  we learn a weight  $w_j$ , so  $w \in \mathbb{R}^d$  and  $b \in \mathbb{R}$ . Given an instance, map it to a real number:

$$\begin{aligned} y &= b + w_1 x_1 + \dots + w_d x_d \\ &= b + \sum w_i x_i = b + w \cdot x \end{aligned}$$

Diagram illustrating the components of the linear regression equation:

- $b$  is labeled "bias" (green arrow).
- $w_1, \dots, w_d$  are collectively labeled "weights" (blue arrows).

For example,  $b = 18$ ,  $w_1 = -0.5$ , etc.

This is a **linear model**.

features target  
 $\{ \langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \dots, \langle \mathbf{x}^{(N)}, \mathbf{y}^{(N)} \rangle \}$

**Goal:** Predict the target using the features

**Regression task:**

Output is a continuous value ( $y \in \mathbb{R}$ )

**Notation:**

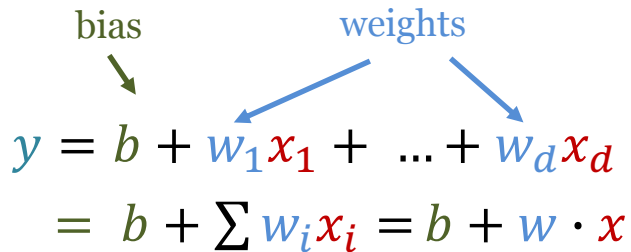
Each instance  $\mathbf{x}^{(i)}$  has  $d$  features:

$[x_1, \dots, x_d]$

$x_j^{(i)}$ : the  $j^{\text{th}}$  feature of instance  $i$

# Aside: bias and notation

For each feature  $x_j$  we learn a weight  $w_j$ , so  $w \in \mathbb{R}^d$  and  $b \in \mathbb{R}$ . Given an instance, map it to a real number:



A diagram illustrating the components of the linear equation. A green arrow labeled "bias" points to the term  $b$  in the equation. Two blue arrows labeled "weights" point to the terms  $w_1 x_1$  and  $w_d x_d$  in the equation.

$$y = b + w_1 x_1 + \dots + w_d x_d$$
$$= b + \sum w_i x_i = b + w \cdot x$$

**Notation:** Sometimes the bias is included as a feature ( $x_0$ ) set to 1. It then becomes:

$$y = w \cdot x$$

# Linear regression

For each feature  $x_j$  we learn a weight  $w_j$

$$y = b + w_1 x_1 + \dots + w_d x_d$$

## Optimization

Find parameters  $(w, b)$  so that the predictions for the *training* data are as close as possible to the known output.

Loss function:  $\frac{1}{2} \sum (\hat{y} - y)^2$

The predicted  $y$

The true  $y$

features

target

$$\{ \langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \dots, \langle \mathbf{x}^{(N)}, \mathbf{y}^{(N)} \rangle \}$$

**Goal:** Predict the target using the features

**Regression task:**

Output is a continuous value ( $y \in \mathbb{R}$ )

**Notation:**

Each instance  $\mathbf{x}^{(i)}$  has  $d$  features:

$$[x_1, \dots, x_d]$$

$x_j^{(i)}$ : the  $j^{\text{th}}$  feature of instance  $i$

# Classification

jkady2682352523@aol.com:

how are you today  
this is amazing website  
there are many kinds of  
phone,camera,laptop,  
television.....  
the price is lower than any other  
website  
the shipping is free

contact: www.cart-10000000.com



*Spam or not?*

features target

$\{ \langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \dots, \langle \mathbf{x}^{(N)}, \mathbf{y}^{(N)} \rangle \}$

**Goal:** Predict the target using the features

**Classification task:**

Output is discrete. Our focus: binary classification:  $\mathbf{y} \in \{0,1\}$  (e.g. 1 = spam)

**Notation:**

Each instance  $\mathbf{x}^{(i)}$  has  $d$  features:  
 $[x_1, \dots, x_d]$

$x_j^{(i)}$ : the  $j^{\text{th}}$  feature of instance  $i$

# Logistic regression

For each feature  $x_j$  we learn a weight  $w_j$ , so  $w \in \mathbb{R}^d$  and  $b \in \mathbb{R}$ . Given an instance, map it to a real number:

$$\begin{aligned} z &= b + w_1 x_1 + \dots + w_d x_d \\ &= b + \sum w_i x_i = b + w \cdot x \end{aligned}$$

Diagram labels: "bias" points to  $b$ , "weights" points to the  $w_i$  terms.

**Classification output** is 0 or 1, but  $z$  can be  $<0$  or  $>1$ . Transform it to a probability (range 0 to 1) using the sigmoid (also called logistic function).

$$p = \frac{1}{1 + e^{-z}}$$

features      target

$$\{ \langle x^{(1)}, y^{(1)} \rangle, \dots, \langle x^{(N)}, y^{(N)} \rangle \}$$

**Goal:** Predict the target using the features

## Classification task:

Output is discrete. Our focus: binary classification:  $y \in \{0,1\}$  (e.g. 1 = spam)

## Notation:

Each instance  $x^{(i)}$  has  $d$  features:  
 $[x_1, \dots, x_d]$

$x_j^{(i)}$ : the  $j^{\text{th}}$  feature of instance  $i$

# Modeling the output

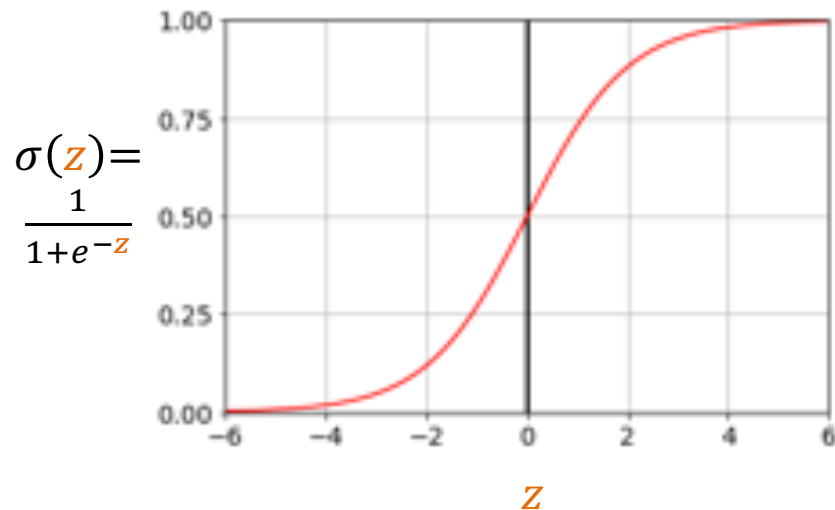
## Logistic regression output:

We want:  $0 \leq \text{output} \leq 1$ .

$$\begin{aligned} p(y = 1|\mathbf{x}) &= \sigma(\mathbf{b} + \mathbf{w} \cdot \mathbf{x}) \\ &= \frac{1}{1 + e^{-(\mathbf{b} + \mathbf{w} \cdot \mathbf{x})}} \end{aligned}$$

$$p(y = 0|\mathbf{x}) = 1 - \sigma(\mathbf{b} + \mathbf{w} \cdot \mathbf{x})$$

## sigmoid function



# Where does the sigmoid function come from?

From probability to odds

p	$p/(1-p)$
0.001	0.001001
0.5	1
0.999	999

# Where does the sigmoid function come from?

From probability to odds

p	p/(1-p)	Log(p/(1-p))
0.001	0.001001	-6.906755
0.5	1	0
0.999	999	6.906755

Logit function

$$z = \log\left(\frac{p}{1-p}\right)$$

So:

$$e^z = \frac{p}{1-p}$$

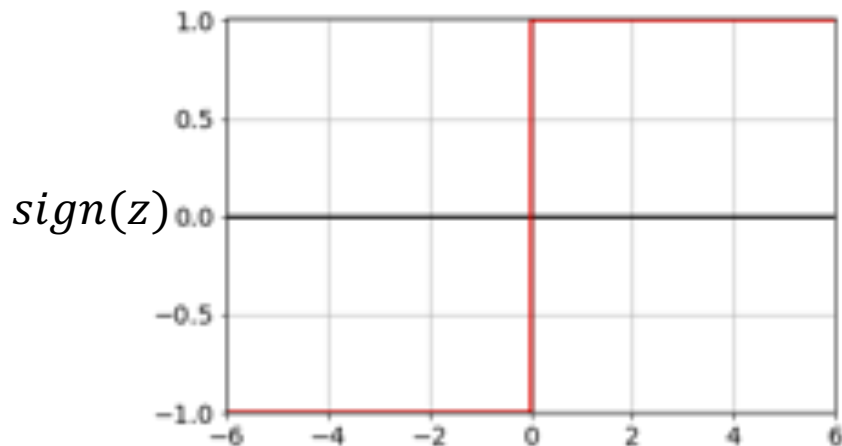
Sigmoid (or logistic) function

$$p = \frac{1}{1 + e^{-z}}$$

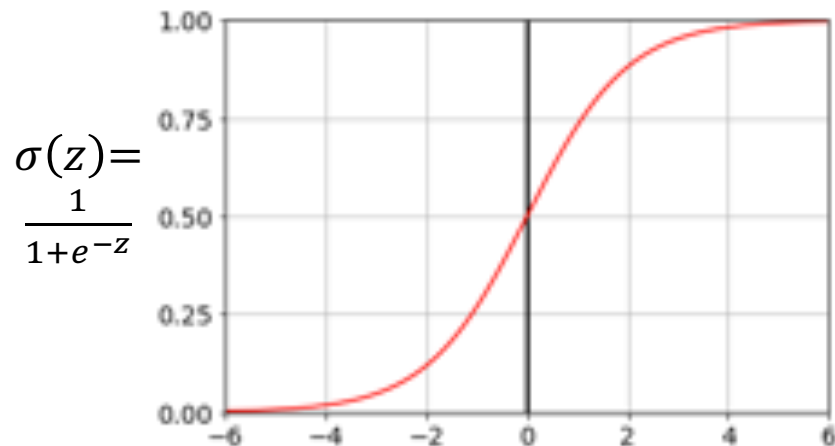


# Aside: why not use the sign function?

**sign function**



**sigmoid function**



*The sign function is not differentiable!*

# Interpretation of the output

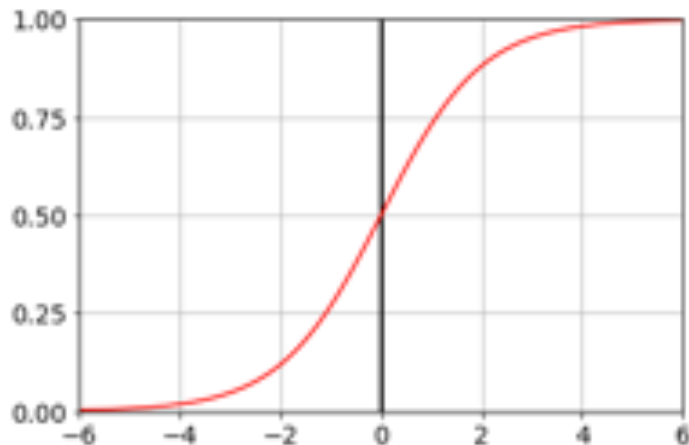
- Model outputs probabilities
  - This gives us much more information than just 0 or 1.
  - For example,  $P(y=1|x) = 0.90$  tells us that the model is very confident. Compare to e.g. when the output  $P(y=1|x) = 0.51$
- Probability can be used for predicting a *class*.
  - For example, predict 1 when  $P(y=1|x) \geq 0.5$

Precision goes up,  
recall goes down

**Question:** What happens to precision and recall when we increase the threshold (e.g. to 0.80?)

# Decision boundary

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-z}}$$



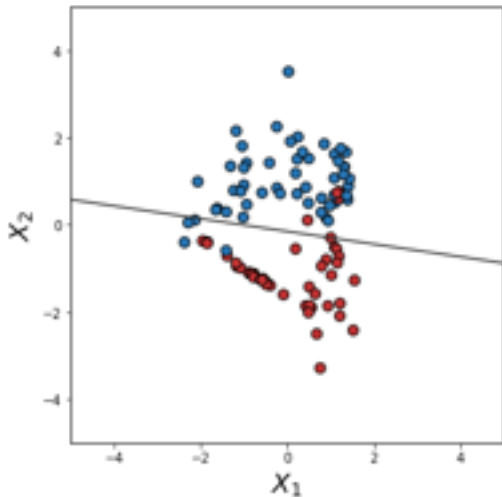
$$z = b + w \cdot \mathbf{x}$$

Predict 1,  
When  $p(y = 1|\mathbf{x}) \geq 0.5$   
Is same as when  $z \geq 0$

Predict 0,  
When  $p(y = 1|\mathbf{x}) < 0.5$   
Is same as when  $z < 0$

**Linear classification rule!**

# Decision boundaries



$$b = 0.37$$

$$w_1 = 0.35$$

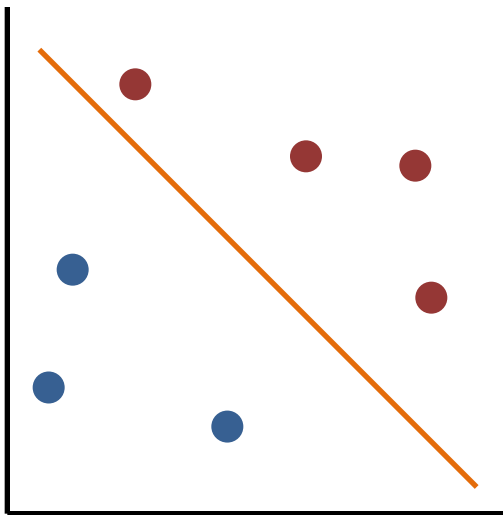
$$w_2 = 2.41$$

Logistic regression is a linear classifier!

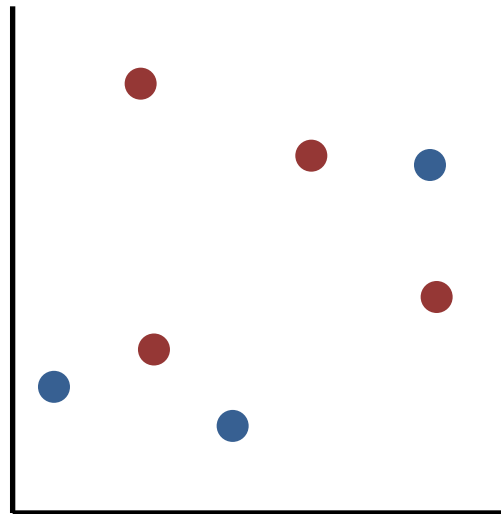
**Question:** Are decision trees linear classifiers?  
Are nearest-neighbor models linear classifiers?

Both are not linear classifiers

# Linearly separable?



Yes!



No!

# Logistic regression: Example

<i>feature</i>	$w_i$	$x_i$
Is the advertisement shown at the top of the page? (1=yes, 0 = no)	0.40	1
Click through rate of the user (0..1)	0.90	0.1
Click through rate of previous showings of the advertisements (other users) (0...1)	1.2	0.2
Capitalized text? (1=yes, 0=no)	0.5	1

$$b = -1$$

Will the user click on the advertisement?

$$z = -1 + 1 * 0.40 + 0.90 * 0.1 + 1.2 * 0.2 + 0.5 * 1 = 0.23$$

$$p = \frac{1}{1 + e^{-z}} = 0.557$$

Yes!

# Logistic regression

For each feature  $x_j$  we learn a weight  $w_j$ , so  $w \in \mathbb{R}^d$  and  $b \in \mathbb{R}$ . Given an instance, map it to a real number:

$$\begin{aligned} z &= b + w_1 x_1 + \dots + w_d x_d \\ &= b + \sum w_i x_i = b + w \cdot x \end{aligned}$$

Diagram labels: "bias" points to  $b$ , "weights" points to the  $w_i$  terms.

$$p(y = 1 | x) = \frac{1}{1 + e^{-z}}$$

How do we learn the weights  $w$  and  $b$ ?

Needed: (1) Loss function and  
(2) Optimization algorithm

features target

$$\{ \langle x^{(1)}, y^{(1)} \rangle, \dots, \langle x^{(N)}, y^{(N)} \rangle \}$$

**Goal:** Predict the target using the features

**Classification task:**

Output is discrete. Our focus: binary classification:  $y \in \{0, 1\}$  (e.g. 1 = spam)

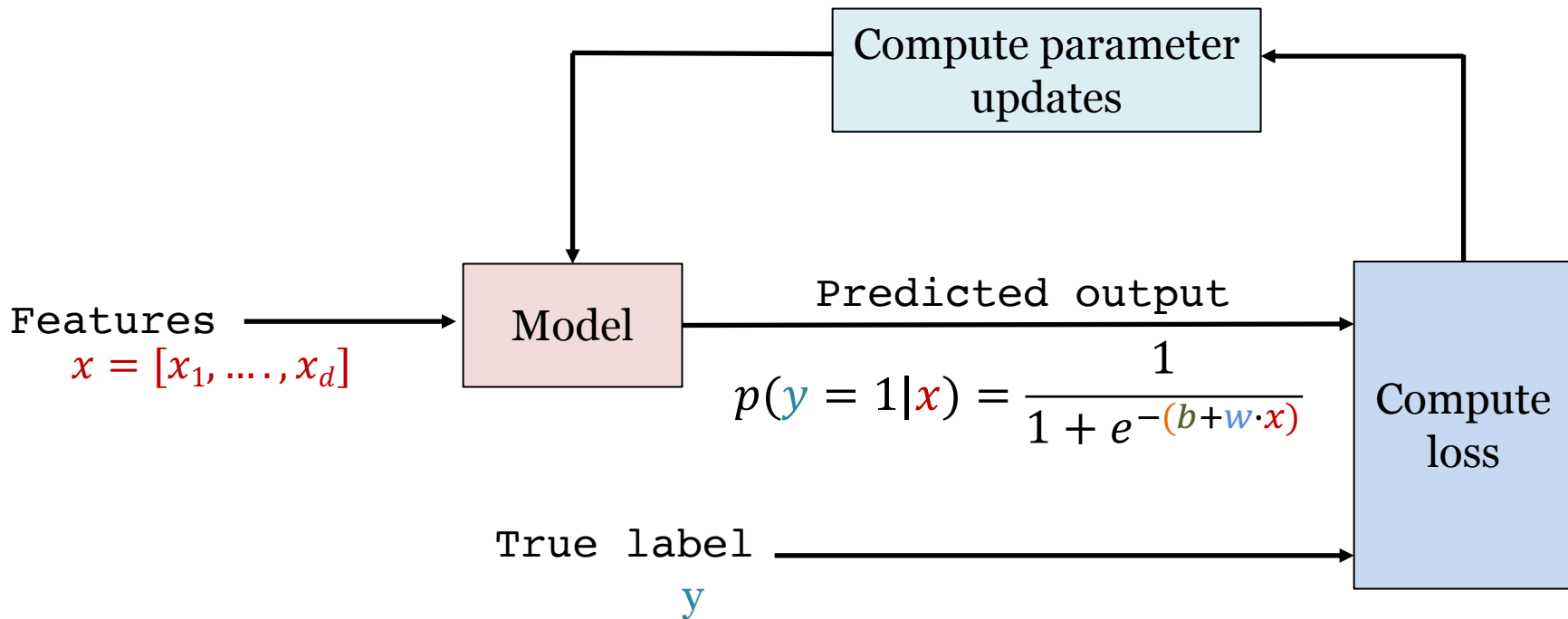
**Notation:**

Each instance  $x^{(i)}$  has  $d$  features:

$$[x_1, \dots, x_d]$$

$x_j^{(i)}$ : the  $j^{\text{th}}$  feature of instance  $i$

# Learning the parameters





# Loss function

## Notation:

$y$  = true label

$\hat{y}$  = classifier output

$= P(y=1 \mid \mathbf{x}; \boldsymbol{\theta})$

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

We want to learn parameters ( $\boldsymbol{\theta} = w, b$ ) that maximize the probability of the true labels ( $y$ ) in the training data ( $\mathbf{x}$ ).

$$\text{if } y=1: P(y=1 \mid \mathbf{x}; \boldsymbol{\theta}) = \hat{y}$$

$$\text{if } y=0: P(y=0 \mid \mathbf{x}; \boldsymbol{\theta}) = 1 - P(y=1 \mid \mathbf{x}; \boldsymbol{\theta}) = 1 - \hat{y}$$

Trick, combine this into one equation!

$$p(y \mid \mathbf{x}; \boldsymbol{\theta}) = \underbrace{\hat{y}^y}_{y=1} \underbrace{(1 - \hat{y})^{1-y}}_{y=0}$$

# Loss function

## Notation:

$y$  = true label

$\hat{y}$  = classifier output

=  $P(y=1 \mid \mathbf{x}; \boldsymbol{\theta})$

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

$$p(y \mid \mathbf{x}; \boldsymbol{\theta}) = \hat{y}^y (1 - \hat{y})^{1-y}$$

Log transformation (a monotone transformation:  
parameters that maximize  $p(y \mid \mathbf{x}, \boldsymbol{\theta})$  will also  
maximize  $\log p(y \mid \mathbf{x}; \boldsymbol{\theta})$ )

$$\begin{aligned}\log(a^b) &= b \log(a) \\ \log(ab) &= \log(a) + \log(b)\end{aligned}$$

$$\log p(y \mid \mathbf{x}; \boldsymbol{\theta}) = y \log \hat{y} + (1-y) \log (1-\hat{y})$$

**Turning it into a loss function (we want to minimize this):** flip the sign!

**Cross-entropy loss** =  $L(\hat{y}, y)$

“How much does the classifier output differ from the correct output?”

$$= - \log p(y \mid \mathbf{x}; \boldsymbol{\theta})$$

$$= - (y \log \hat{y} + (1-y) \log (1-\hat{y}))$$

# Loss function

## Notation:

$y$  = true label

$\hat{y}$  = classifier output

=  $P(y=1 \mid \mathbf{x}; \boldsymbol{\theta})$

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

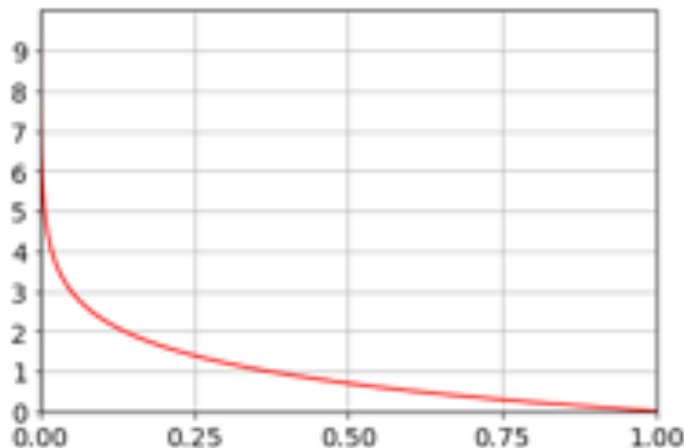
**Cross-entropy loss** =  $L(\hat{y}, y)$

$$= -\log p(y \mid \mathbf{x}; \boldsymbol{\theta})$$

$$= - (y \log \hat{y} + (1-y) \log (1-\hat{y}))$$

“How much does the classifier output differ from the correct output?”

**when  $y = 1$ :**  $L(\hat{y}, y) = -\log \hat{y}$



# Loss function

Recall:

$\hat{y}$  = classifier output

$y$  = true label

We want to find the parameters  $\boldsymbol{\theta} = w, b$  that minimize the loss for the whole dataset with  $N$  examples:

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

# Gradient descent

**Goal:** Find the parameters  $\theta = w, b$  that minimizes this loss

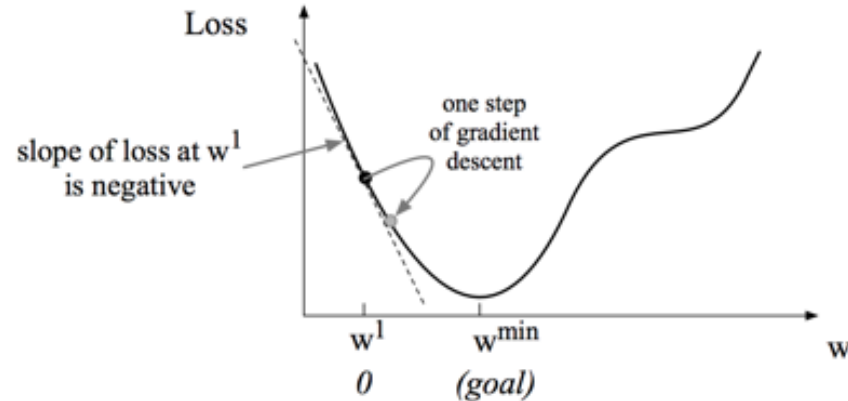
$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum_i L(\hat{y}^{(i)}, y^{(i)}; \theta)$$

Let's start simple! Let  $w$  be a scalar.

Move in the reverse direction from the slope of the loss function

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

next step    current step    learning rate    slope



[J&M, chapter 5, Fig 5.3]

*Gradient is a multi-variable generalization of the slope!*

# Gradient descent example

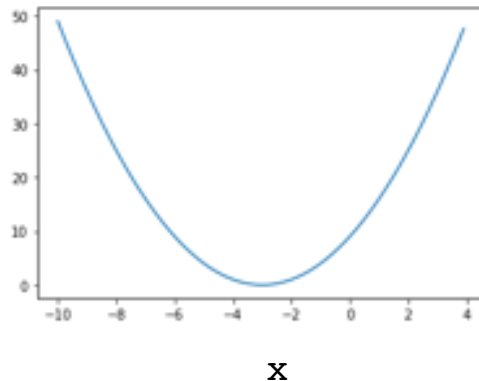
$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

next step   current step   learning rate   slope

Let's start at  $x_0 = 4$ ,  
learning rate = 0.25

$$x_1 = 4 - 0.25 * (2 * (4 + 3)) = 0.5$$

$$y = (x + 3)^2$$
$$dy = 2 * (x + 3)$$



4  
0.5  
-1.25  
-2.125  
-2.5625  
-2.78125  
-2.890625  
-2.9453125  
-2.97265625  
-2.986328125  
-2.9931640625

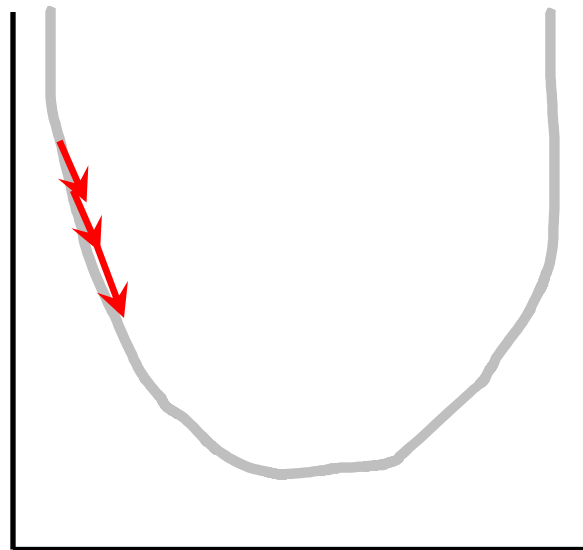
Converges to -3!

# Gradient descent: learning rate

When it is too **large**, gradient descent can even lead to increased training error.

When it is too **small**, training is slow and optimization might get stuck.

*Usually start with a higher learning rate and decrease it over time.*

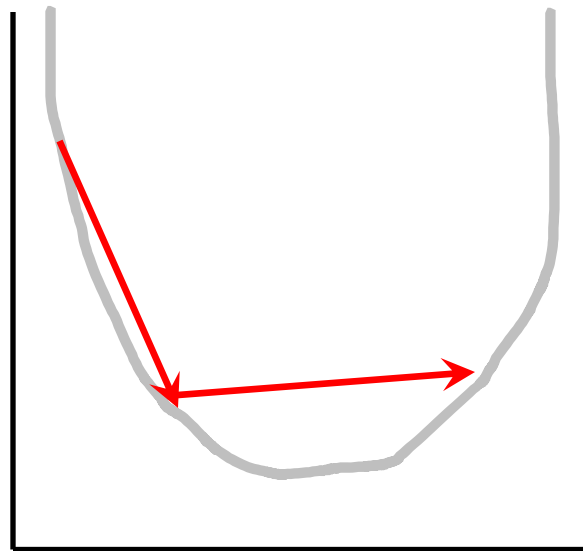


# Gradient descent: learning rate

When it is too **large**, gradient descent can even lead to increased training error.

When it is too **small**, training is slow and optimization might get stuck

*Usually start with a higher learning rate and decrease it over time.*





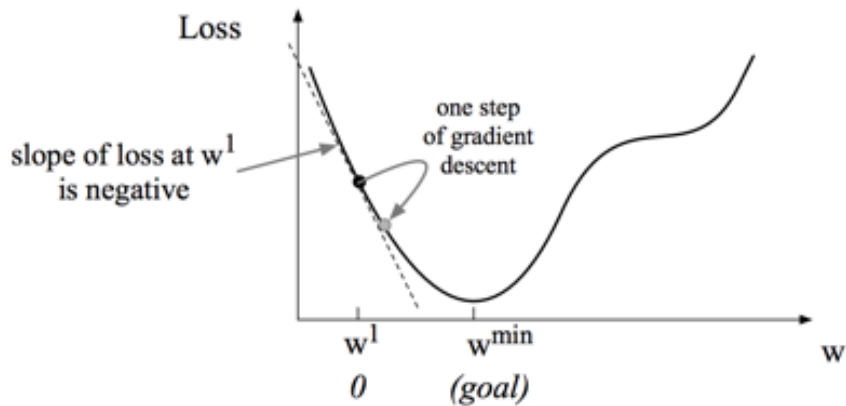
# Gradient Descent

**Goal:** Find the parameters  $\theta = w, b$  that minimizes this loss

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum L(\hat{y}, y; \theta)$$

Gradient is a multi-variable generalization of the slope.

$$\nabla_{\theta} L(\hat{y}, y; \theta) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(\hat{y}, y; \theta) \\ \frac{\partial}{\partial w_2} L(\hat{y}, y; \theta) \\ \dots \end{bmatrix}$$



$$\theta^{t+1} = \theta^t - \eta \nabla_{\theta} L(\hat{y}, y; \theta)$$

[J&M, chapter 5, Fig 5.3]

next step   current step   learning rate   gradient

# Gradient logistic regression

Recall:

$\hat{y}$  = classifier output

$y$  = true label

$\log(a^b) = b \log(a)$

$$\begin{aligned}\text{Cross-entropy loss} &= L(\hat{y}, y) \\ &= -\log p(y|x; \theta) \\ &= -(y \log \hat{y} + (1-y) \log (1-\hat{y}))\end{aligned}$$

$$\frac{\partial L(\hat{y}, y)}{\partial w_j} = (\hat{y} - y)x_j = (\sigma(b + w \cdot x) - y)x_j$$

# Gradient Descent

An  
alternative is  
mini-batch  
training:

*Compute  
average loss  
over a mini-  
batch of  $m$   
examples*


```
function STOCHASTIC GRADIENT DESCENT( $L()$ ,  $f()$ ,  $x$ ,  $y$ ) returns  $\theta$ 
    # where:  $L$  is the loss function
    #  $f$  is a function parameterized by  $\theta$ 
    #  $x$  is the set of training inputs  $x^{(1)}, x^{(2)}, \dots, x^{(n)}$ 
    #  $y$  is the set of training outputs (labels)  $y^{(1)}, y^{(2)}, \dots, y^{(n)}$ 

     $\theta \leftarrow 0$ 
    repeat til done # see caption
        For each training tuple  $(x^{(i)}, y^{(i)})$  (in random order)
            1. Optional (for reporting): # How are we doing on this tuple?
               Compute  $\hat{y}^{(i)} = f(x^{(i)}; \theta)$  # What is our estimated output  $\hat{y}$ ?
               Compute the loss  $L(\hat{y}^{(i)}, y^{(i)})$  # How far off is  $\hat{y}^{(i)}$  from the true output  $y^{(i)}$ ?
            2.  $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$  # How should we move  $\theta$  to maximize loss?
            3.  $\theta \leftarrow \theta - \eta g$  # Go the other way instead
    return  $\theta$ 
```

# Regularization

To prevent overfitting, a regularization term  $R(w)$  can be added. Recall, we want to find the parameters  $\theta = w, b$  that minimizes the loss. We now add a regularization term ( $R(\theta)$ )

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum L(\hat{y}, y; \theta) + \lambda R(\theta)$$



This is a form of inductive bias!

**RECAP!**

**The L2 norm:**

$$\|a\|_2 = \sqrt{\sum a_i^2}$$

**The L1 norm:**

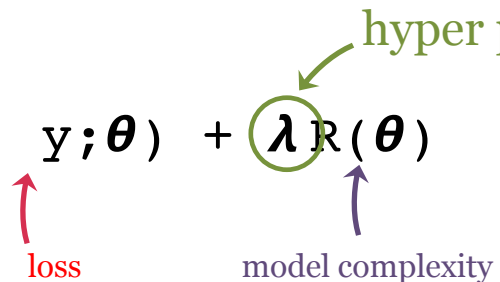
$$\|a\|_1 = \sum |a_i|$$

# Regularization

**Question:** We can't set the regularization parameter  $\lambda$  by looking at the training error, why?

To prevent overfitting, a regularization term  $R(w)$  can be added. Recall, we want to find the parameters  $\theta = w, b$  that minimizes the loss. We now add a regularization term ( $R(\theta)$ )

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum \mathcal{L}(\hat{y}, y; \theta) + \lambda R(\theta)$$

  
loss                      hyper parameter                      model complexity

This is a form of inductive bias!

**L2 regularization (or, ridge regularization):**  $R(\theta) = \|\theta\|_2^2 = \sum \theta_i^2$   
(the square of the L2 norm of the weight values)

**L1 regularization (or, lasso regularization):**  $R(\theta) = \|\theta\|_1 = \sum |\theta_i|$

**RECAP!**

**The L2 norm:**

$$\|a\|_2 = \sqrt{\sum a_i^2}$$

**The L1 norm:**

$$\|a\|_1 = \sum |a_i|$$

# Multiclass classification

Recall: the **sigmoid**.

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

The **softmax** is a generalization of the sigmoid to  $k$  classes.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

Input vector  $\mathbf{z} = [z_1, z_2, \dots, z_k] \rightarrow$   
 $[\text{softmax}(z_1), \text{softmax}(z_2), \dots, \text{softmax}(z_k)]$

# Comparison with decision trees & nearest neighbors

## Features:

- Decision trees: only a small number of features is used
- K-nearest neighbor: all features are used with equal weight
- Logistic regression: All features are used, but some features are more important than others.

## Decision boundaries:

- K-nearest neighbors and decision trees can have *non-linear* decision boundaries
- Logistic regression results in a *linear decision* boundary

# Neural networks



# Neural networks

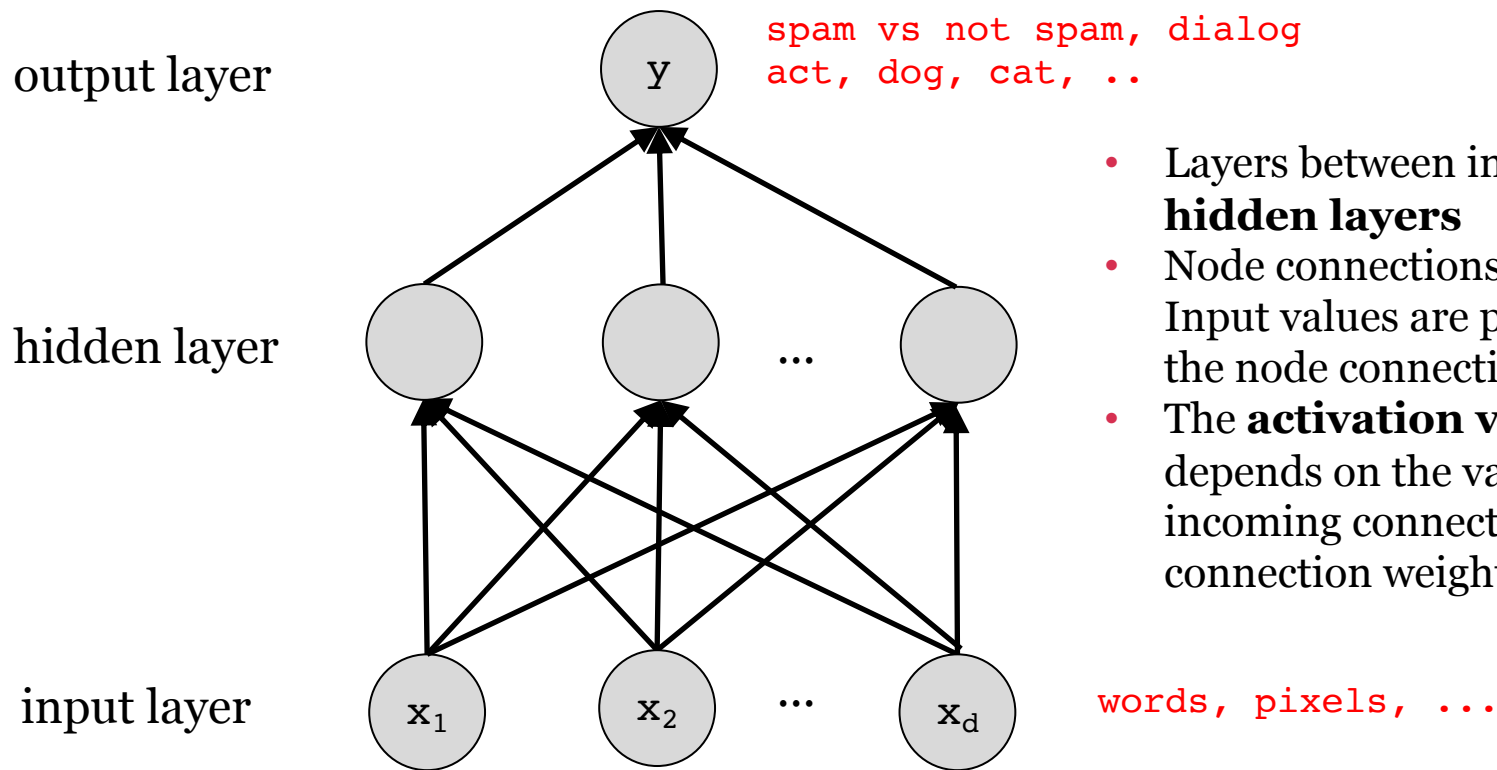
Have been around for a *long time*:

- McCulloch-Pitts neuron (McCulloch and Pitts, 1943)
- Perceptron (Rosenblatt 1958)
- LeNet-5 (LeCun et al. 1998): convolutional network for digital recognition
- ...

*Now:*

- Better optimization methods
- New non-linear functions (ReLU)
- More hidden layers ('deep learning')
- Better hardware (CPUs, GPUs, TPUs,..)

# A simple neural network



- Layers between input and output: **hidden layers**
- Node connections are **weighted**  
Input values are propagated along the node connections
- The **activation value** of a node depends on the value of nodes of incoming connections and the connection weight

# Building blocks of neural nets: units

$$\begin{aligned} z &= b + w_1 x_1 + \dots + w_d x_d \\ &= b + \sum w_i x_i = b + w \cdot x \end{aligned}$$

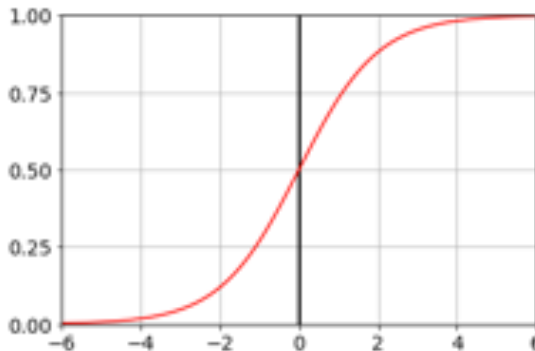
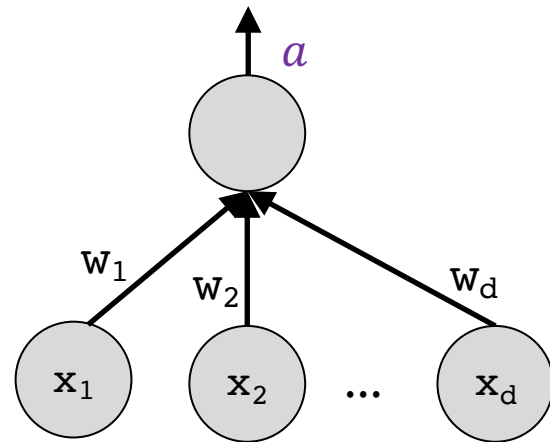
Neural units apply a **non-linear activation function**  $f$  to  $z$ , resulting in an **activation** value

$$a = f(z)$$

Usually used for  
output layer  
(binary  
classification)

**sigmoid**

*This should look  
familiar!  
(logistic regression)*



# Building blocks of neural nets: units

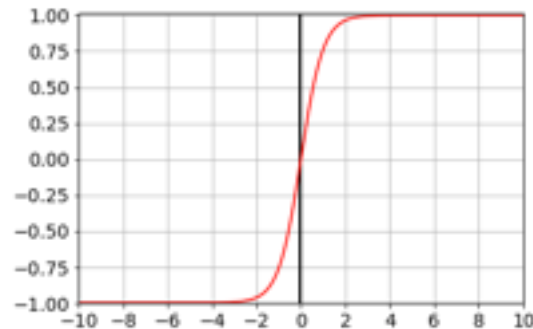
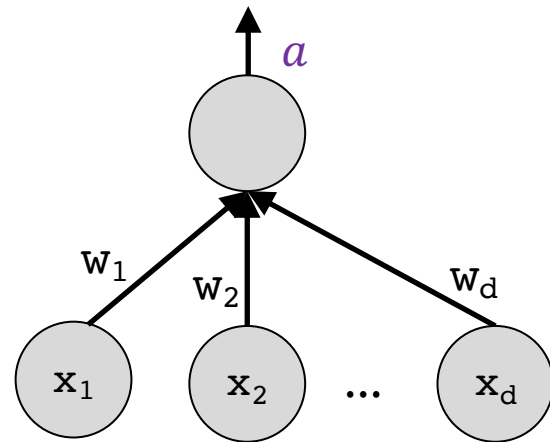
$$\begin{aligned} z &= b + w_1 x_1 + \dots + w_d x_d \\ &= b + \sum w_i x_i = b + w \cdot x \end{aligned}$$

Neural units apply a **non-linear activation function**  $f$  to  $z$ , resulting in an **activation** value

$$a = f(z)$$

Usually used for  
hidden layers

$$\begin{aligned} &\text{tanh} \\ f(z) &= \frac{e^z - e^{-z}}{e^z + e^{-z}} \end{aligned}$$



# Building blocks of neural nets: units

$$\begin{aligned} z &= b + w_1 x_1 + \dots + w_d x_d \\ &= b + \sum w_i x_i = b + w \cdot x \end{aligned}$$

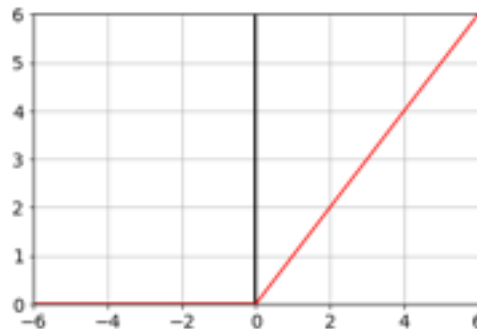
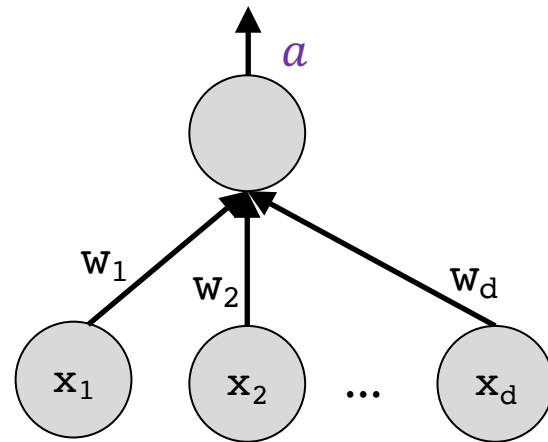
Neural units apply a **non-linear activation function**  $f$  to  $z$ , resulting in an **activation** value

$$a = f(z)$$

Usually used for  
hidden layers  
(often 'default'  
choice)

**Rectified linear unit  
(ReLU)**

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



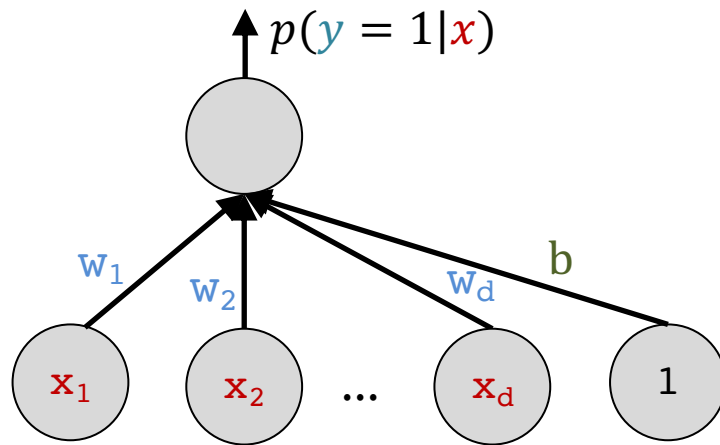
RECAP!

# Logistic Regression

**Logistic regression:**

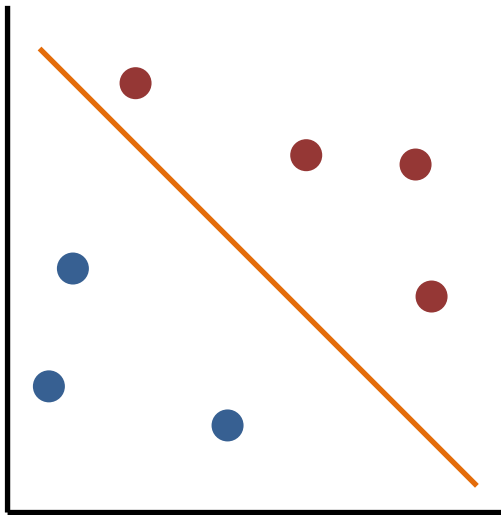
$$p(y = 1|x) = \frac{1}{1 + e^{-z}} \quad \text{with } z = b + w \cdot x$$

Logistic regression is just a neural network with **no** hidden layers and a sigmoid activation function!

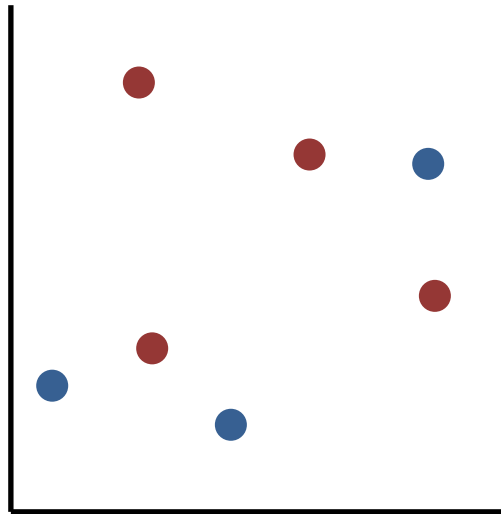


RECAP!

# Linearly separable?



Yes!



No!

We need **non-linear** activation functions  
to model more complex decision boundaries!

*(A network with multiple layers but only linear activation  
functions still results in a linear decision boundary!)*

# XOR example

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

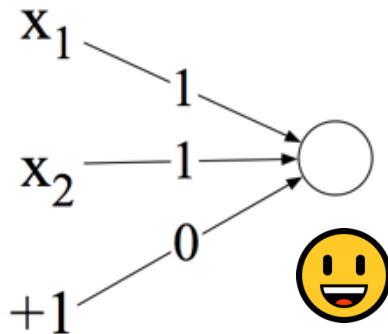
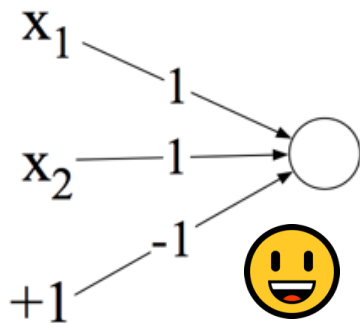
AND

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR



?



Perceptron  
(no non-linear activation)

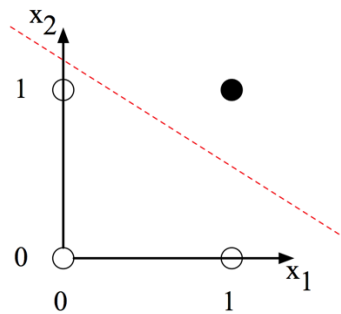
0, if  $w \cdot x + b \leq 0$   
1, if  $w \cdot x + b > 0$



# XOR example

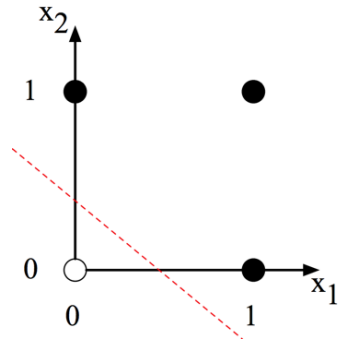
$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

AND



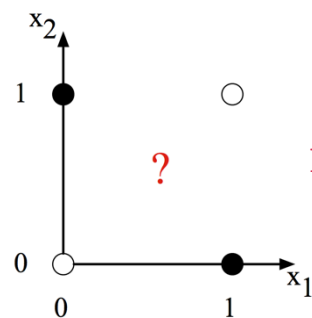
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

OR



$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

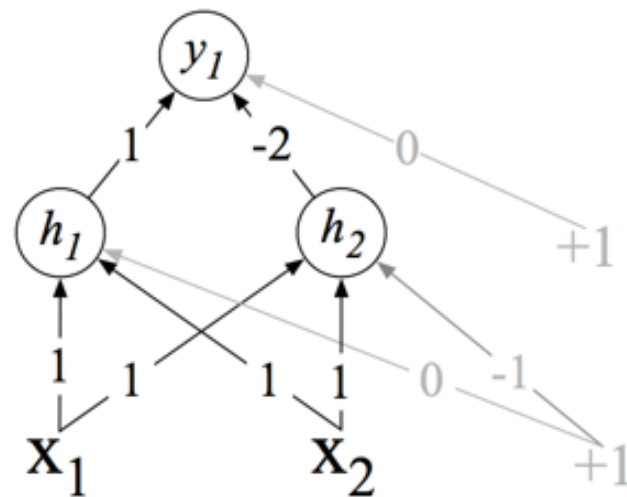
XOR



**XOR:**  
not linearly separable!

# XOR network

x1	x2	h1	h2	y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0

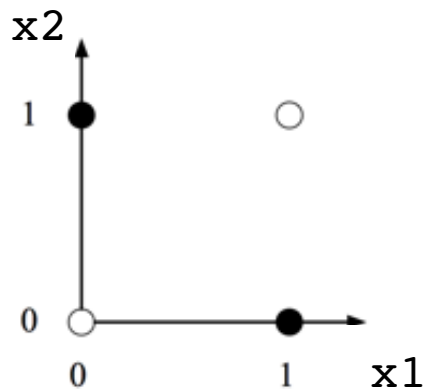


[J&M, Fig. 7.6,  
based on Goodfellow et al. 2016]

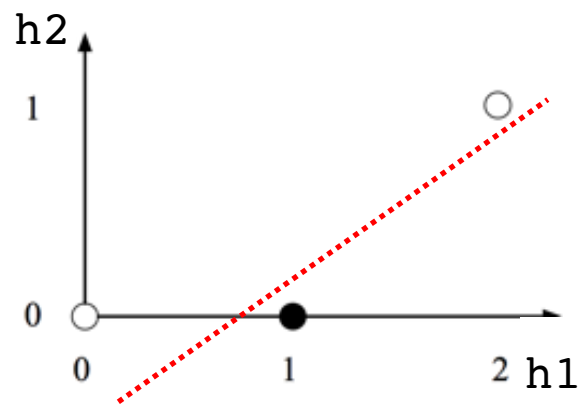
The units are ReLU units ( $\max(0, x)$ )

# XOR network: Learning representations

$x_1$	$x_2$	$h_1$	$h_2$	$y$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0



a) The original  $x$  space



b) The new  $h$  space

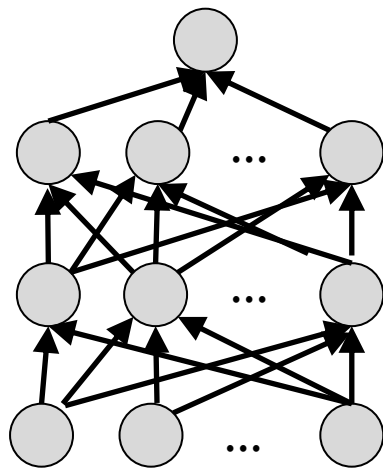
**Question:** Is the new  $h$  space linearly separable?

[J&M, Fig. 7.7,  
based on Goodfellow et al. 2016]

# Learning representations

**Previously** (logistic regression, decision trees, etc...): Features were *manually* specified.

**Deep neural networks:** Input are usually *low level features* (characters, words) or pixels). *Neural networks can automatically learn useful representations of the input at different levels of abstraction.*



## Language:

Lower layers usually capture syntactic information, higher layers capture semantic information

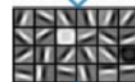
## Feature representation



3rd layer  
"Objects"



2nd layer  
"Object parts"



1st layer  
"Edges"



Pixels

<https://deeplearningworkshopnips2010.files.wordpress.com/2010/09/nips10-workshop-tutorial-final.pdf>

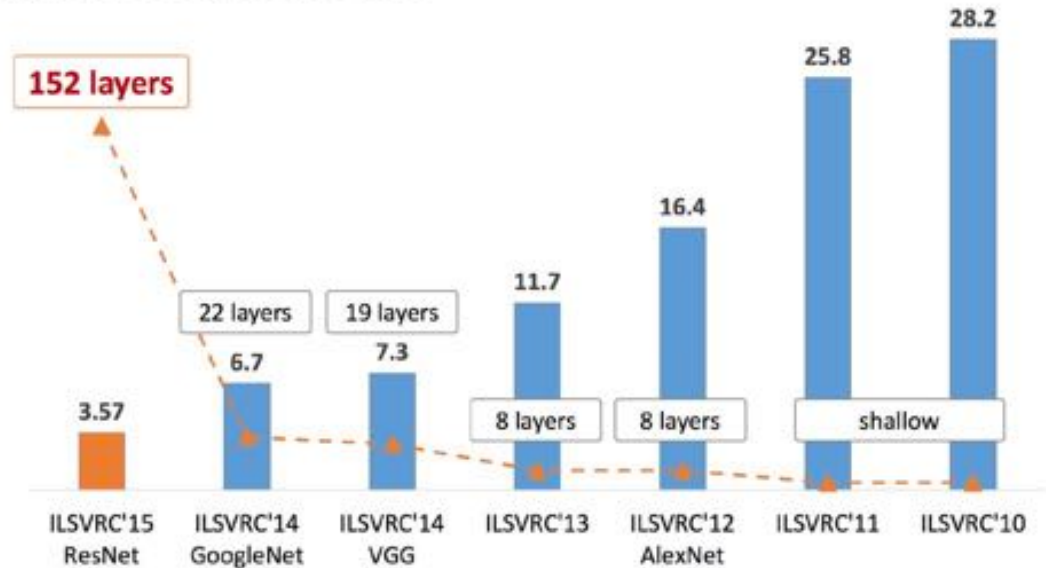
# Deep neural networks

**Deep** neural networks have **many** layers



IMAGENET

ImageNet experiments



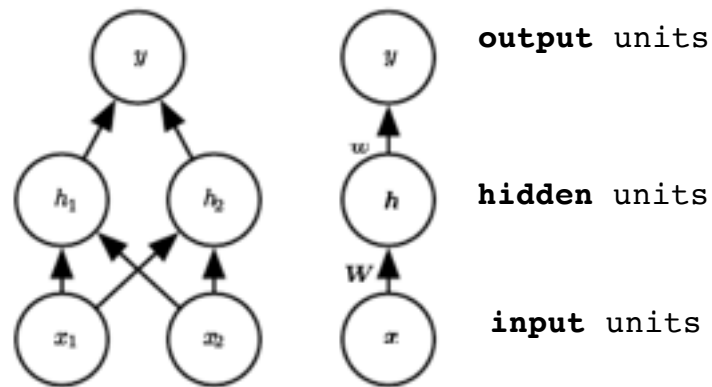
ImageNet Classification top-5 error (%)

# Feed forward network

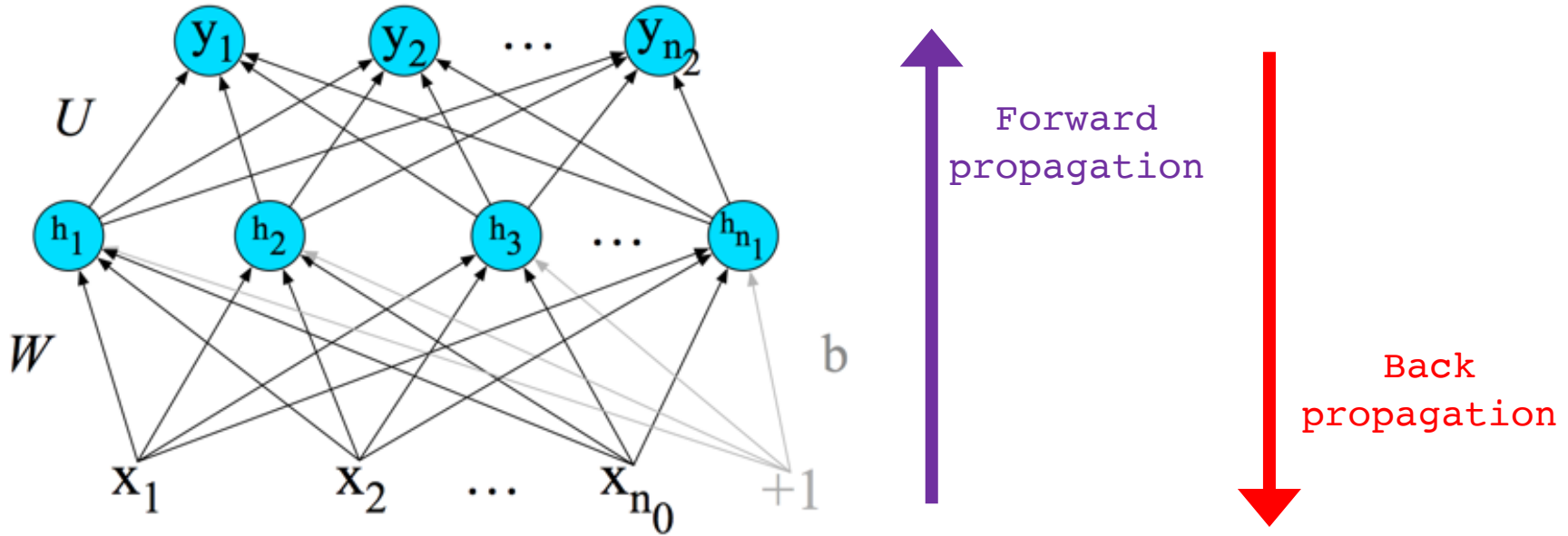
A **feed-forward network**:

- A multilayer network
- Units are connected but no cycles
- The output from units in each layer are passed to units in the next layer, no output passed back to lower layers

Also sometimes called:  
**multi-layer perceptrons (or MLPs)**



# Feed forward network



[J&M, Fig. 7.8]

# Matrices

$$\mathbf{B} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\mathbf{B} \in \mathbb{R}^{2 \times 3}$$

$$\mathbf{B}_{12} = 2$$

$$\mathbf{H} = \begin{bmatrix} H_{11} & \cdots & H_{1n} \\ \vdots & \ddots & \vdots \\ H_{m1} & \cdots & H_{mn} \end{bmatrix}$$

$$\mathbf{H} \in \mathbb{R}^{m \times n}$$

$$\mathbf{B}\mathbf{a} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 * 2 + 2 * 0 + 3 * 1 \\ 4 * 2 + 5 * 0 + 6 * 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 14 \end{bmatrix}$$

## Vectors:

$$\mathbf{a} = [2, 0, 1]$$

$$\mathbf{a} \in \mathbb{R}^3$$

$$\mathbf{c} = [c_1, \dots, c_d]$$

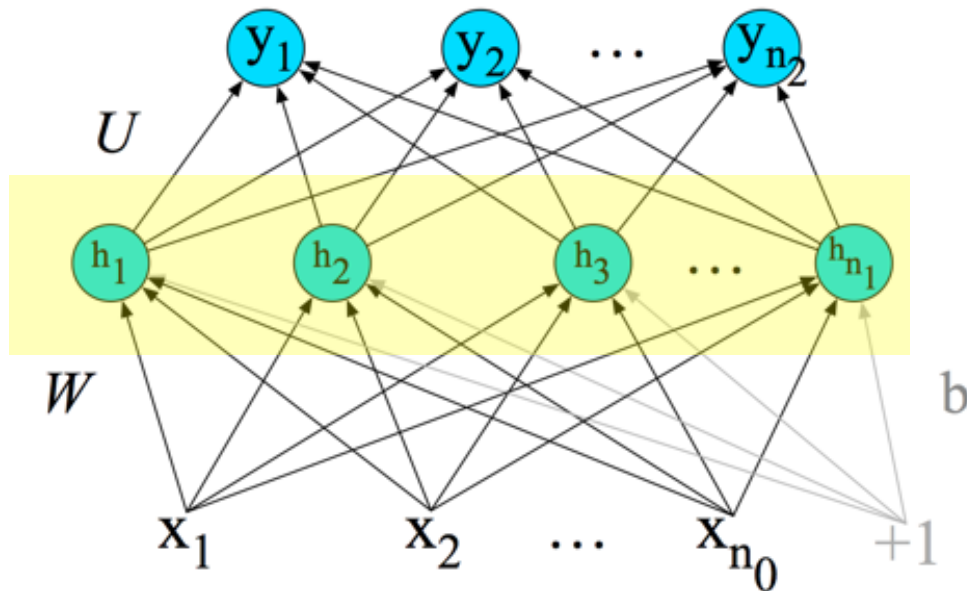
$$\mathbf{c} \in \mathbb{R}^d$$

## See also:

- The Matrix Cookbook
- Books/lectures by Gilbert Strang
- Python: numpy



# Feed forward network: forward propagation



[J&M, Fig. 7.8]

$$\begin{array}{ll} \mathbf{x} \in \mathbb{R}^{n_0} & \mathbf{b} \in \mathbb{R}^{n_1} \\ \mathbf{W} \in \mathbb{R}^{n_1 \times n_0} & \mathbf{h} \in \mathbb{R}^{n_1} \end{array}$$

Recall: one single hidden unit:

$$h = g(b + w \cdot x)$$

For an entire hidden layer:

$$h_1 = g(b_1 + W_{11}x_1 + \dots + W_{1n_0}x_{n_0})$$

$$h_2 = g(b_2 + W_{21}x_1 + \dots + W_{2n_0}x_{n_0})$$

Etc..

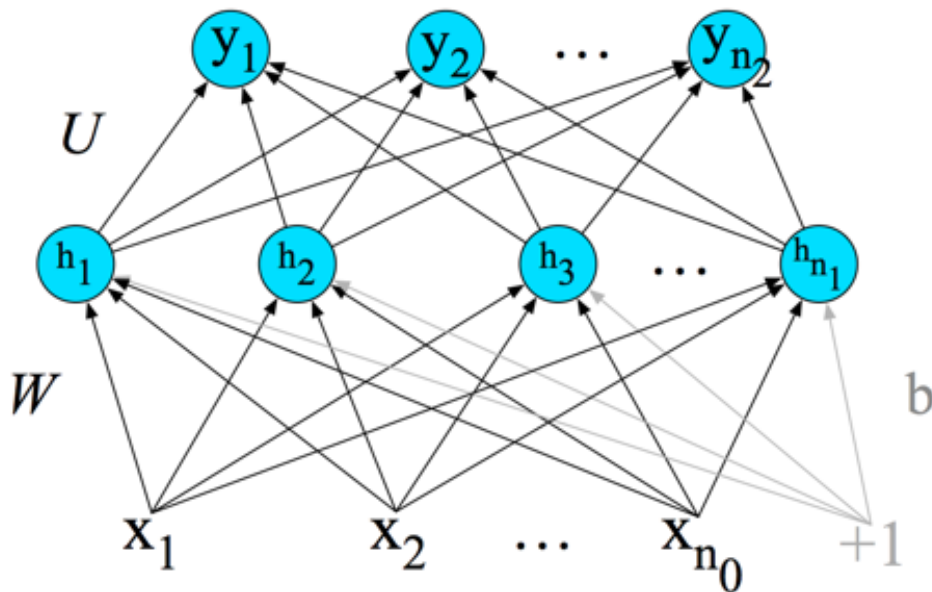
$W_{ij}$  the weight of the connection between  $h_i$  and  $x_j$

Using matrix operations:

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

$\curvearrowright$  e.g. sigmoid or ReLU

# Feed forward network: forward propagation



[J&M, Fig. 7.8]

$$h = g(b + Wx)$$

$$z = Uh$$

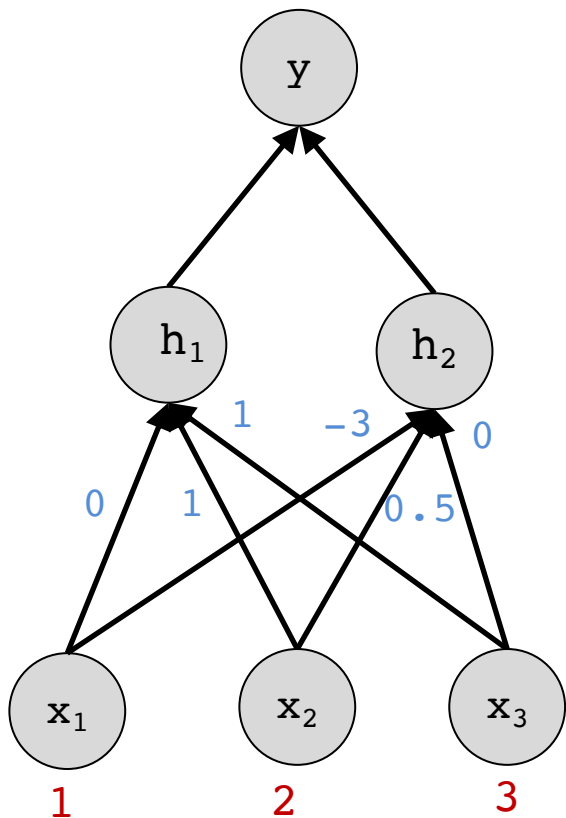
$$y = \text{softmax}(z)$$

*“Just logistic regression  
on features (or representations)  
learned in  $h$ ”*

$$x \in \mathbb{R}^{n_0} \quad b \in \mathbb{R}^{n_1} \quad U \in \mathbb{R}^{n_2 \times n_1}$$

$$W \in \mathbb{R}^{n_1 \times n_0} \quad h \in \mathbb{R}^{n_1}$$

# Feed forward network: example



$$\mathbf{x} = [1, 2, 3]$$

$$h_1 = g(0 * 1 + 1 * 2 + 1 * 3) = g(5)$$

$$h_2 = g(-3 * 1 + 0.5 * 2 + 0 * 3) = g(-2)$$

Using ReLU activation functions:

$$\mathbf{h} = [h_1, h_2] = [\text{ReLU}(5), \text{ReLU}(-2)] = [5, 0]$$

**Using matrix multiplications:**

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 \\ -3 & 0.5 & 0 \end{bmatrix}$$

$$\mathbf{W}\mathbf{x} = [5, -2]$$

$$\mathbf{h} = \text{ReLU}(\mathbf{W}\mathbf{x}) = [5, 0]$$

Recall:

$$\text{ReLU}(x) = \max(x, 0)$$

# Training a feed forward network

Same ingredients as for logistic regression:

- Loss function
- Optimization algorithm

# Training a feed forward network

Same ingredients as for logistic regression:

- **Loss function**
- Optimization algorithm

$$\begin{aligned} \text{Cross-entropy loss} &= L(\hat{y}, y) \\ (seen\ before) \quad &= -\log p(y|x; \theta) \end{aligned}$$

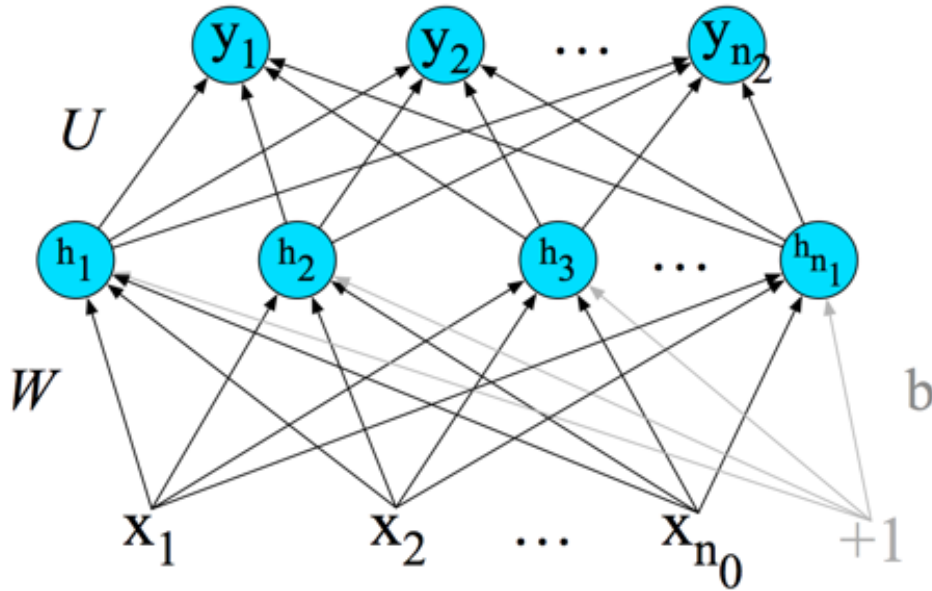
# Training a feed forward network

Same ingredients as for logistic regression:

- Loss function
- **Optimization algorithm**

Similar idea, but calculating the gradient is a bit more complicated than for logistic regression..

# Feed forward network: back propagation



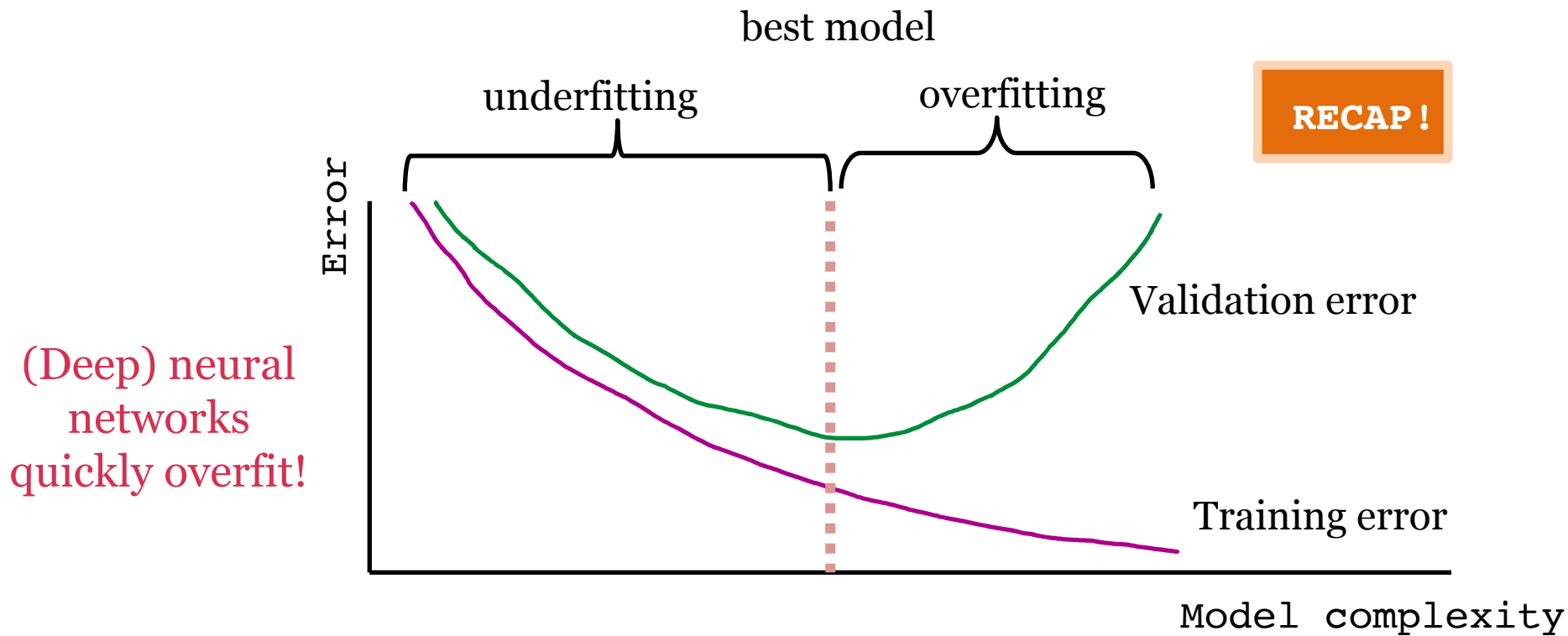
[J&M, Fig. 7.8]

Back  
propagation

*Intuitively, the (derivative of the) error for a node is distributed among previous nodes according to the weights*

*(you don't need to know the details of back propagation for this class)*

# Preventing overfitting





# Regularization

**Logistic regression:**

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \frac{1}{N} \sum \mathcal{L}(\hat{y}, y; \boldsymbol{\theta}) + \lambda \mathcal{R}(\boldsymbol{\theta})$$

Diagram annotations:  
- A red arrow points from the text "loss" to the loss function  $\mathcal{L}$ .  
- A green arrow points from the text "hyper parameter" to the parameter  $\lambda$ .  
- A purple arrow points from the text "model complexity" to the regularization term  $\mathcal{R}(\boldsymbol{\theta})$ .

**RECAP !**

**L2 regularization**

$$\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum \boldsymbol{\theta}_i^2$$

**L1 regularization**

$$\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum |\boldsymbol{\theta}_i|$$

**Same idea for neural networks, but now for matrices:**

$$\mathcal{R}(W) = \|W\|_F^2 = \sum_i \sum_j W_{ij}^2$$

L2 regularization, for historic purposes this is called the (squared) Frobenius norm

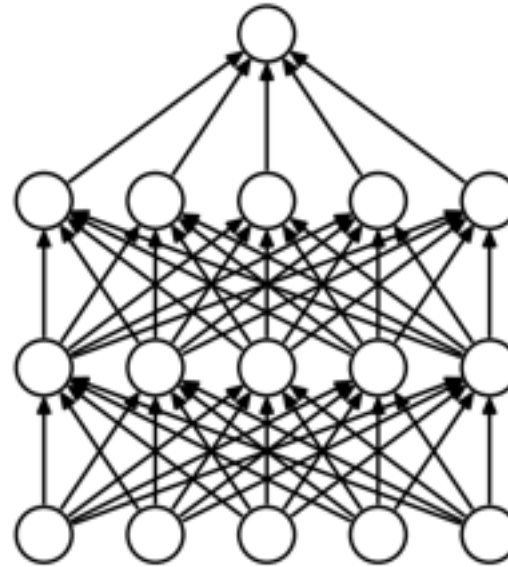
$$\mathcal{R}(W) = \|W\|_1 = \sum_i \sum_j |W_{ij}|$$

L1 regularization

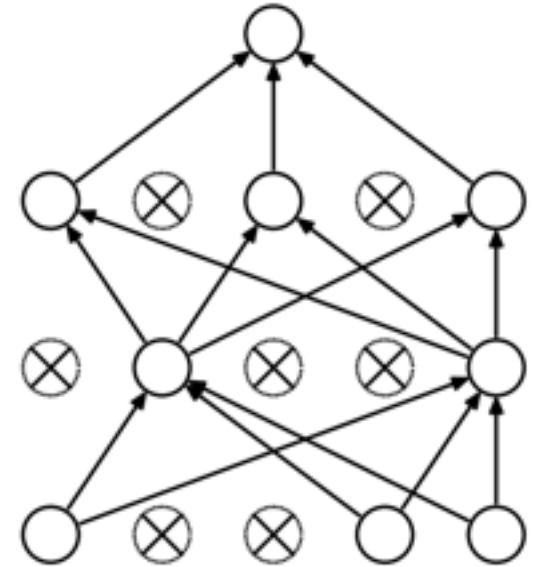
# Preventing overfitting: dropout

Randomly set some neurons to zero during training.

**Hyperparameter:**  
The probability of setting neurons to zero (0.5 is common)



(a) Standard Neural Net



(b) After applying dropout.

**Question:** Your neural network is underfitting. Should you increase or decrease the dropout probability?

Srivastava et al. 2014

# Hyperparameters

- Number of hidden layers
- Size of hidden layers at each layer
- Learning rate
- Batch size
- Drop out rate
- Regularization parameters
- Activation functions
- *and so on ....*

Lots of ‘tricks’ to train neural networks!

See also: <https://karpathy.github.io/2019/04/25/recipe/>  
(A Recipe for Training Neural Networks Apr 25, 2019)

# Beyond feed forward networks

# Recurrent Neural Networks

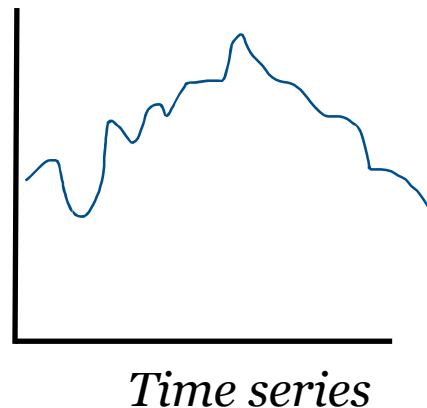
Feed forward networks are not made for **sequential data**

time series, e.g., financial data

speech recognition

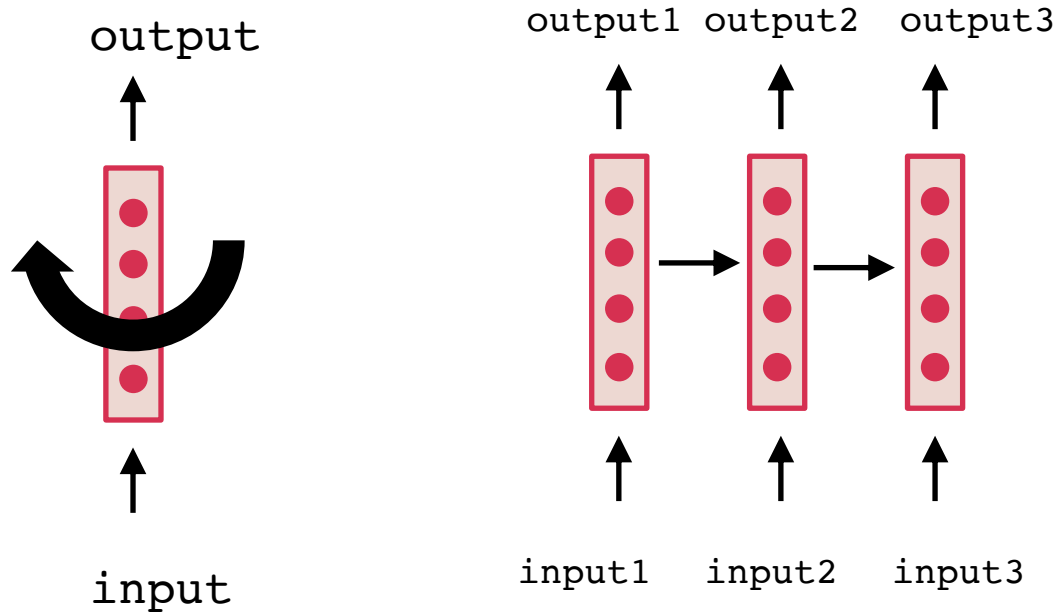
handwriting recognition

language (e.g. sentences)



This movie was not great

# Recurrent Neural Networks

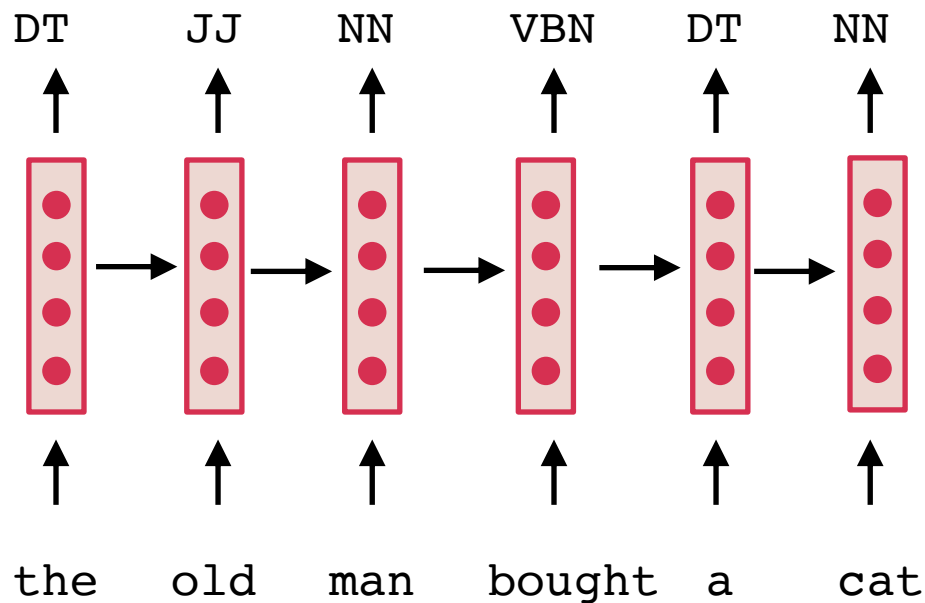


Take sequential input.  
Apply the same weights  
on each time step.

Can handle sequences of  
arbitrary lengths

*Prediction depends on the  
results for previous  
elements of the sequence*

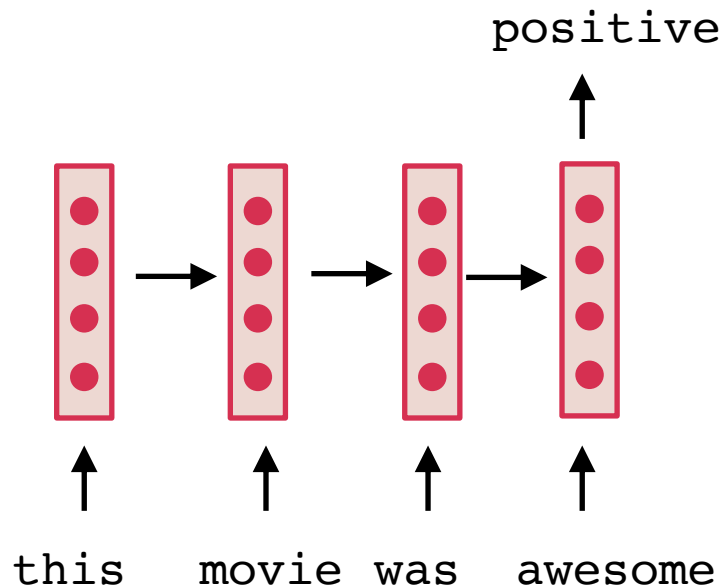
# Recurrent Neural Networks



Take sequential input.  
Apply the same weights  
on each time step.

*RNNs for sequence tagging  
(e.g. POS tagging)*

# Recurrent Neural Networks

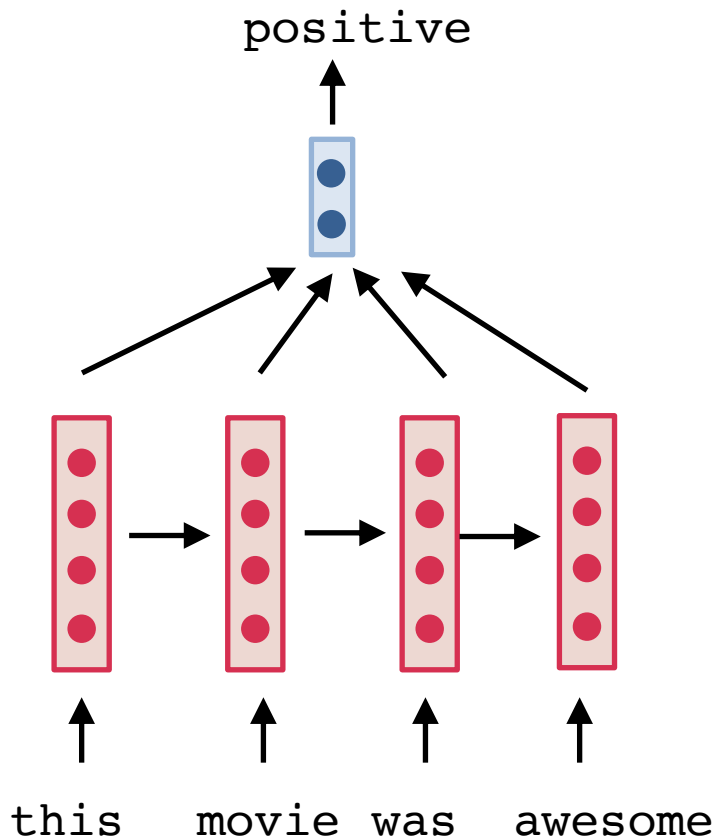


Take sequential input.  
Apply the same weights  
on each time step.

*RNNs for classification  
(e.g. sentiment analysis).  
Use the final hidden state.*



# Recurrent Neural Networks

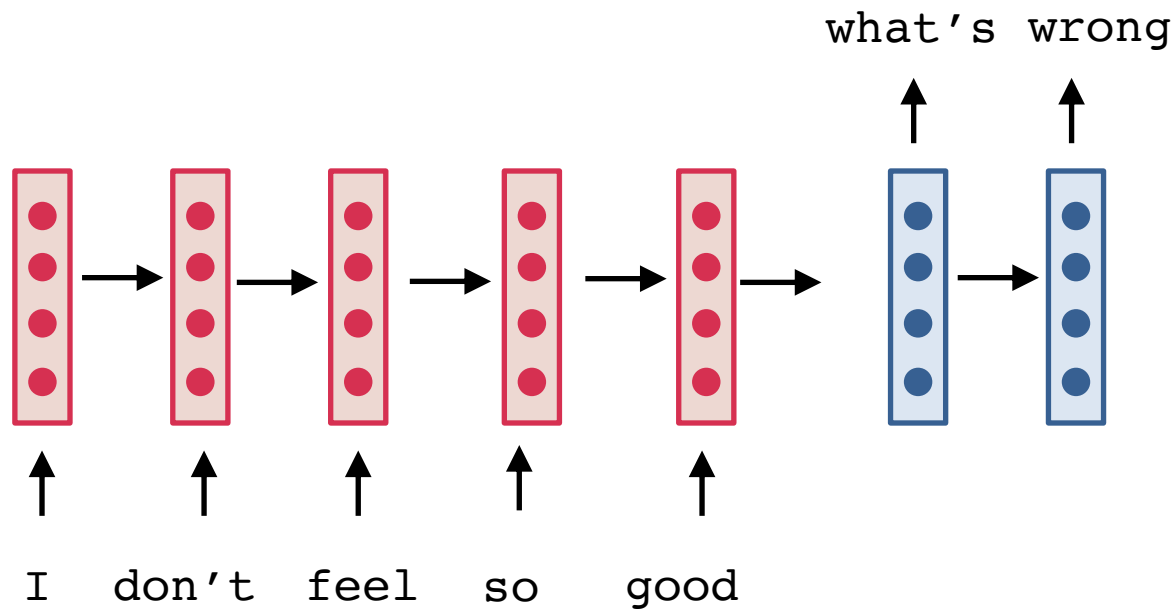


Take sequential input.  
Apply the same weights  
on each time step.

*RNNs for classification  
(e.g. sentiment analysis).*

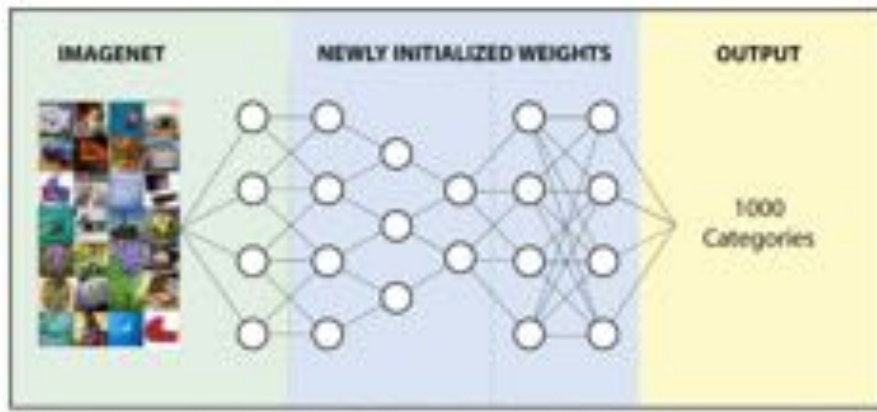
*Alternative: Aggregate all  
hidden states (e.g. mean or  
max)*

# Recurrent Neural Networks

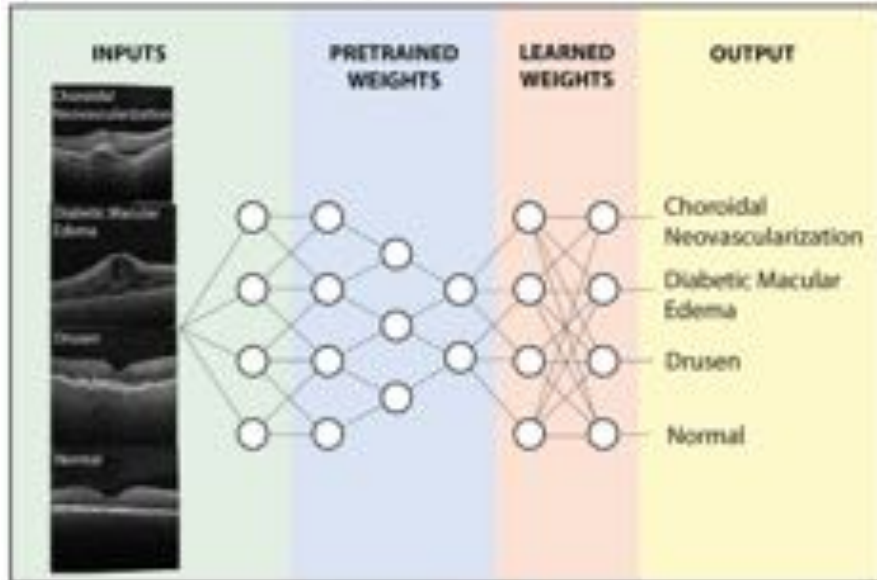


*RNNs for text  
generation  
(e.g. dialogue  
systems!)*

*Also called sequence-  
to-sequence networks  
(seq2seq)*



↓  
**TRANSFER  
LEARNING**



## Transfer learning

Train a model on a large dataset (e.g. Imagenet). Retrain part of the model for a task with less data.

[Image from Kermany et al.,  
Cell 2018]

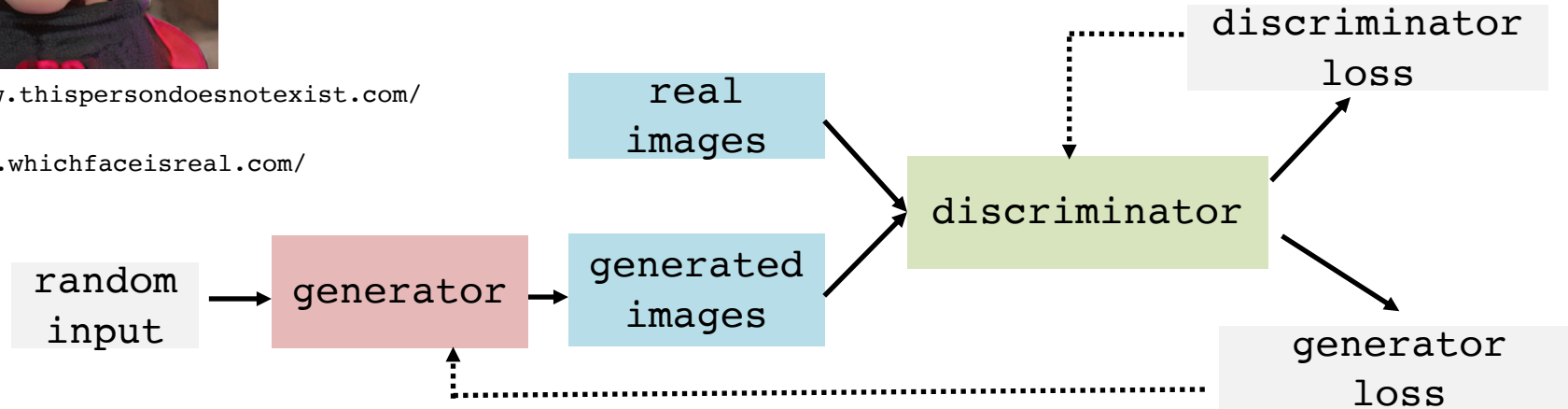
# Generative Adversarial Network (GAN)



<https://www.thispersondoesnotexist.com/>

<http://www.whichfaceisreal.com/>

The **generator** learns to generate data.  
The **discriminator** learns to distinguish  
the generated data from real data.



GANs: Goodfellow et al. 2014

# Neural networks: pros and cons

- Can learn complex non-linear hypotheses
- Various types of architectures (e.g. for sequential series, adversarial networks).
- More difficult to interpret (but this is an active area of research!)
- Requires lots of data to train (but ways to mitigate this are for example transfer learning)
- Training neural networks is sometimes seen as ‘black magic’, many tricks involved!
- Deep neural networks can be *very* computationally expensive



# You should know

- What linear regression is
- What a loss function is
- What logistic regression is (e.g. sigmoid, decision boundary, cross-entropy, gradient descent for logistic regression, regularization)
- The main idea of neural networks (the types of activation functions, their relation to logistic regression, strengths compared to classifiers like logistic regression, ways to prevent overfitting)

# Libraries

- Keras <https://keras.io/> (friendly wrapper around TensorFlow)
- PyTorch <https://pytorch.org/>
- TensorFlow <https://www.tensorflow.org>

# Thanks

Some slides based on (or inspired by) slides by  
Matt Gormley, Andrew Ng and Marijn Schraagen