Machine learning advanced
Methods in AI research

Dong Nguyen
Sept 2020

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

$\{<x^{(1)}, y^{(1)}>, \ldots, <x^{(N)}, y^{(N)}>\}$

Goal: Predict the target using the features

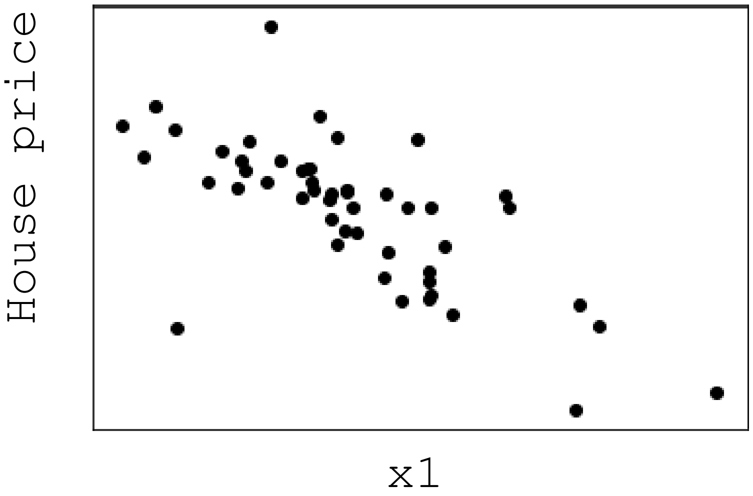
Need to define **features**, characteristics of the instances that the model uses for predictions (words in a document, movie ratings, etc..)

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

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

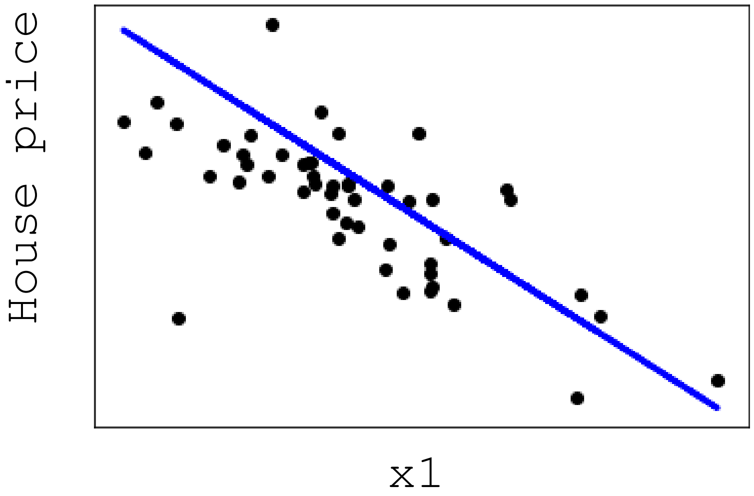
Goal: Predict the target using the features

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

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

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

Linear regression



features target

$\{<x^{(1)}, y^{(1)}>, \ldots, <x^{(N)}, y^{(N)}>\}$

Goal: Predict the target using the features

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

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

$x_j^{(i)}$: the j^{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:

$$\overset{}{y} = \underbrace{b}_{\text{bias}} + \underbrace{w_1}_{\text{weights}} x_1 + \ldots + w_d x_d$$

$$= b + \sum w_i x_i = b + w \cdot x$$

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

This is a **linear model**.

features target

$\{<x^{(1)}, y^{(1)}>, \ldots, <x^{(N)}, y^{(N)}>\}$

Goal: Predict the target using the features

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

Notation:
Each instance $x^{(i)}$ has d features: $[x_{1}, \ldots, x_d]$

$x_j^{(i)}$: the j^{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:

bias weights

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

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

This is a **linear model**.

features target

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

Goal: Predict the target using the features

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

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

$x_j^{(i)}$: the j^{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:

$$y = \overset{\text{bias}}{b} + \underbrace{w_1 x_1 + \ldots + w_d x_d}_{\text{weights}}$$

$$= b + \sum w_i x_i = b + w \cdot x$$

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

This is a **linear model**.

feature	w_i	x_i
number of bedrooms	30k	2
has garden	25k	0

bias term = 250k

predicted house price:
250 + 2 * 30 + 0 * 25 = 310k

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:

$$y = \underbrace{b}_{\text{bias}} + w_1 x_1 + \ldots + w_d x_d$$

weights

$$= b + \sum w_i x_i = b + w \cdot x$$

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

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

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

$\{<x^{(1)}, y^{(1)}>, \ldots, <x^{(N)}, y^{(N)}>\}$

Goal: Predict the target using the features

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

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

$x_j^{(i)}$: the j^{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-looooo00.com
```

Spam or not?

features target

$$\{<x^{(1)}, y^{(1)}>, \ldots, <x^{(N)}, y^{(N)}>\}$$

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: $[x_1, \ldots, x_d]$

$x_j^{(i)}$: the j^{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:

$$z = \underbrace{b}_{\text{bias}} + \underbrace{w_1}_{\text{weights}} x_1 + \ldots + w_d x_d$$

$$= b + \sum w_i x_i = b + w \cdot x$$

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

$\{<x^{(1)}, y^{(1)}>, \ldots, <x^{(N)}, y^{(N)}>\}$

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: $[x_{1,} \ldots, x_d]$

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

Modeling the output

Logistic regression output:

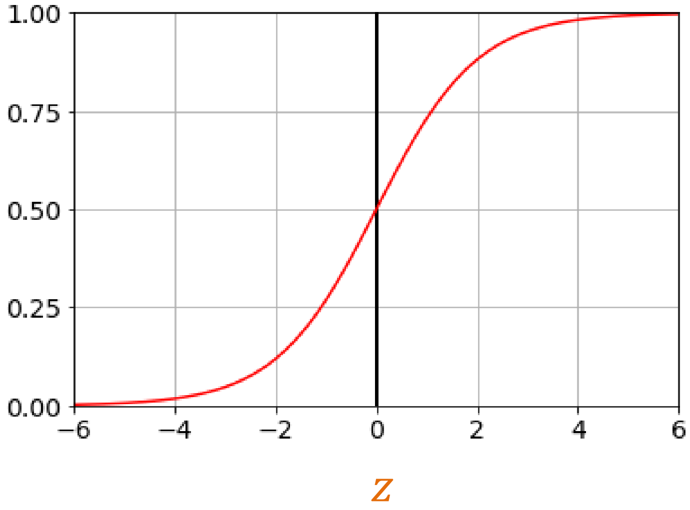
We want: 0 <= output <= 1.

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

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

sigmoid function

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



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:

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

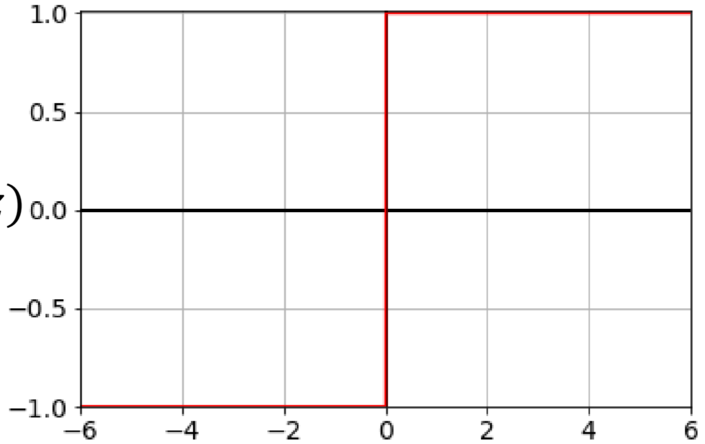
Sigmoid (or logistic) function

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

Aside: why not use the sign function?

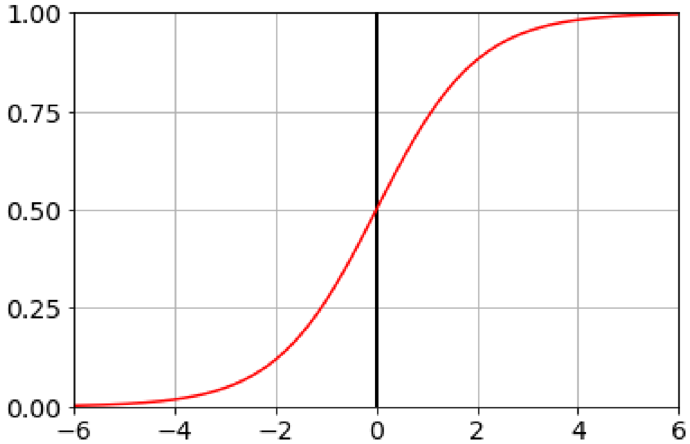
sign function

$sign(z)$



sigmoid function

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



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 $P(y=1|x) \geq 0.5$

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

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 $P(y=1|x) = 0.51$

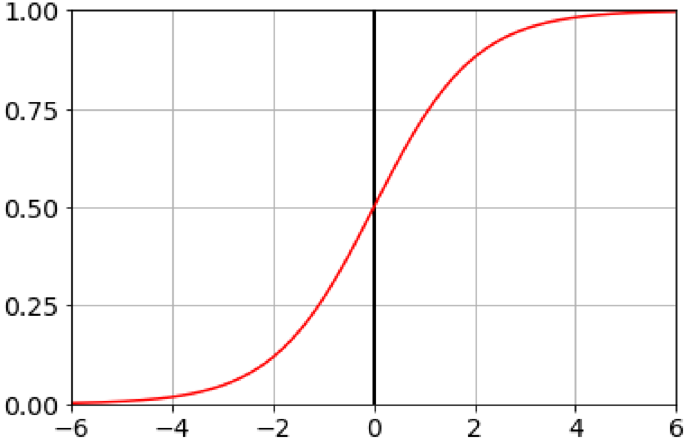
- Probability can be used for predicting a *class*.
  - For example, predict 1 when $P(y=1|x) \geq 0.5$

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

Precision goes up,
recall goes down

Decision boundary

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



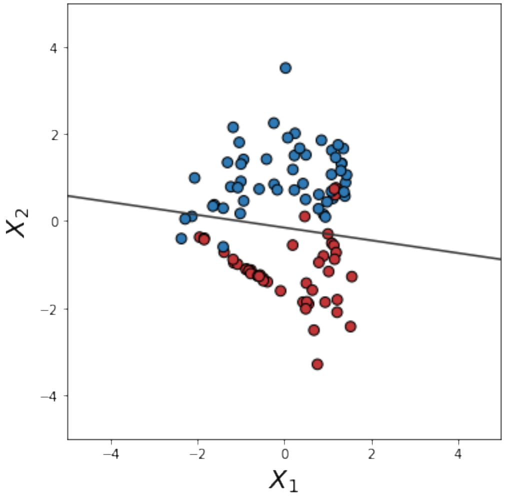
Predict 1,
When $p(y=1|x) \geq 0.5$
Is same as when $z \geq 0$

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

$$z = b + w \cdot x$$

Linear classification rule!

Decision boundaries

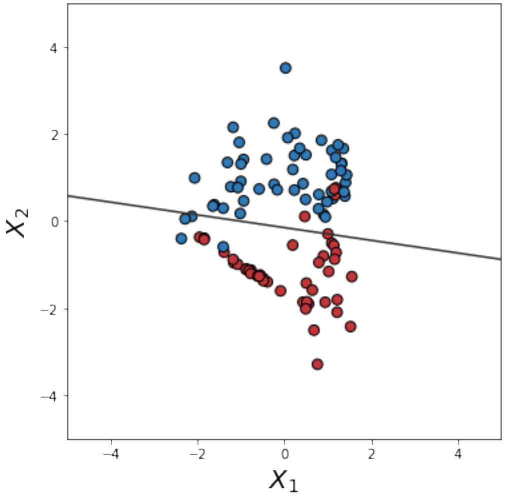


b = 0.37
w_1 = 0.35
w_2 = 2.41

Logistic regression is a linear classifier!

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

Decision boundaries



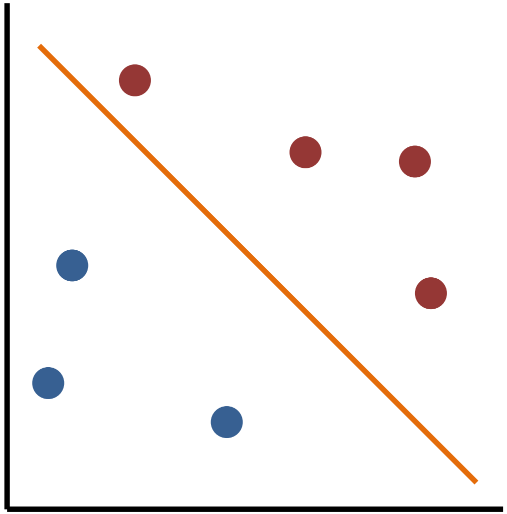
b = 0.37
w_1 = 0.35
w_2 = 2.41

Logistic regression is a linear classifier!

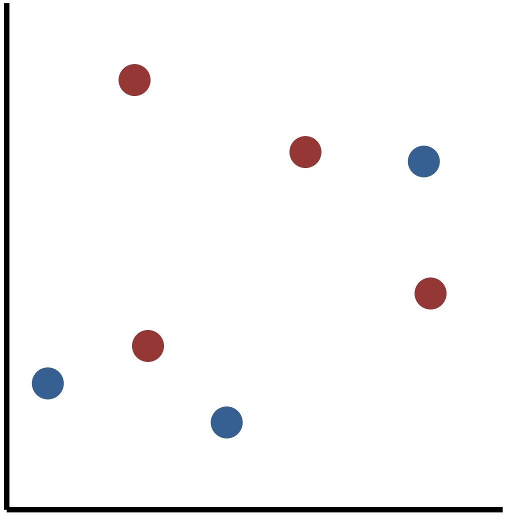
Question: Are decision trees linear classifiers?
Are nearest-neighbor models linear classifiers?

Both are not linear classifiers

Linearly separable?



Yes!



No!

Logistic regression: Example

feature	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

Will the user click on the advertisement?

b=-1

Logistic regression: Example

feature	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:

$$z = \underbrace{b}_{\text{bias}} + \underbrace{w_1}_{\text{weights}} x_1 + \ldots + w_d x_d$$

$$= b + \sum w_i x_i = b + w \cdot x$$

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

features target

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

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: $[x_1, \ldots, x_d]$

$x_j^{(i)}$: the j^{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:

bias weights

$$z = b + w_1 x_1 + \ldots + w_d x_d$$

$$= b + \sum w_i x_i = b + w \cdot x$$

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

How do we learn the weights w and b?

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

features target

$\{<x^{(1)}, y^{(1)}>, \ldots, <x^{(N)}, y^{(N)}>\}$

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: $[x_1, \ldots, x_d]$

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

Learning the parameters

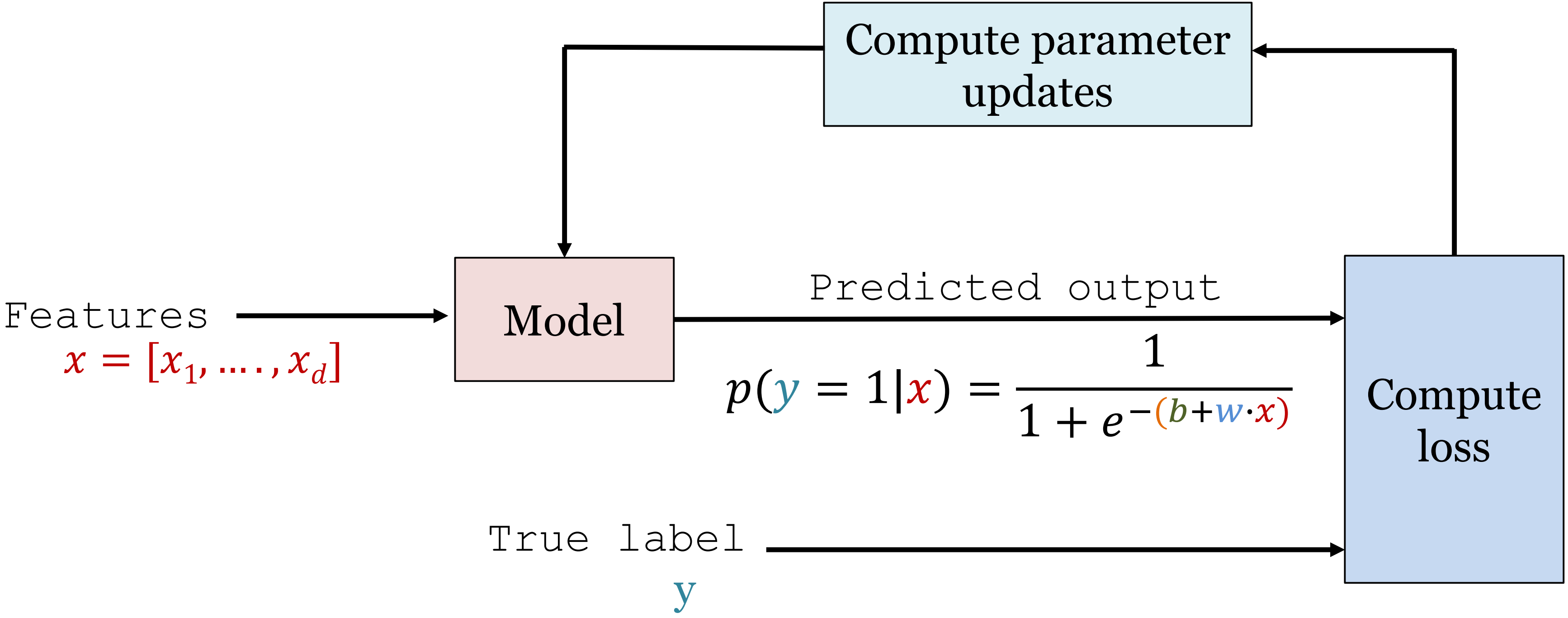


Figure based on https://developers.google.com/machine-learning/crash-course/reducing-loss/an-iterative-approach

Loss function

Notation:
y = true label
$\hat{y}$ =classifier output
$= P(y=1|x;\, \boldsymbol{\theta})$
$= \sigma(w \cdot x + b)$

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

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

Loss function

Notation:
y = true label
$\hat{y}$ =classifier output
 = P(y=1|x; $\boldsymbol{\theta}$)
 = $\sigma(w \cdot x + b)$

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

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

Trick, combine this into one equation!

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

Loss function

Notation:
y = true label
$\hat{y}$ =classifier output
$= P(y=1 | x; \boldsymbol{\theta})$
$= \sigma(w \cdot x + b)$

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

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

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

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

Loss function

Notation:
y = true label
$\hat{y}$ =classifier output
$= P(y{=}1 \mid x;\ \boldsymbol{\theta})$
$= \sigma(w \cdot x + b)$

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

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

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

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

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

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

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

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

"How much does the classifier output differ from the correct output?"

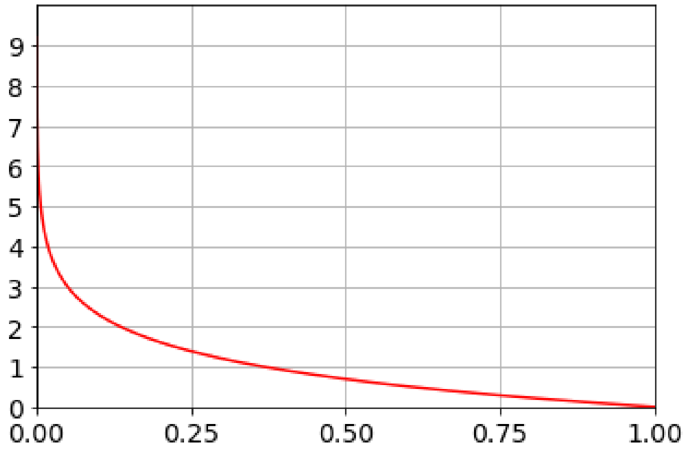
Loss function

Notation:
y = true label
$\hat{y}$ =classifier output
= P(y=1|x; $\boldsymbol{\theta}$)
= $\sigma(w \cdot x + b)$

Cross-entropy loss = L($\hat{y}$, y)
= - log p(y|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 $\hat{y}$



Aside: cross-entropy

x	p(x)	q(x)	s(x)
A	0.1	0.2	0.6
B	0.8	0.6	0.1
C	0.1	0.2	0.3

$$H(p,q) = -\sum p(x)\log(q(x))$$

```
H(p,q) = -0.1 * ln(0.2) – 0.8 * ln(0.6) – 0.1 * ln(0.2) = 0.731
```

```
H(s,q) = 1.50
```

How to compare two probability distributions?

Aside: cross-entropy

when calculating the loss, in practice both base 2 and base e (ln) is used

x	p(x)	q(x)	s(x)
A	0.1	0.2	0.6
B	0.8	0.6	0.1
C	0.1	0.2	0.3

$$H(p,q) = -\sum p(x)\log(q(x))$$

```
H(p,q) = -0.1 * ln(0.2) -
0.8 * ln(0.6) - 0.1 *
ln(0.2) = 0.731
```

How to compare two probability distributions?

```
H(s,q) = 1.50
```

Aside: cross-entropy

Class	True label	Classifier A
A	0	0.1
B	1	0.8
C	0	0.1

$$H(p,q) = -\sum p(x)\log(q(x))$$

loss classifier A

`-1 * ln(0.8)` = 0.223

Aside: cross-entropy

Class	True label	Classifier A	Classifier B
A	0	0.1	0.8
B	1	0.8	0.1
C	0	0.1	0.1

$$H(p,q) = -\sum p(x)\log(q(x))$$

loss classifier A

`-1 * ln(0.8)` = 0.223

loss classifier B

`-1 * ln(0.1)` = 2.303

Loss function

```
Recall:
ŷ = 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:

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

Gradient descent

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

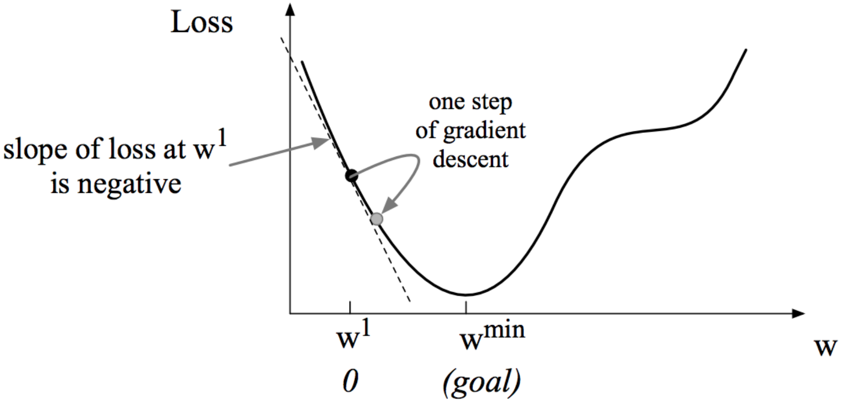
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_i \mathrm{L}(\hat{y}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

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 descent

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

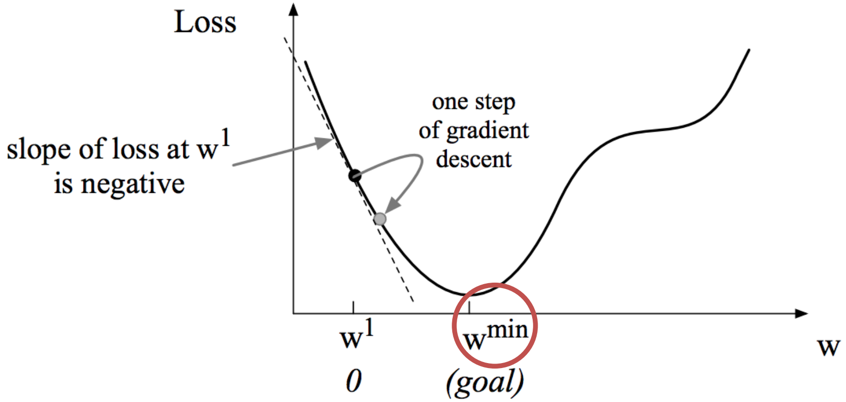
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_i \mathrm{L}(\hat{y}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

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]

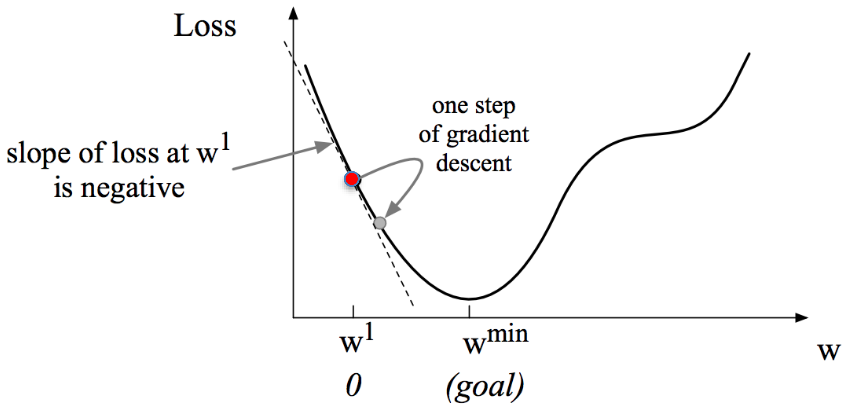
Gradient descent

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

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

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

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



[J&M, chapter 5, Fig 5.3]

$$\underbrace{w^{t+1}}_{\text{next step}} = \underbrace{w^t}_{\text{current step}} - \underbrace{\eta}_{\text{learning rate}} \underbrace{\frac{d}{dw} f(x; w)}_{\text{slope}}$$

Gradient descent

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

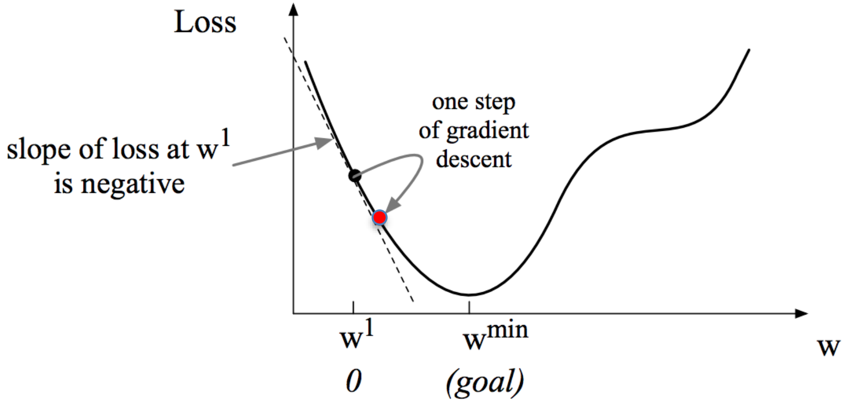
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_i \mathrm{L}(\hat{y}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

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 descent

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

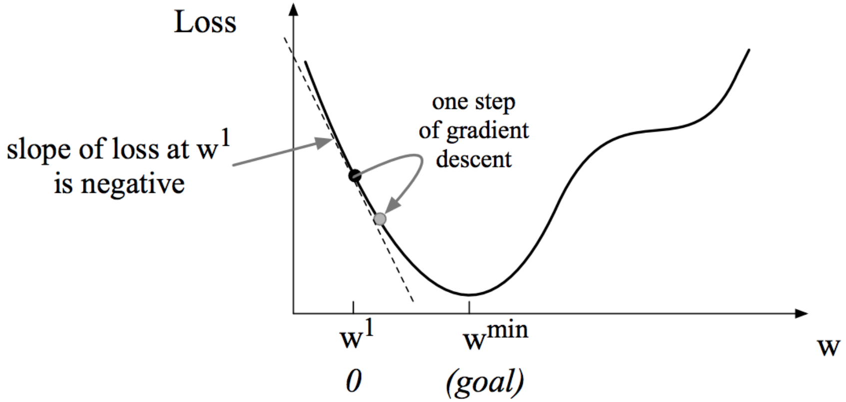
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_i \mathrm{L}(\hat{y}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

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]

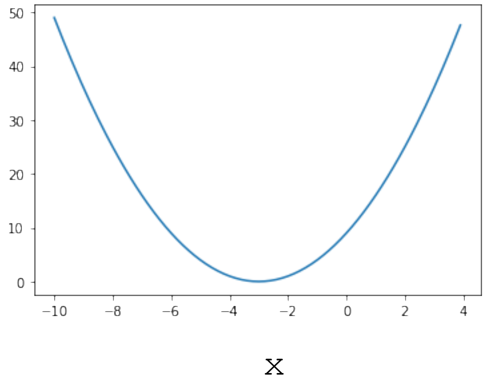
Gradient is a multi-variable generalization of the slope!

Gradient descent example

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

next step current step learning rate slope

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



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

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

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

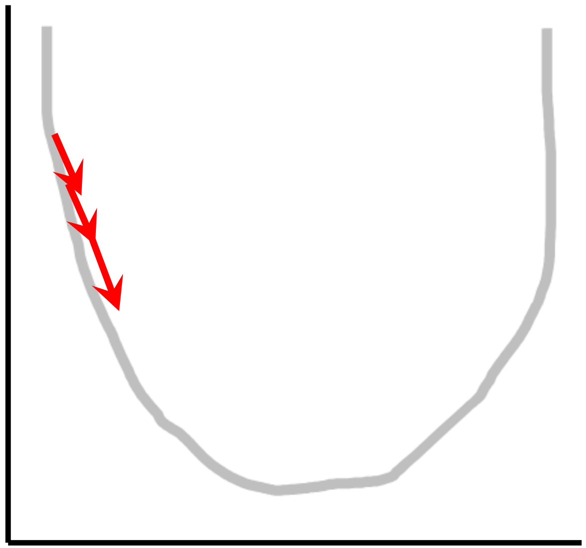
Converges to -3!

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.

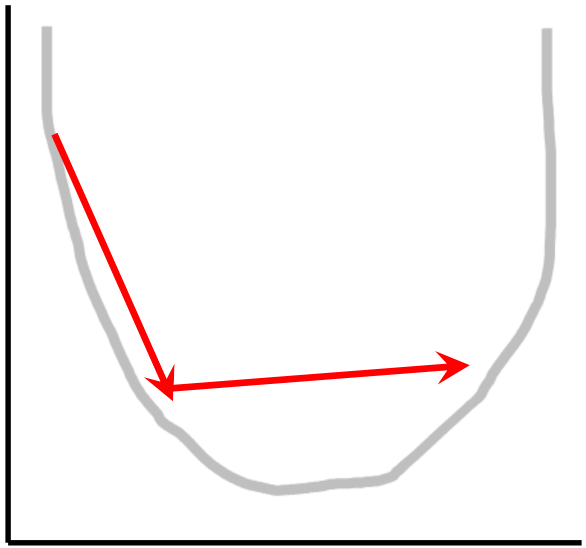


Gradient descent: learning rate

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 $\boldsymbol{\theta} = w, b$ that minimizes this loss

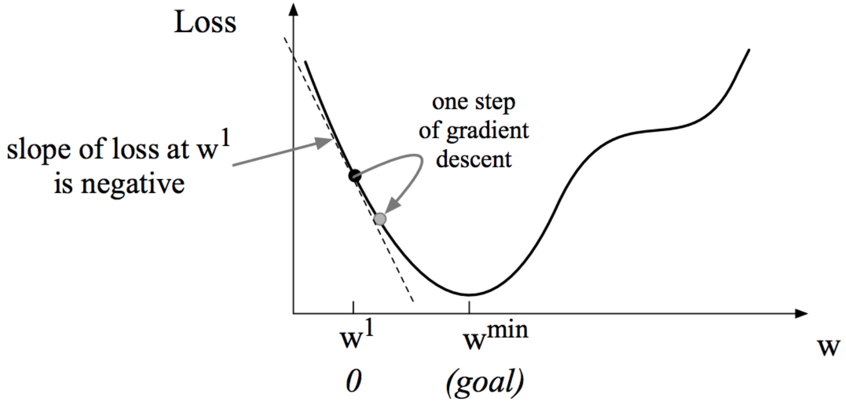
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum \mathrm{L}(\hat{y},\ y;\ \boldsymbol{\theta})$$

Gradient is a multi-variable generalization of the slope.

$$\nabla_{\boldsymbol{\theta}} \mathrm{L}(\hat{y},\ y;\ \boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial w1} & \mathrm{L}(\hat{y},\ y;\ \theta) \\ \frac{\partial}{\partial w2} & \mathrm{L}(\hat{y},\ y;\ \theta \\ \dots & \dots. \end{bmatrix}$$

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

next step current step learning rate gradient



[J&M, chapter 5, Fig 5.3]

Gradient logistic regression

Recall:
$\hat{y}$ = classifier output
y = true label
log(a^b) = b log(a)

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

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

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

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

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** θ
    # where: L is the loss function
    #   f is a function parameterized by θ
    #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$
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 θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
return θ

[J&M, chapter 5, Fig 5.5]

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** θ
    # where: L is the loss function
    #   f is a function parameterized by θ
    #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$
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 θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
return θ

[J&M, chapter 5, Fig 5.5]

Regularization

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

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum \underbrace{\mathrm{L}(\hat{y},\ \mathrm{y}; \boldsymbol{\theta})}_{\text{loss}} + \boldsymbol{\lambda} \underbrace{\mathrm{R}(\boldsymbol{\theta})}_{\text{model complexity}}$$

RECAP!

The L2 norm:

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

The L1 norm:

$$\|\boldsymbol{a}\|_1 = \sum |a_i|$$

Regularization

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

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \underbrace{\frac{1}{N}\sum \mathrm{L}(\hat{y},\ y;\boldsymbol{\theta})}_{\text{loss}} + \boldsymbol{\lambda}\,\underbrace{\mathrm{R}(\boldsymbol{\theta})}_{\text{model complexity}}$$

RECAP!

The L2 norm:

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

The L1 norm:

$$\|\boldsymbol{a}\|_1 = \sum |a_i|$$

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

$\boldsymbol{\theta}$ = [0.1, 0.25, 0.05], R($\boldsymbol{\theta}$) = $0.1^2 + 0.25^2 + 0.05^2 = 0.075$

Regularization

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

hyper parameter

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

loss

model complexity

RECAP!

The L2 norm:

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

The L1 norm:

$$\|\boldsymbol{a}\|_1 = \sum |a_i|$$

L2 regularization (or, ridge regularization): $\mathrm{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum \boldsymbol{\theta}_i^2$
(the square of the L2 norm of the weight values)

L1 regularization (or, lasso regularization): $\mathrm{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum |\boldsymbol{\theta}_i|$

Regularization

Question: We can't set the regularization parameter $\boldsymbol{\lambda}$ 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 $\boldsymbol{\theta}$ =w, b that minimizes the loss. We now add a regularization term (R($\boldsymbol{\theta}$))

hyper parameter

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

loss

model complexity

RECAP!

The L2 norm:

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

The L1 norm:

$$\|\boldsymbol{a}\|_1 = \sum |a_i|$$

L2 regularization (or, ridge regularization): $\mathrm{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum \boldsymbol{\theta}_i^2$
(the square of the L2 norm of the weight values)

L1 regularization (or, lasso regularization): $\mathrm{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum |\boldsymbol{\theta}_i|$

Multiclass classification

Recall: the **sigmoid**.

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

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

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

Input vector `z = [z₁, z₂, ...zₖ]` → `[softmax(z₁), softmax(z₂),.., .softmax(zₖ)]`

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

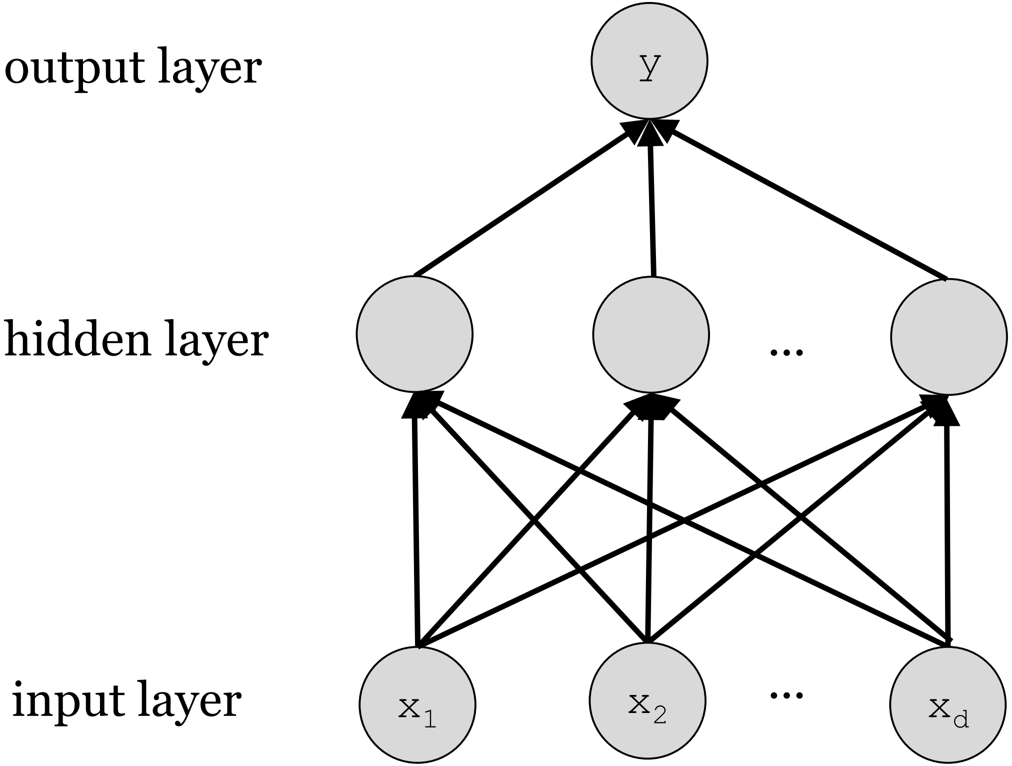
Have been around for a *long time*:

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

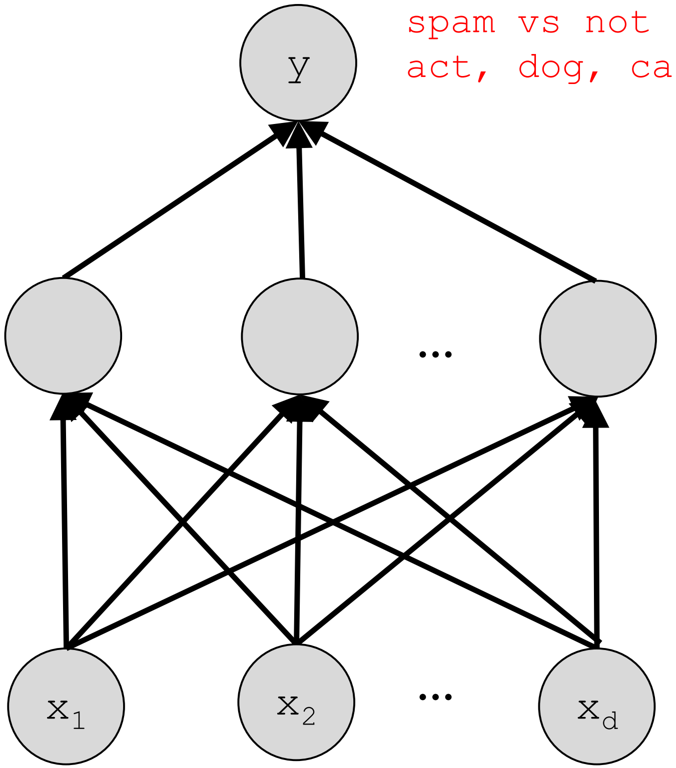
A simple neural network

output layer

spam vs not spam, dialog act, dog, cat, ..

hidden layer

input layer

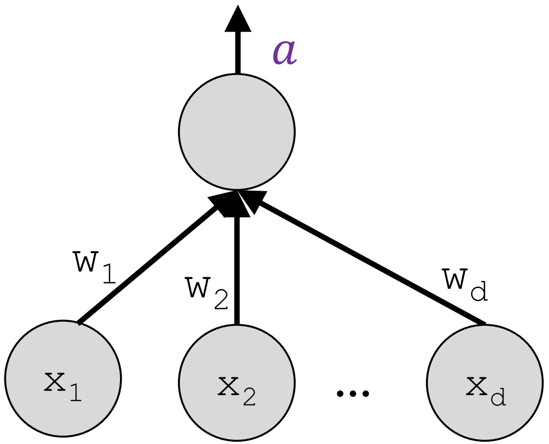


words, pixels, ...

- 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

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

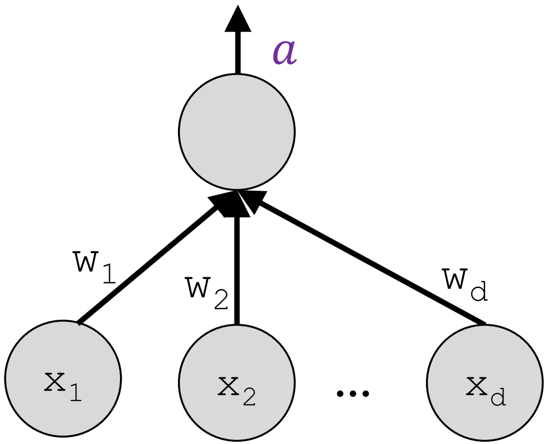


Neural units apply a **non-linear activation function** *f* to z, resulting in an **activation** value

$$a = f(z)$$

Building blocks of neural nets: units

$$z = b + w_1 x_1 + \ ... + w_d x_d$$
$$= \ b + \sum w_i x_i = b + w \cdot x$$



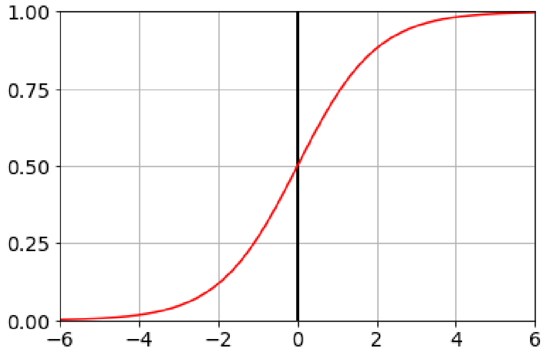
Neural units apply a **non-linear activation function** f to z, resulting in an **activation** value

$$a = f(z)$$

Usually used for output layer (binary classification)

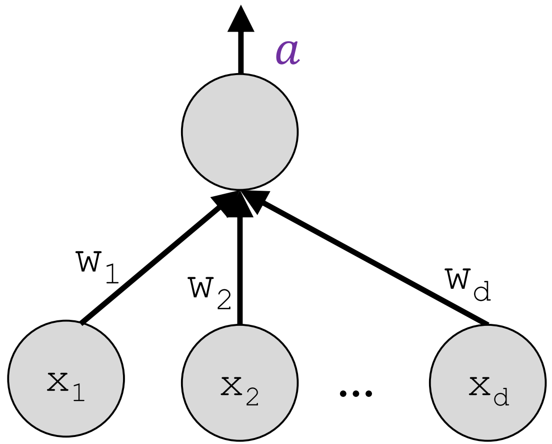
sigmoid

This should look familiar! (logistic regression)



Building blocks of neural nets: units

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

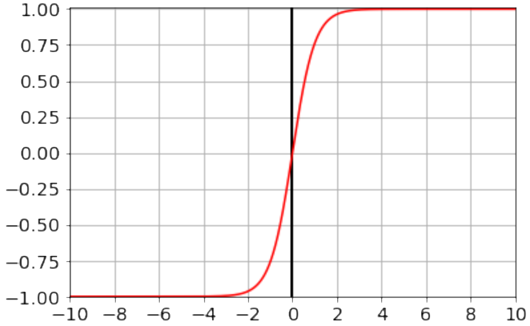


Neural units apply a **non-linear activation function** f to z, resulting in an **activation** value

$$a = f(z)$$

tanh

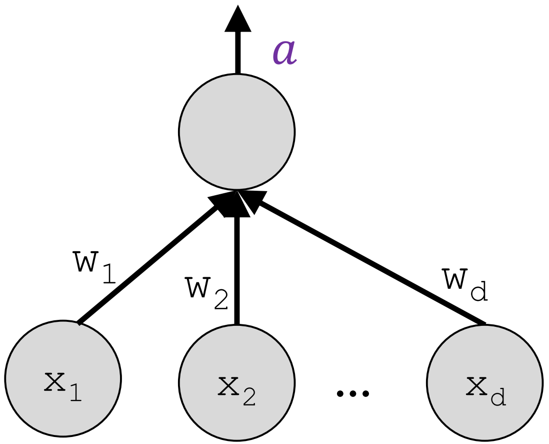
$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Usually used for hidden layers

Building blocks of neural nets: units

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



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)

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

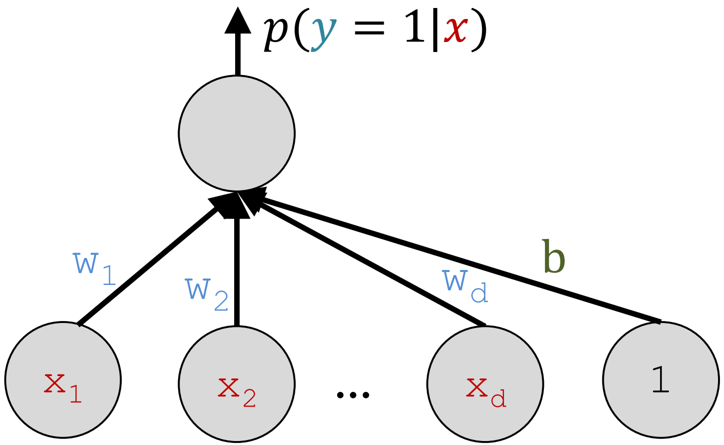
RECAP!

Logistic Regression

Logistic regression:

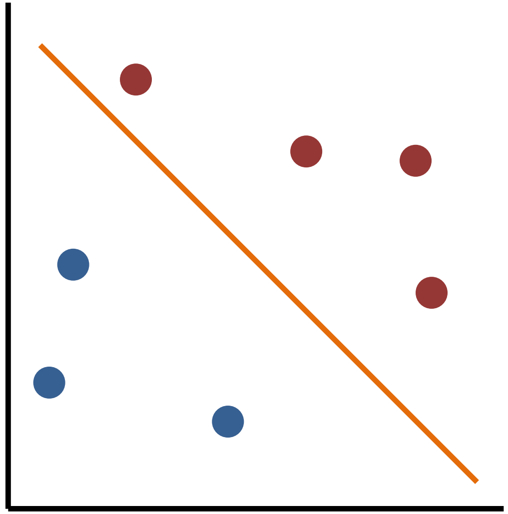
$$p(y = 1|x) = \frac{1}{1 + e^{-z}} \qquad \text{with} \quad z = b + w \cdot x$$

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

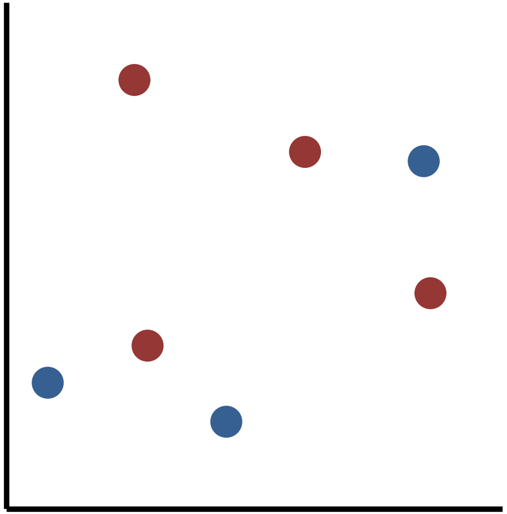


RECAP!

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

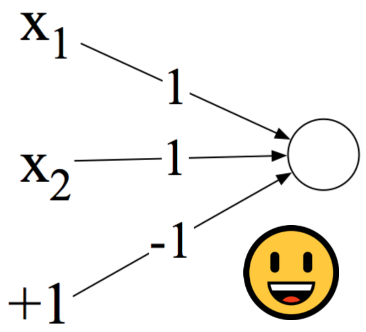
AND

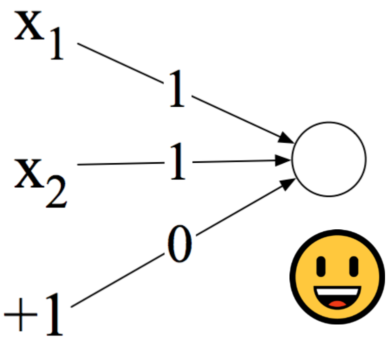
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

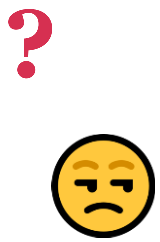
OR

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

XOR







Perceptron
(no non-linear activation)
0, if $w \bullet x + b \leq 0$
1, if $w \bullet x + b > 0$

[J&M, Fig. 7.4]

XOR example

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

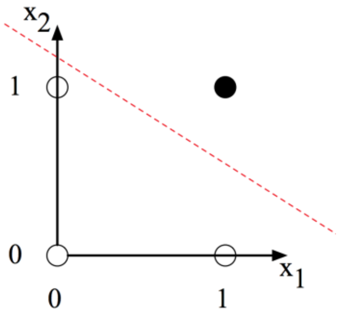
AND

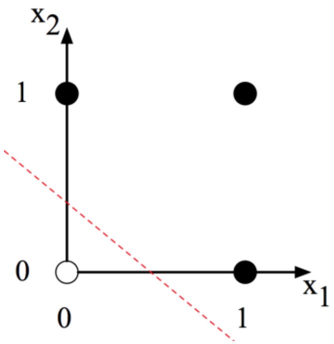
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

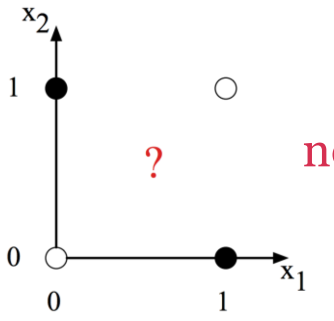
OR

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

XOR





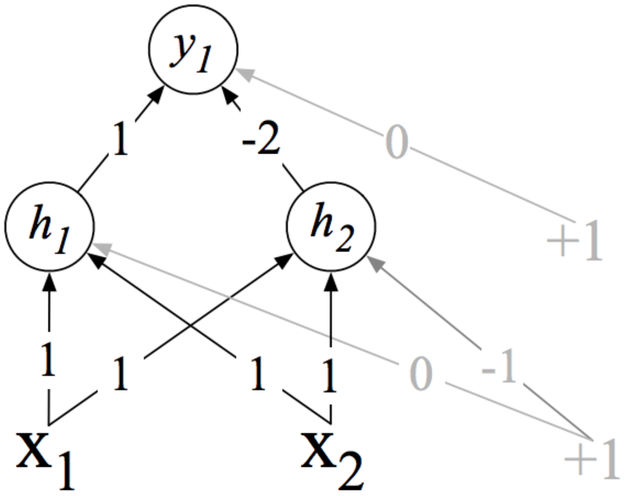


XOR:
not linearly separable!

[J&M, Fig. 7.5]

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

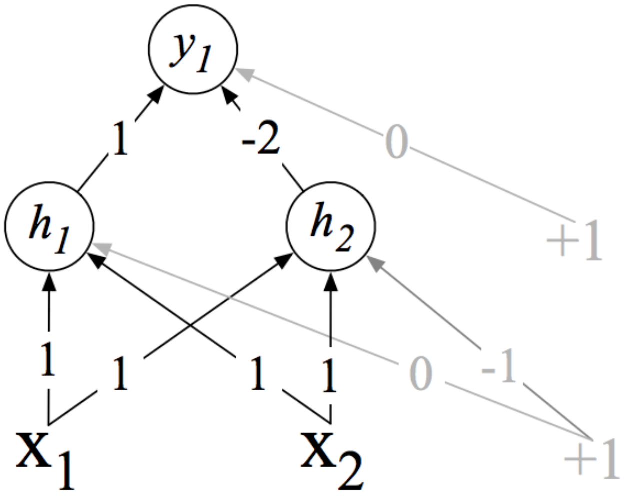


[J&M, Fig. 7.6,
based on Goodfellow et al. 2016]

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

XOR network

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



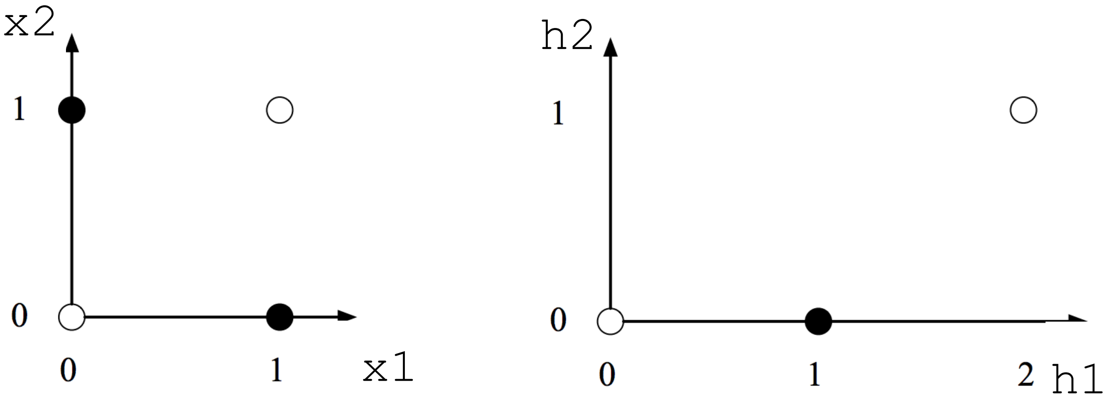
[J&M, Fig. 7.6,
based on Goodfellow et al. 2016]

h1 = max(0, 0*1 + 0 * 1 + 1 * 0) = 0
h2 = max(0, 0*1 + 0 * 1 + 1 * -1) = 0

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

XOR network: Learning representations

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



a) The original *x* space

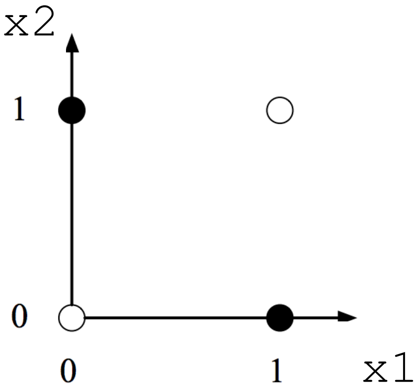
b) The new *h* space

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

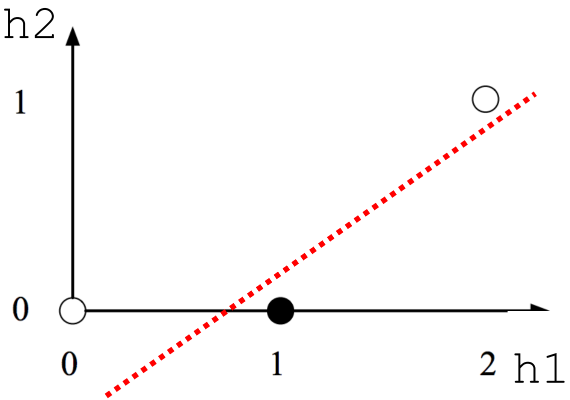
Question: Is the new *h* space linearly separable?

XOR network: Learning representations

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



a) The original *x* space



b) The new *h* space

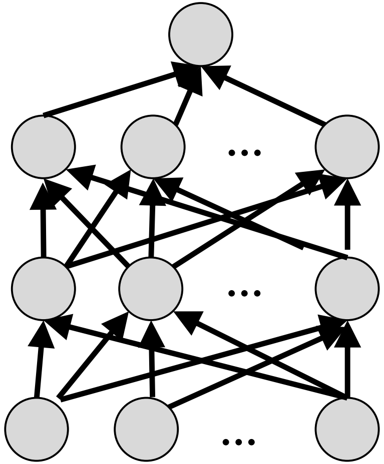
Question: Is the new *h* space linearly separable?

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

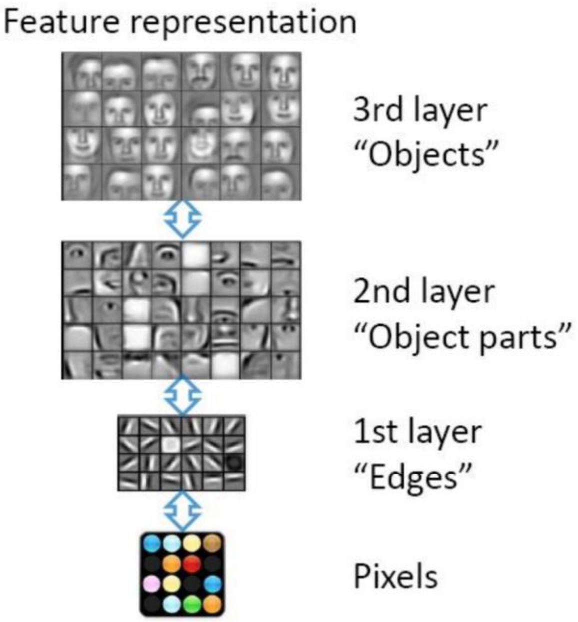
Learning representations

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

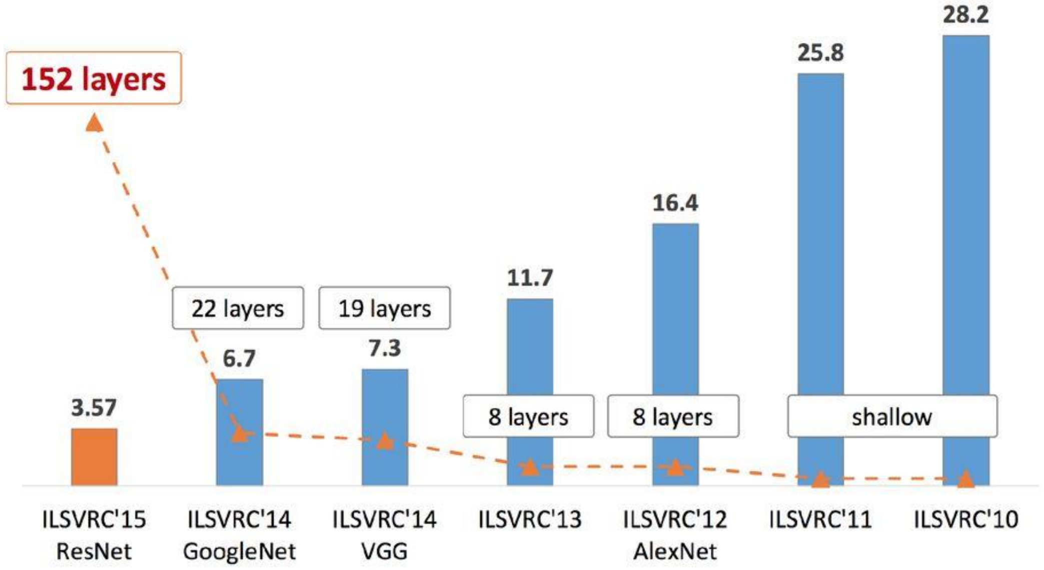


https://deeplearningworkshopnips2010.files.wordpress.com/2010/09/nips10-workshop-tutorial-final.pdf

Deep neural networks

Deep neural networks have **many** layers

ImageNet experiments



ImageNet Classification top-5 error (%)

18

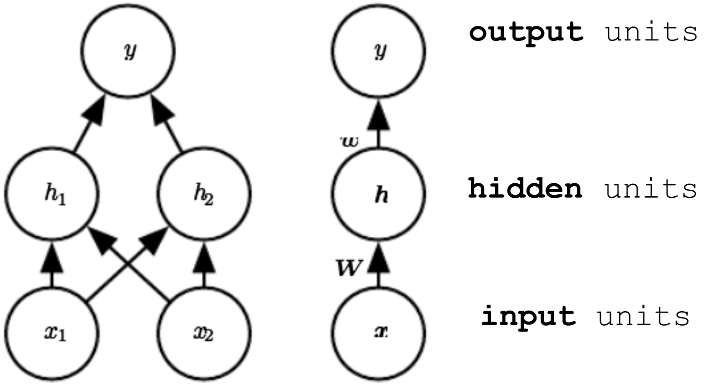
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Feed forward network

A **feed-forward network**:

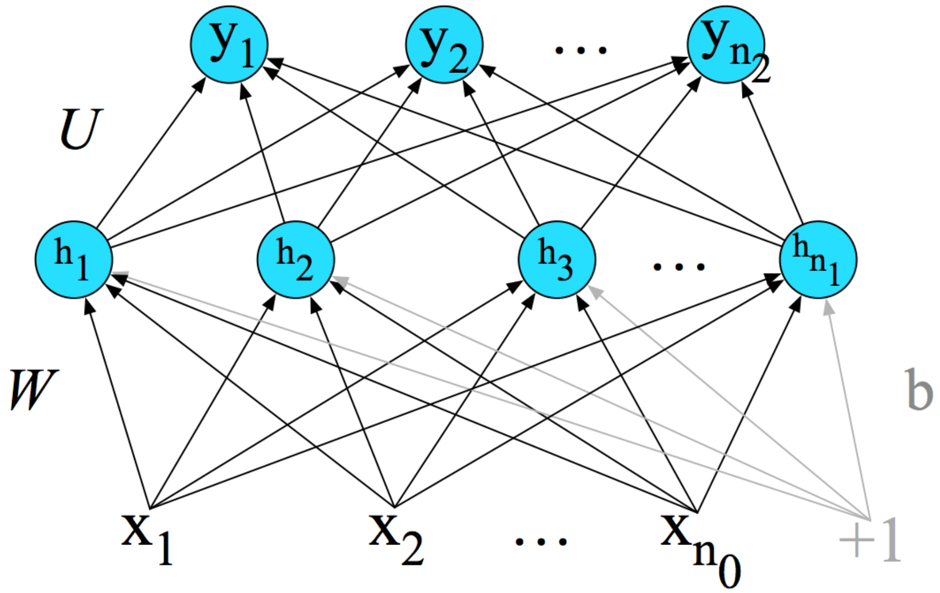
- A multilayer network
- Units are connected but no cycles
- The output from units in each layer are passed to units in the next layer, no output passed back to lower layers

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



http://www.deeplearningbook.org/contents/mlp.html

Feed forward network



[J&M, Fig. 7.8]

Forward propagation

Back propagation

Matrices

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

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

$\mathbf{B} \in \mathbb{R}^{2x3}$

$\mathbf{H} \in \mathbb{R}^{mxn}$

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

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

Vectors:

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

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

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

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

See also:

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

Matrices

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

$\mathbf{B} \in \mathbb{R}^{2x3}$

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

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

$\mathbf{H} \in \mathbb{R}^{mxn}$

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

Vectors:

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

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

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

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

See also:

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

Matrices

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

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

$\mathbf{B} \in \mathbb{R}^{2x3}$

$\mathbf{H} \in \mathbb{R}^{mxn}$

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

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

Vectors:

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

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

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

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

See also:

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

Matrices

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

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

$\mathbf{B} \in \mathbb{R}^{2x3}$

$\mathbf{H} \in \mathbb{R}^{mxn}$

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

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

Vectors:

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

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

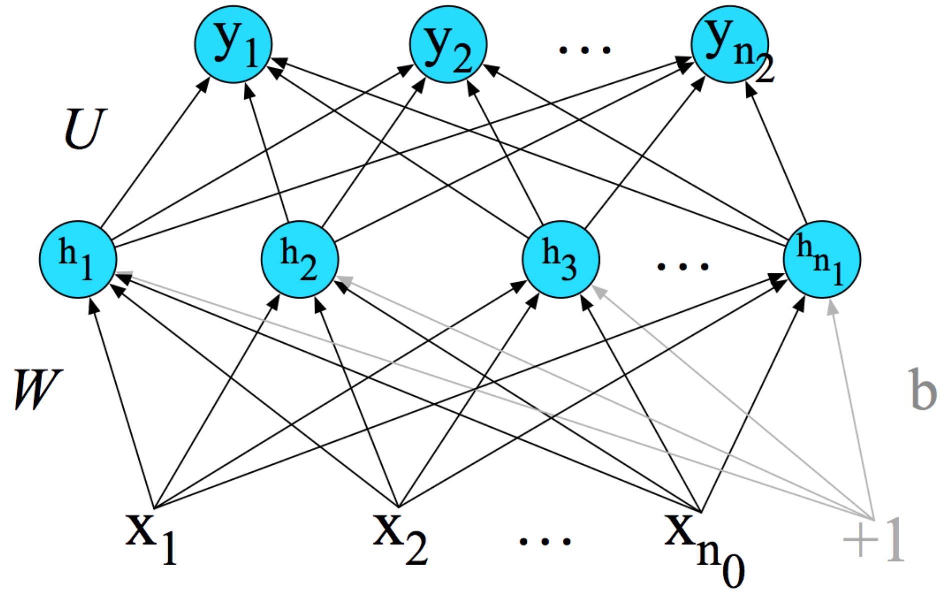
$\mathbf{c} = [c_1, \ldots, c_d]$

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

See also:

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

Feed forward network: forward propagation



[J&M, Fig. 7.8]

$x \in \mathbb{R}^{n0}$ $b \in \mathbb{R}^{n1}$

$W \in \mathbb{R}^{n1 \times n0}$ $h \in \mathbb{R}^{n1}$

Recall: one single hidden unit:

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

For an entire hidden layer:

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

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

Etc..

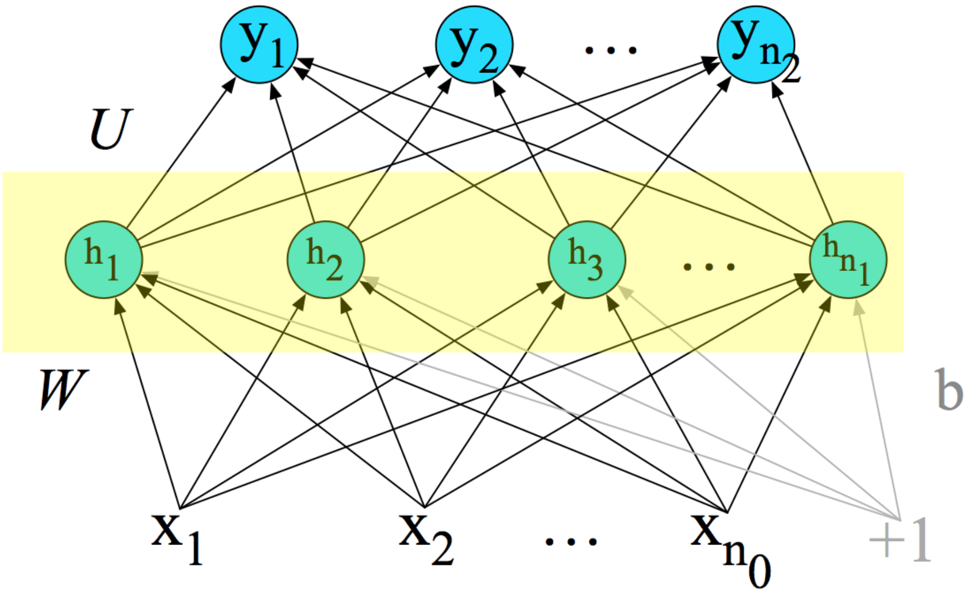
W_{ij} the weight of the connection between h_i and x_j

Using matrix operations:

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

e.g. sigmoid or ReLU

Feed forward network: forward propagation



[J&M, Fig. 7.8]

$x \in \mathbb{R}^{n0}$ $\quad$ $b \in \mathbb{R}^{n1}$

$W \in \mathbb{R}^{n1 \times n0}$ $\quad$ $h \in \mathbb{R}^{n1}$

Recall: one single hidden unit:

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

For an entire hidden layer:

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

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

Etc..

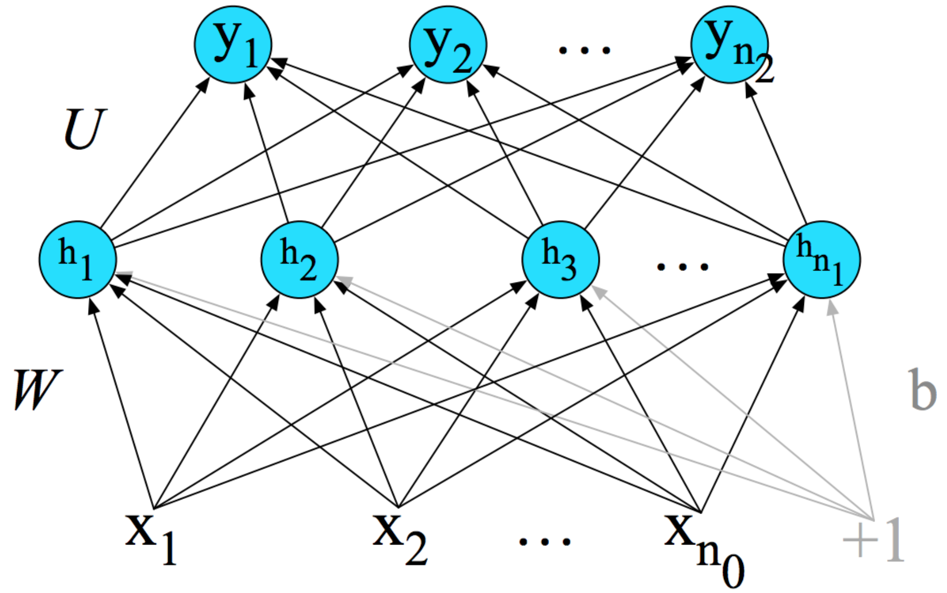
W_{ij} the weight of the connection between h_i and x_j

Using matrix operations:

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

e.g. sigmoid or ReLU

Feed forward network: forward propagation



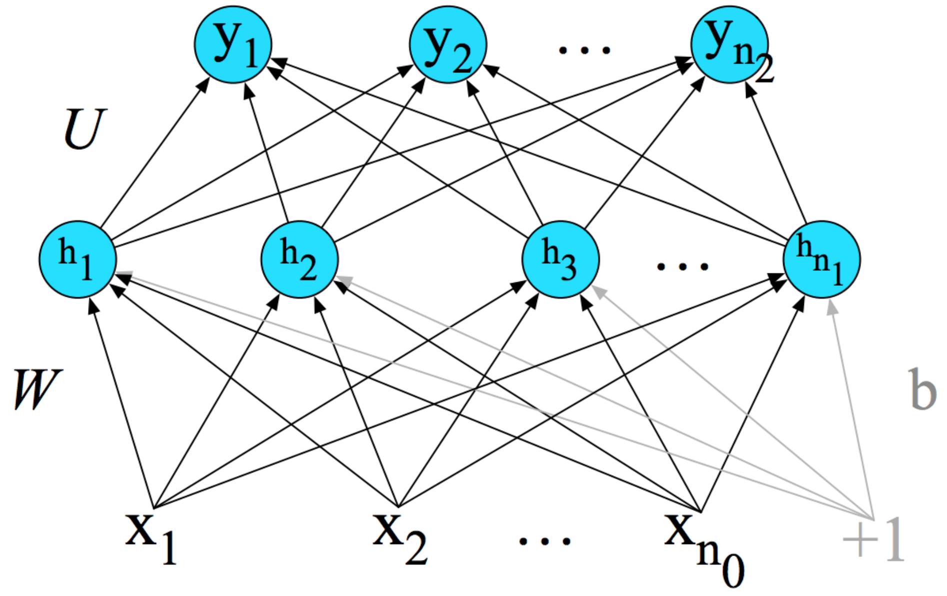
[J&M, Fig. 7.8]

$$h = g(b + Wx)$$
$$z = Uh$$
$$y = \text{softmax}(z)$$

$x \in \mathbb{R}^{n0}$ $b \in \mathbb{R}^{n1}$ $U \in \mathbb{R}^{n2 \times n1}$

$W \in \mathbb{R}^{n1 \times n0}$ $h \in \mathbb{R}^{n1}$

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"

$x \in \mathbb{R}^{n0}$

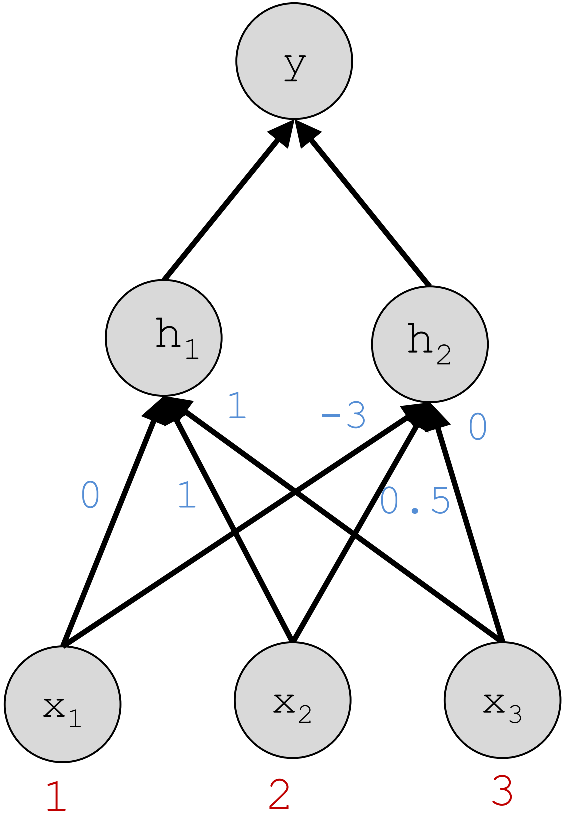
$W \in \mathbb{R}^{n1 \times n0}$

$b \in \mathbb{R}^{n1}$

$h \in \mathbb{R}^{n1}$

$U \in \mathbb{R}^{n2 \times n1}$

Feed forward network: example



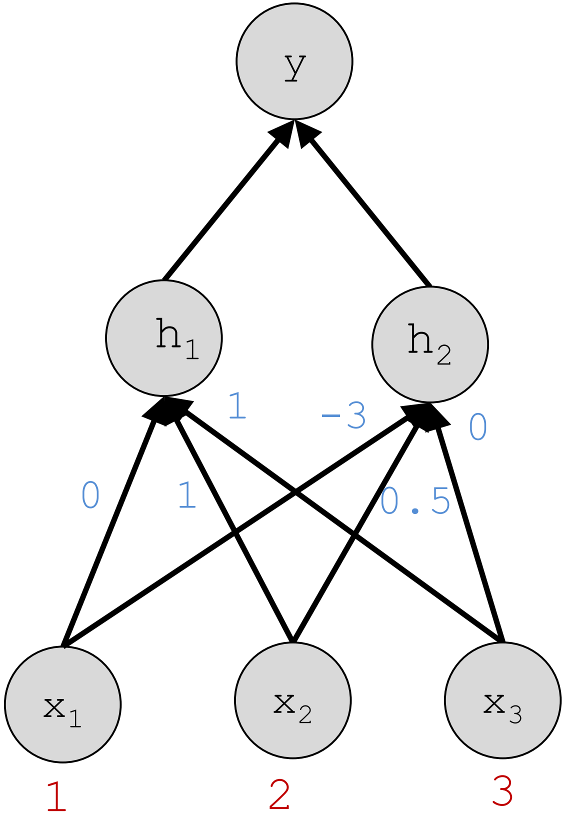

```
x = [1, 2, 3]

h1 = g(0 * 1 + 1 * 2 + 1 * 3) = g(5)
h2 = g(-3 * 1 + 0.5 * 2 + 0 * 3) = g(-2)

Using ReLU activation functions:
h = [h1, h2] = [ReLU(5), ReLU(-2)] = [5, 0]
```

```
Recall:
ReLU(x) = max(x, 0)
```

Feed forward network: example




```
x = [1, 2, 3]

h1 = g(0 * 1 + 1 * 2 + 1 * 3) = g(5)
h2 = g(-3 * 1 + 0.5 * 2 + 0 * 3) = g(-2)

Using ReLU activation functions:
h = [h1, h2] = [ReLU(5), ReLU(-2)] = [5, 0]
```

```
Recall:
ReLU(x) = max(x, 0)
```

Using matrix multiplications:

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

```
Wx = [5, -2]

h = ReLU(Wx) = [5, 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

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

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

(seen before)

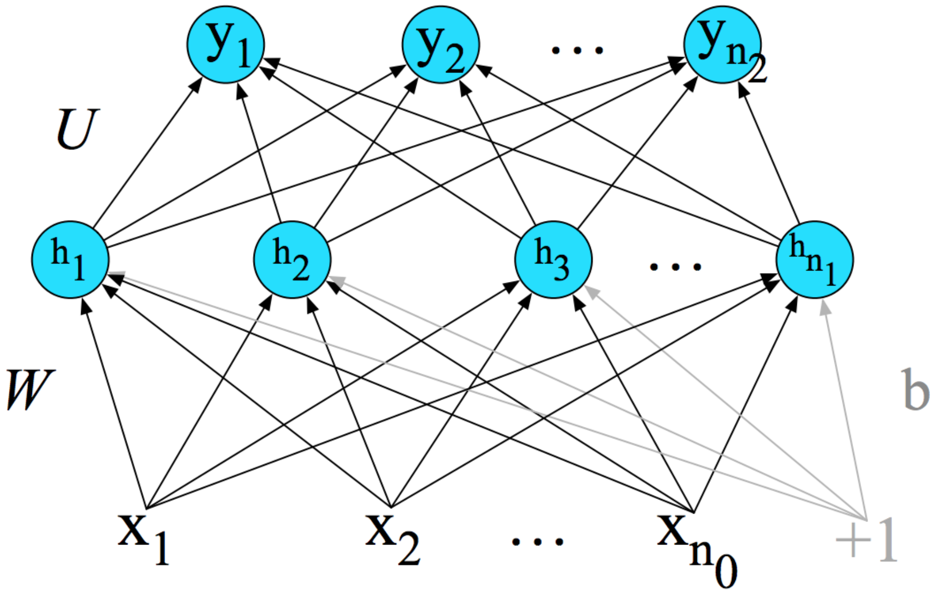
Training a feed forward network

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



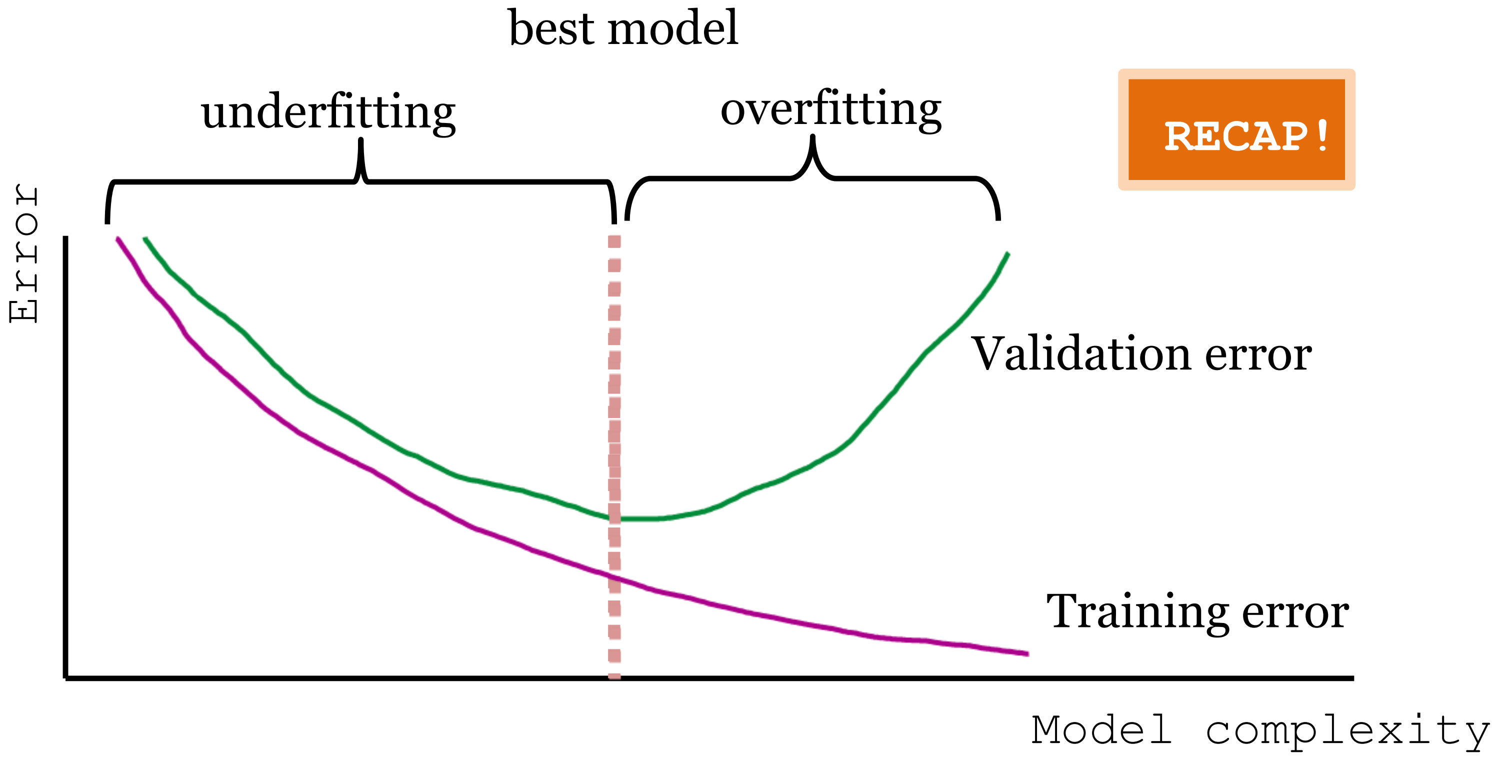
[J&M, Fig. 7.8]

Back
propagation

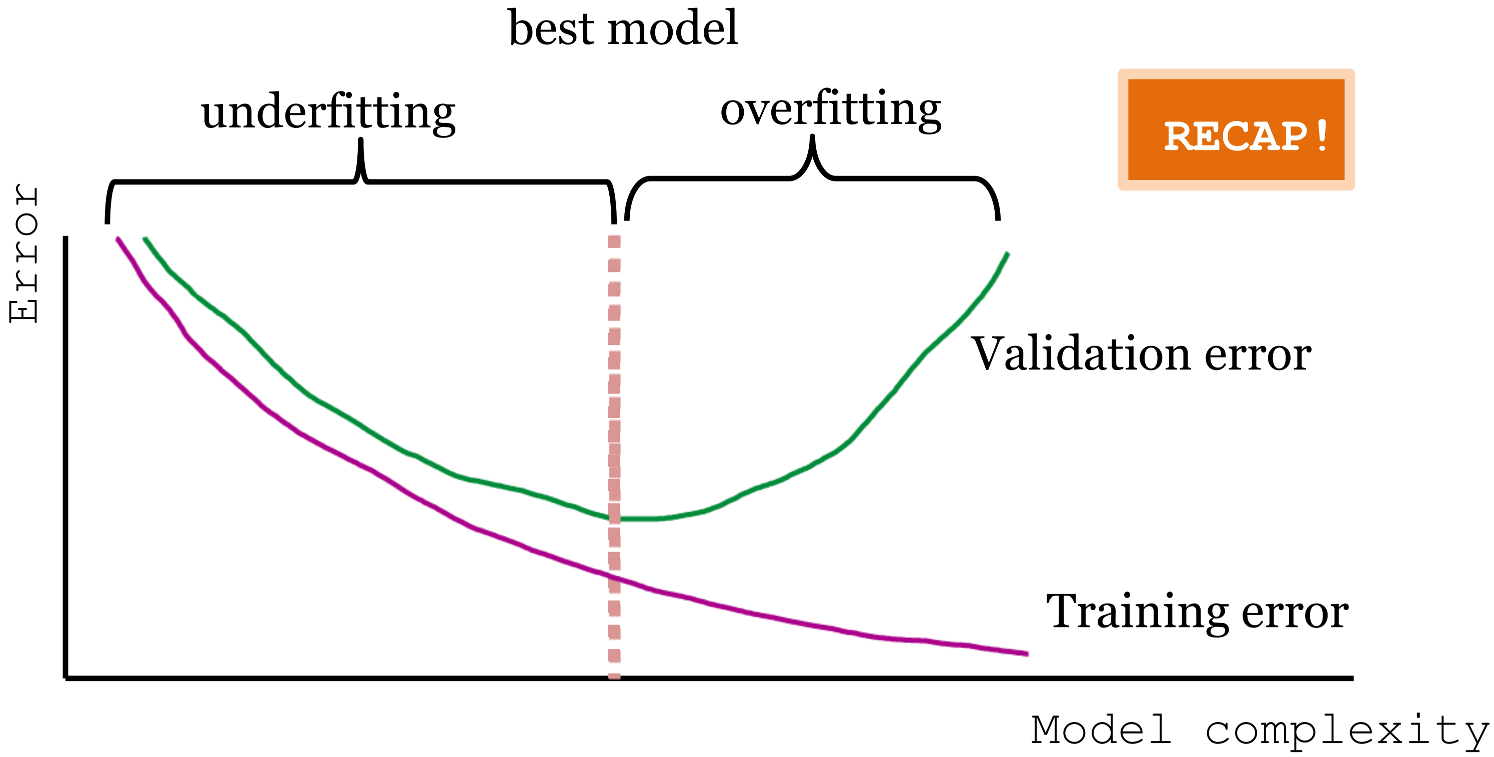
Intuitively, the (derivative of the) error for a node is distributed among previous nodes according to the weights

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

Preventing overfitting



Preventing overfitting



RECAP!

(Deep) neural networks quickly overfit!

Regularization

Logistic regression:

hyper parameter

RECAP !

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

loss

model complexity

L2 regularization

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

L1 regularization

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

Regularization

Logistic regression:

RECAP !

hyper parameter

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

loss

model complexity

L2 regularization

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

L1 regularization

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

Same idea for neural networks, but now for matrices:

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

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

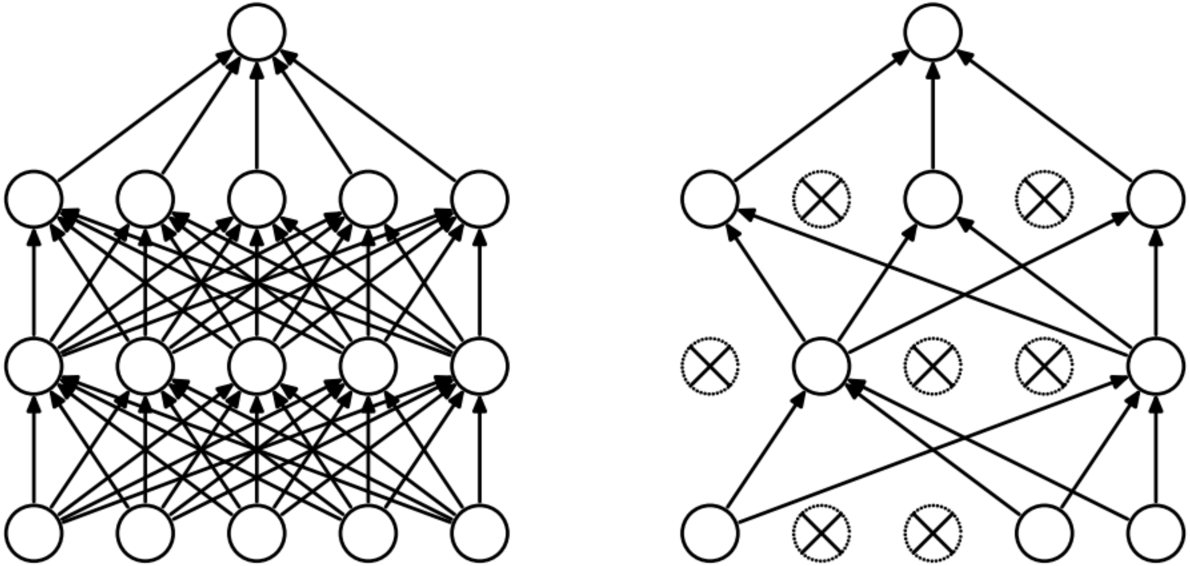
$$\mathrm{R(W)} = \|\mathrm{W}\|_1 = \sum_i \sum_j |W_{ij}|$$

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.

Srivastava et al. 2014

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

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!

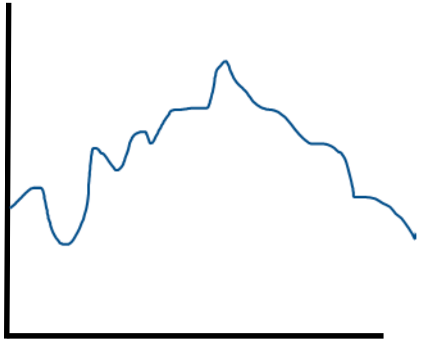
See also: https://karpathy.github.io/2019/04/25/recipe/
(A Recipe for Training Neural Networks Apr 25, 2019)

Beyond feed forward networks

Recurrent Neural Networks

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

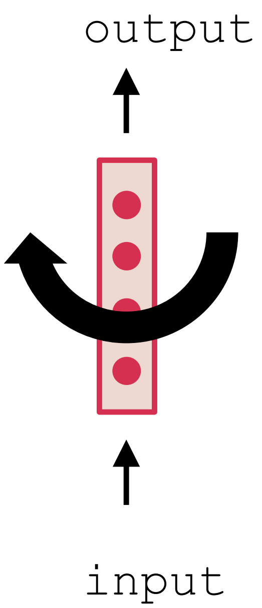
time series, e.g., financial data,
speech recognition,
language (e.g. sentences)

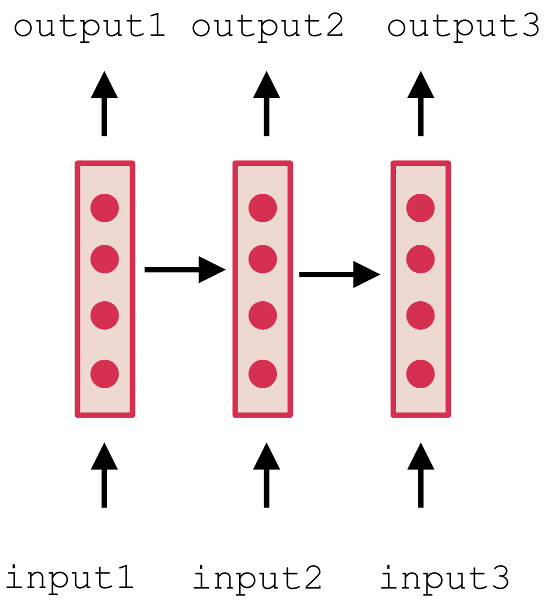


Time series

This movie was not great

Recurrent Neural Networks



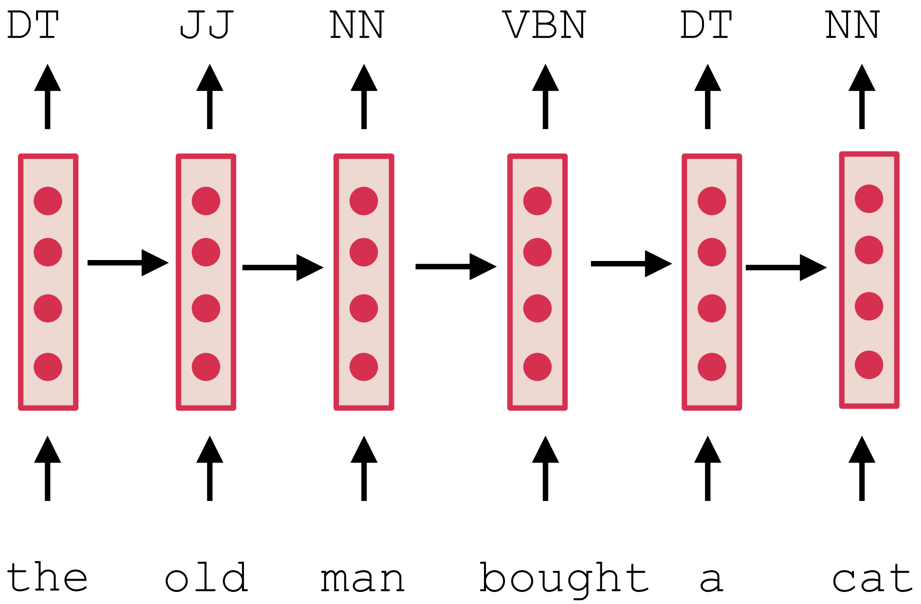


Take sequential input. Apply the same weights on each time step.

Can handle sequences of arbitrary lengths

Prediction depends on the results for previous elements of the sequence

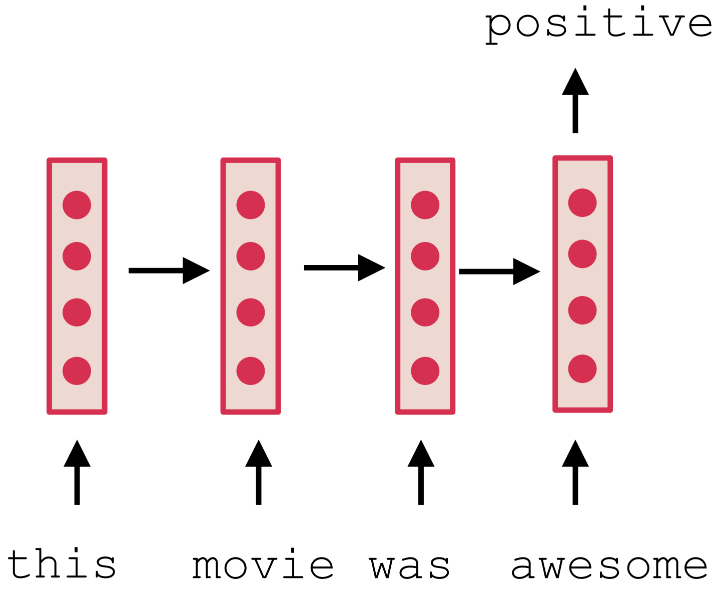
Recurrent Neural Networks



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

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

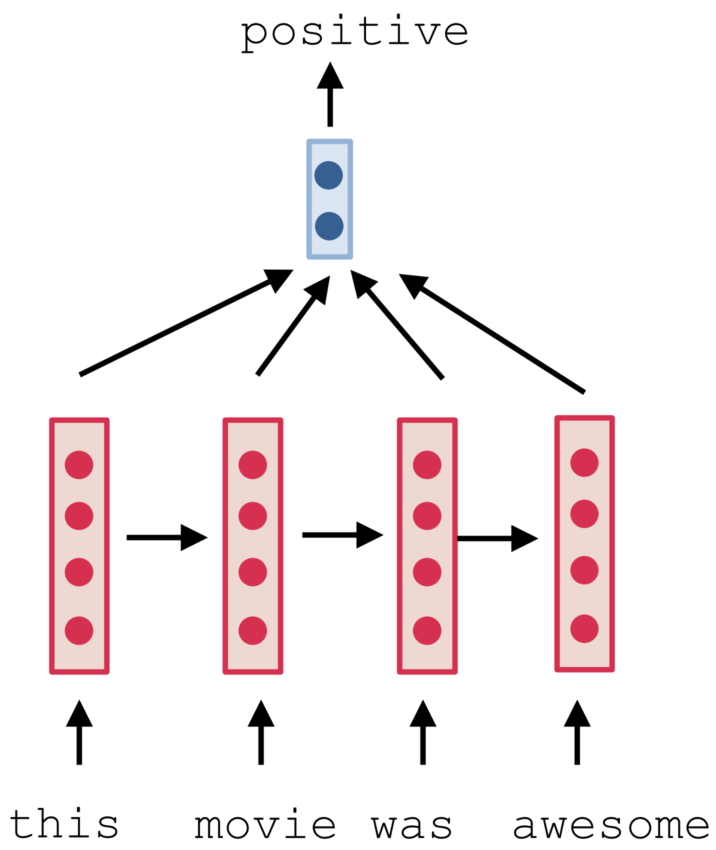
Recurrent Neural Networks



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

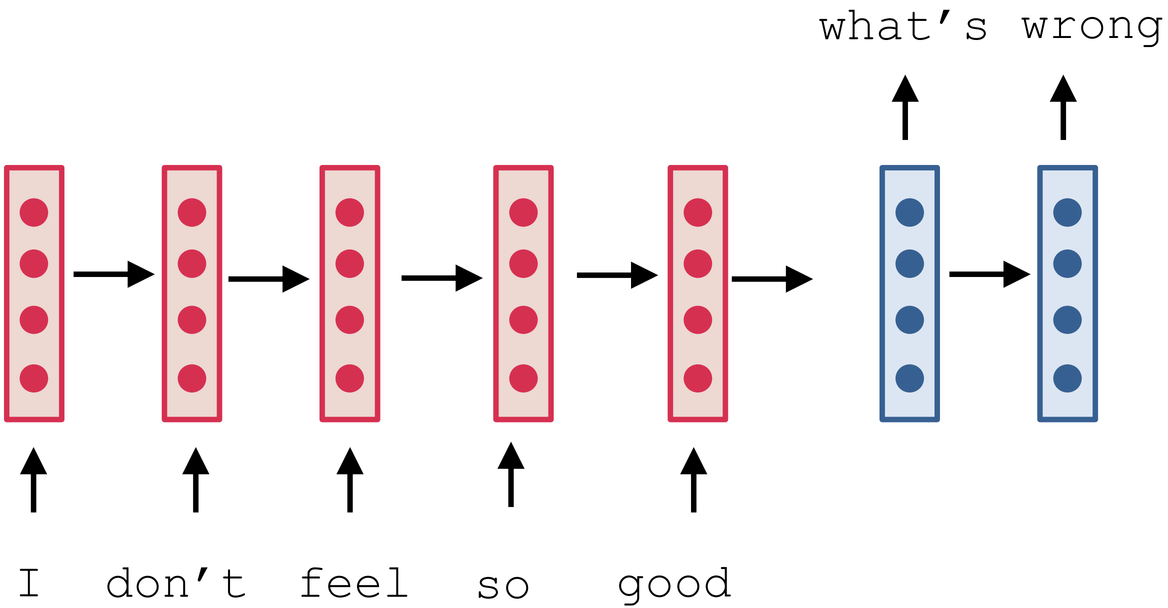


Take sequential input. Apply the same weights on each time step.

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

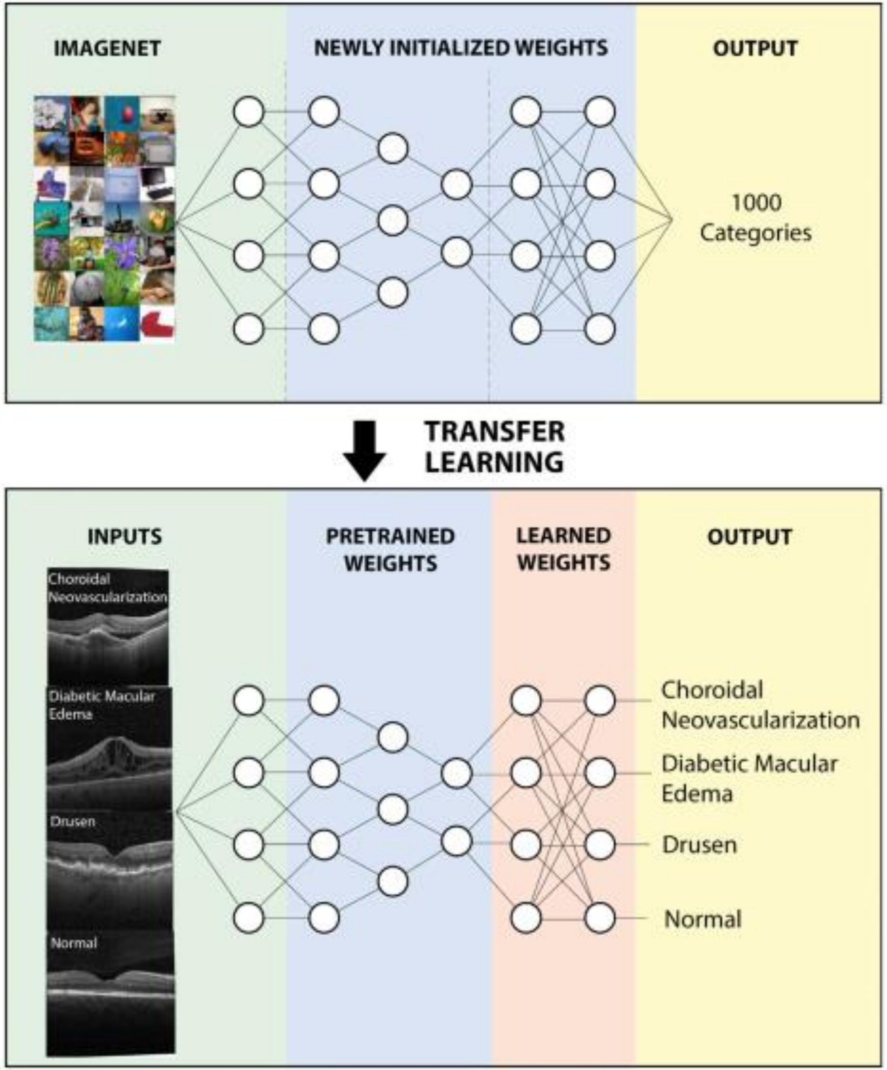
Alternative: Aggregate all hidden states (e.g. mean or max)

Recurrent Neural Networks



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

Also called sequence-to-sequence networks (seq2seq)



Transfer learning

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

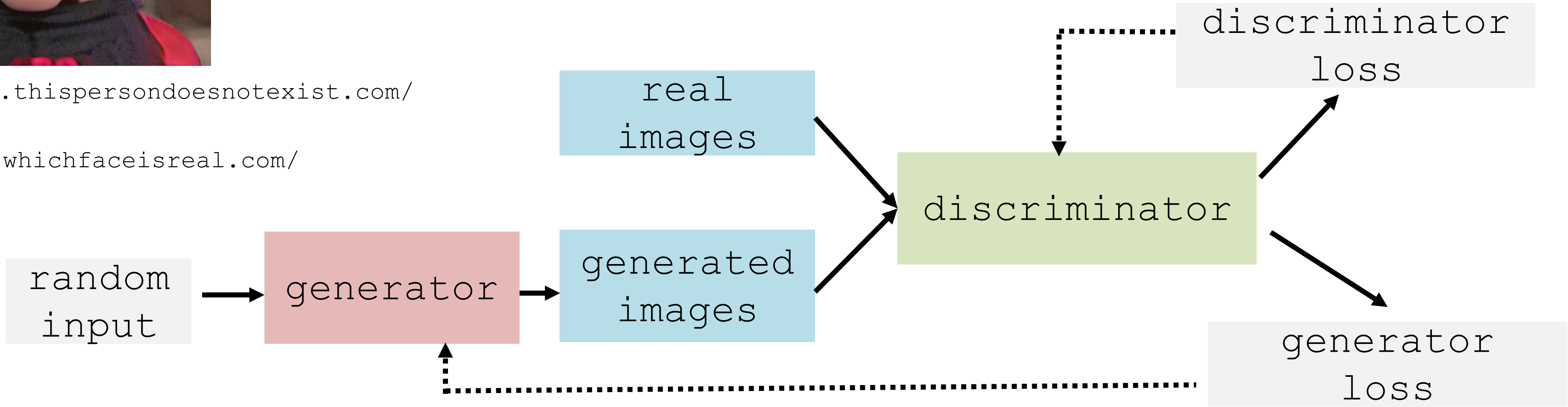
[Image from Kermany et al., Cell 2018]

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


http://www.whichfaceisreal.com/

http://www.whichfaceisreal.com/

Neural networks: pros and cons

- Can learn complex non-linear hypotheses
- Various types of architectures (e.g. for sequential series, adversarial networks).

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