Lecture 4

06.10.2025

Lecture 4

06.10.2025

Today's plan and announcements

- Review: last week's summary
- Feature engineering
- Neural networks

- Quiz 1: 15.10 during the exercise hour. ~~They are~~ It is 30 minutes and no aid is allowed (no books/notes, no electronics).
- How to prepare?
  - Problem sets 1, 2
  - Exercises in the lecture

Review: logistic regression exercise

- You are given a dataset $\{(x^i, y^i)\}_{i=1}^3 = \{(1,1),(2,0),(3,1)\}$. You will use linear regression to predict label of a new datapoint x^{test}. Consider a logistic predictor: $\hat{y} = 1 \overset{z=}{\iff} b + wx > 0$

1. Write the logistic regression loss function given the above training data.
2. Derive the gradient of the loss function with respect to b, w ... use chain rule
3. Why is the loss convex in the predictor parameters b, w? Hessian is positive semi definite
4. What is the probability of x^{test} belonging to class 0? $1 - \sigma(z^{\text{test}}) = \frac{e^{-z^{\text{test}}}}{1+e^{-z^{\text{test}}}}$
5. You find that your prediction error was large and consider using a degree 2-polynomial for encoding the feature. What are the new features?

$f_{\text{poly}}(x) = c_0 + c_1 x + c_2 x^2$

$x^i = \begin{bmatrix} 1 \\ x^i \\ (x^i)^2 \end{bmatrix}$

new features ... $(1, x, x^2)$:

Review: logistic regression exercise solution

$x^1 = 1, \quad x^2 = 2, \quad x^3 = 3,$

$y^1 = 1 \quad y^2 = 0, \quad y^3 = 1$

$z = wx + b$, predicter : $\begin{cases} \hat{y}^i = 1 & \text{if } z^i \geq 0 \\ \hat{y}^i = 0 & \text{else} \end{cases}$

1. $J(w,b) = \frac{-1}{3}\left(\underbrace{\log \frac{1}{1+e^{-z^1}}}_{y^1=1} + \underbrace{\log\left(1 - \frac{1}{1+e^{-z^2}}\right)}_{y^2=0} + \underbrace{\log\left(\frac{1}{1+e^{-z^3}}\right)}_{y^3=1}\right)$

try

$\sigma(z) = \frac{1}{1+e^{-z}}$ prob. of predicting class 1

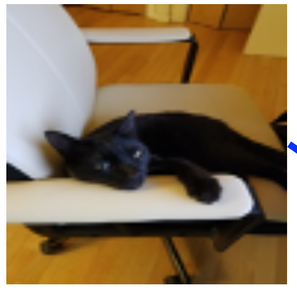
Feature engineering

What is an input representation in ML?

A representation is a mathematical form (*e.g.*, a vector)

- It describes an observation in the real-world (*e.g.*, an image, waveforms, signals, …)
- It is used for subsequent steps (*e.g.*, a classifier) to produce the outcome of interest (*e.g.*, recognizing objects)
- It is often more compact than the original observation (lower dimensionality)
- It is potentially more robust to nuisances
- With a good representation, subsequent steps should be easier

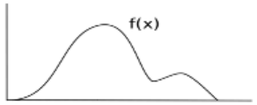
The machine learning pipeline

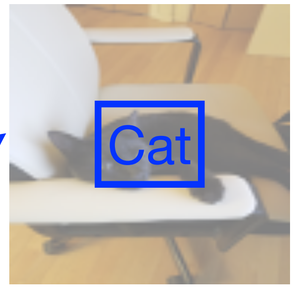


Representation

[81 20 84 64 58 39 17 54 72 15]

ML algorithm





Hey, what's the weather today?

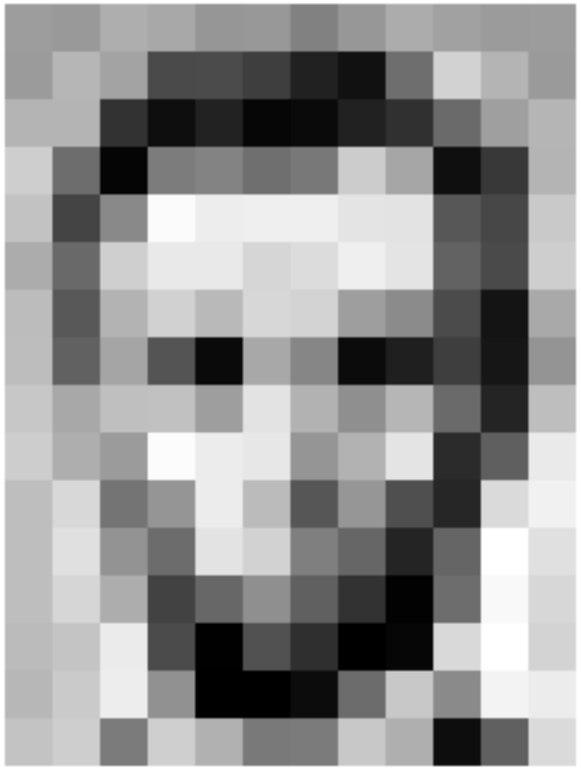
What is a Feature?

A feature vector is a representation,
a mathematical form that describes an observation in the real-world…

Examples of representations

Image, pixel values

The image is translated into features representing the value of each pixel in the image

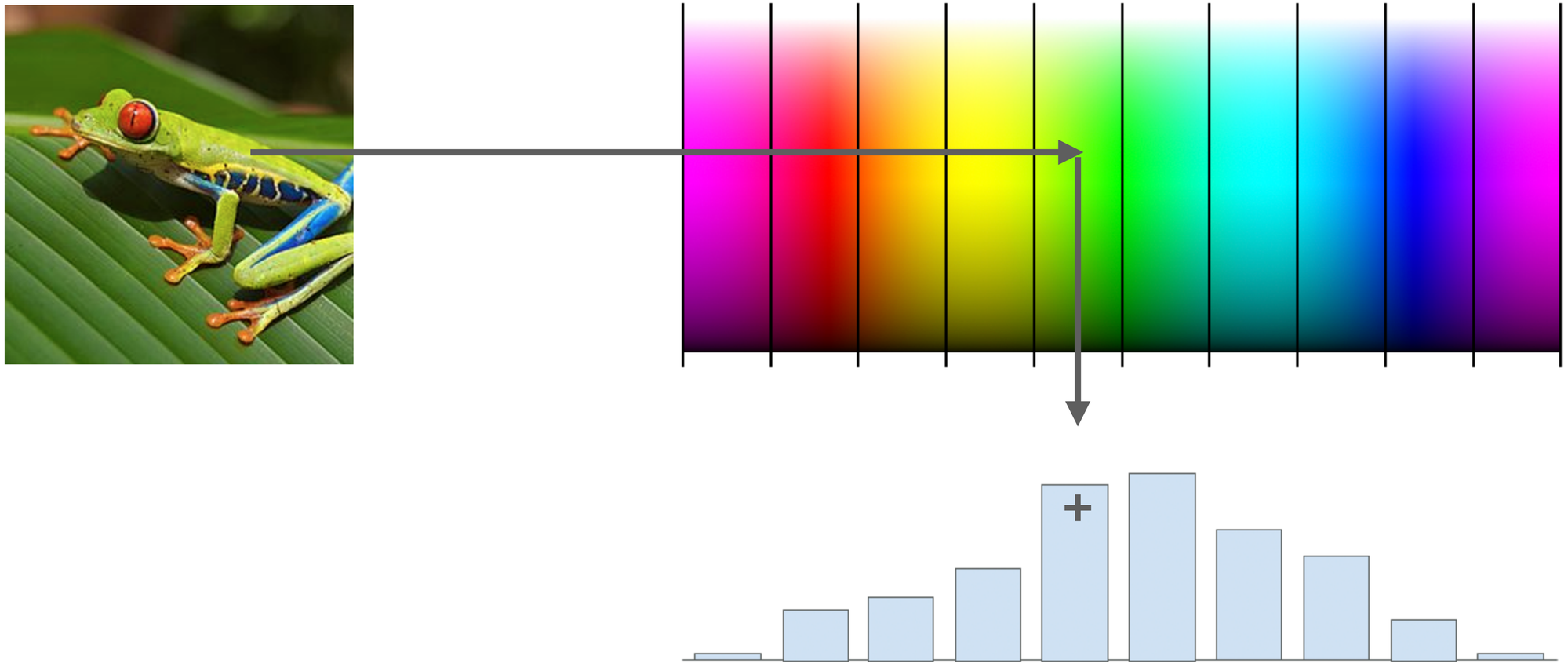


157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

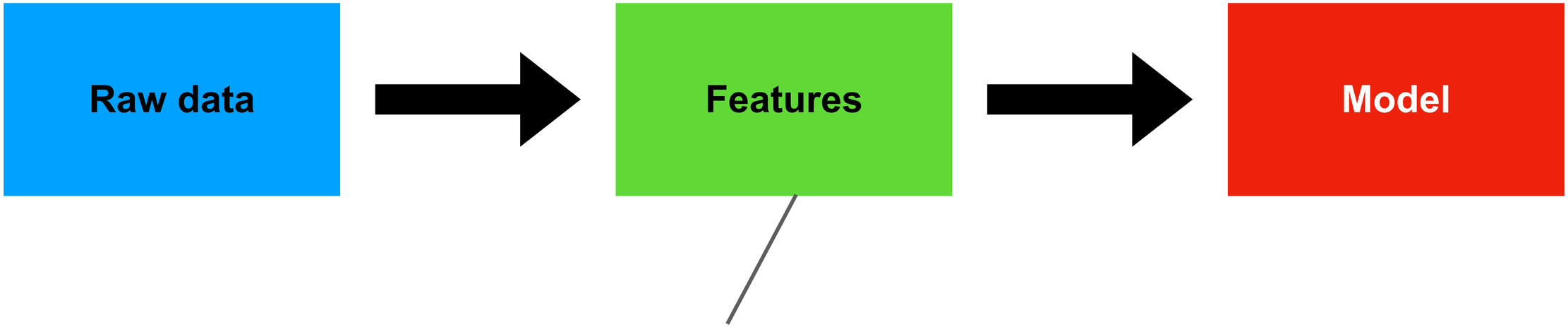
Examples of representations
Image, color histogram

number of pixels that have colors in each of a fixed list of color ranges



Feature engineering

transforming **raw data** into a **feature vector** that represents the underlying data well



Designing clever features is a key part of the machine learning

For simple models, most of the “heavy lifting” is done there

Types of features

- **Numerical**
  - *e.g.,* height, temperature, price, …
- **Ordinal (an intrinsic order on the categories)**
  - *e.g.,* “like”, “somewhat like”, “neutral”, “somewhat dislike”, “dislike”
- **Categorical (no intrinsic order on the finite categories)**
  - *e.g.,* color, gender, species, …

Preprocessing: the process of transforming raw feature vectors into a representation that is more suitable for ML algorithms

Techniques differ depending on type of feature:

- Numerical, ordinal and categorical features need to be handled differently

Summary statistics for "looking at" a feature

Data: $\{x^i, y^i\}_{i=1}^N$, $x^i \in \mathbb{R}^d$. Let's consider a given feature vector: $\{x_j^i\}_{i=1}^N$.

Mean: average value. $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_j^i$

Variance: $\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_j^i - \mu_j)^2$, Standard deviation: σ_j

Quantile: order $\{x_j^i\}_{i=1}^N$ from lowest to highest value

k-th q-quantile is x_j^I, where $I = \frac{Nk}{q}$ is index of data, example:

If I is not an integer, round up to nearest integer

If I is an integer, ~~round up to nearest integer~~

Summary statistics for “looking at” a feature

Data: ordered

$q = 4$

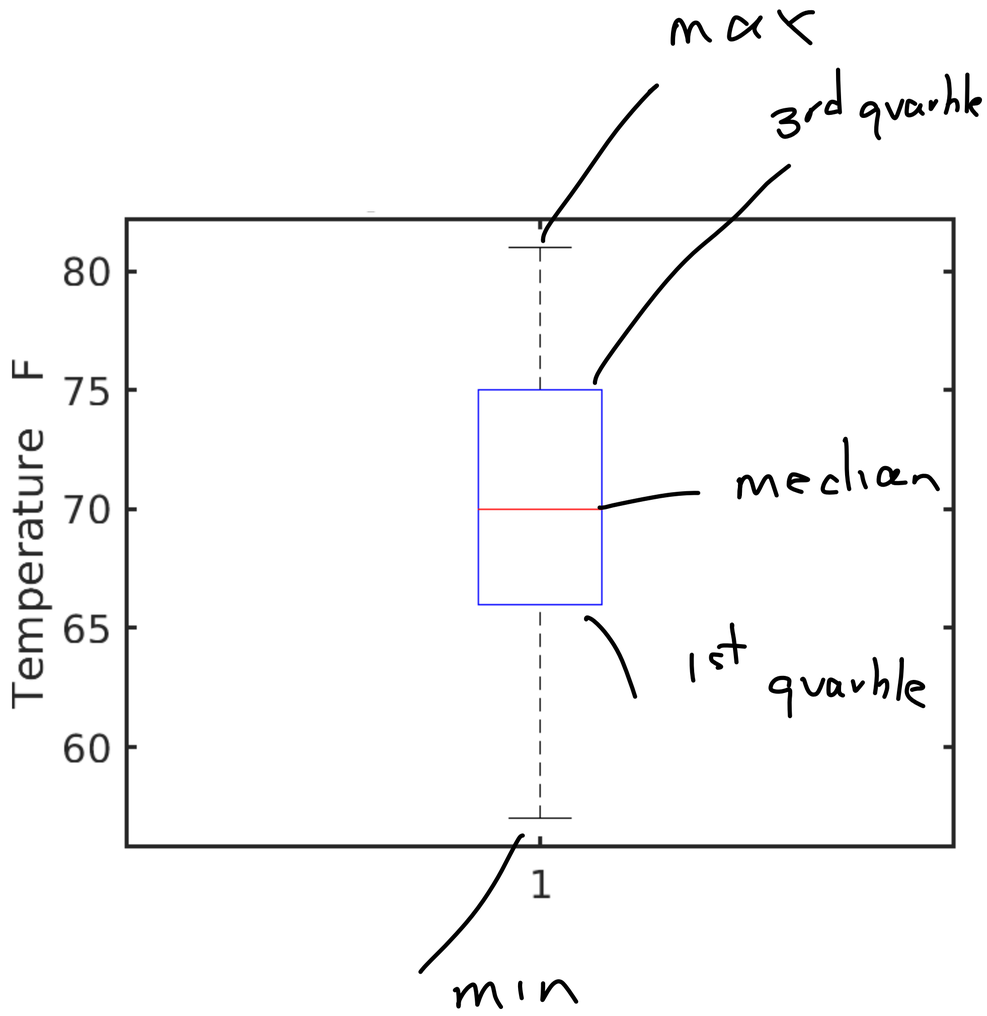
Mean: average value

Mode: most occurring value

Median: (2nd quartile or 50th percentile) , $\frac{N \times 2}{4}$

1st quartile (25th percentile) $\frac{N \times 1}{4}$

3rd quartile (75th percentile): $\frac{N \times 3}{4}$



In example x_j is temperature.

EPFL

Example of summary statistics

Feature value:

$x^1 \; x^2 \; \ldots \; x^7 \; x^8 \; \ldots \; \ldots \; x^{14}$

$x = (0,1,2,3,3,5,7,8,9,10,14,15,17,200)^T \in \mathbb{R}^{14}$, $N = 14$ data points

Verify

Mean: 21

Median: 7.5 $\frac{14 \times 2}{4} = 7 = I$, x^I, or $\frac{x^I + x^{I+1}}{2}$

Mode: 3 most occurs value

1/4 Quantile: 3 $\rightarrow \frac{N \times 1}{4} = \frac{14}{4} = 3.5$, round up to 4, $x^4 = 3$

3/4 Quantile: 14 $\rightarrow \frac{N \times 3}{4} = 10.5$, round up to 11, $x^{11} = 14$

standard deviation: 51.79

Range: 200 $(\max x^i - \min x^i) = (200 - 0) = 200$

Inter (1/4, 3/4) quantile range: 11 $(x^{11} - x^4) = 14 - 3 = 11$

Numerical features

Feature scaling: bring all data points of a given feature to same scale

Crucial step in preprocessing:

- Many classifiers (e.g. KNN that we will see in next lectures) rely on distance metrics
- Gradient descent will converge faster
- Coefficients are penalized appropriately (in the case where regularization is applied)

Data normalization - example

Example:

KNN with Palmer Penguins dataset

Features: body mass & bill length

Q: Which feature matters the most for the distance metric?

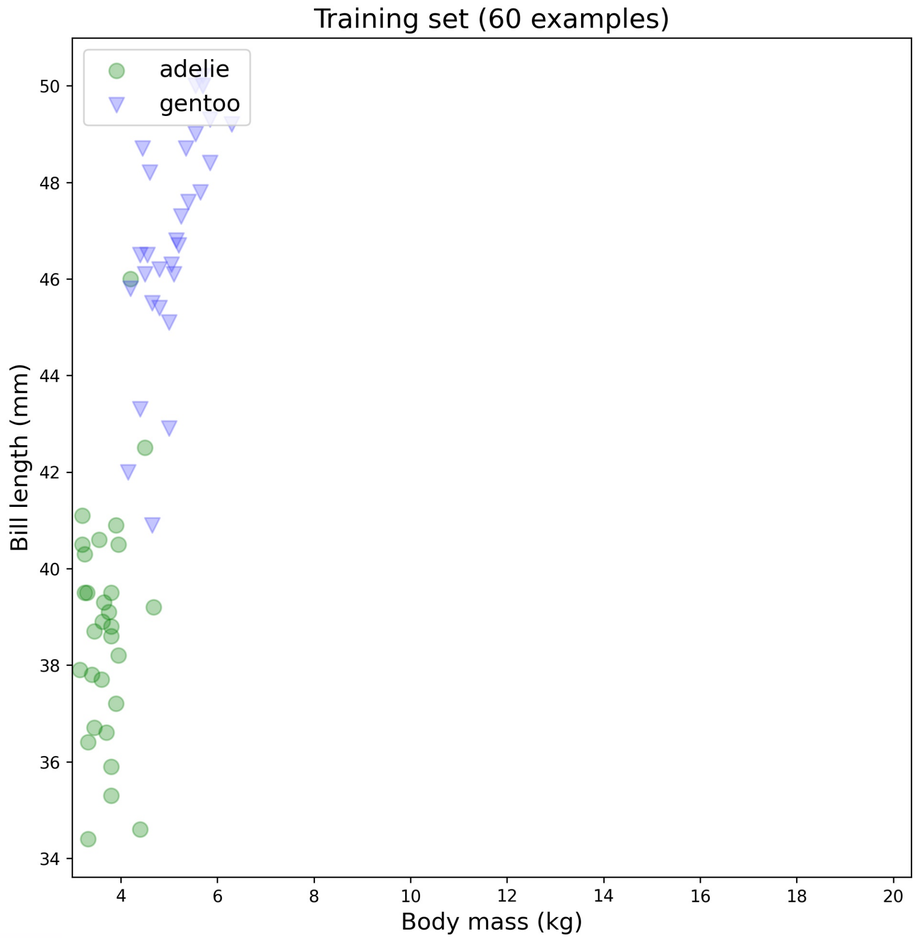
- If body mass in **kg** and bill length in **mm**

  → **bill length** matters more

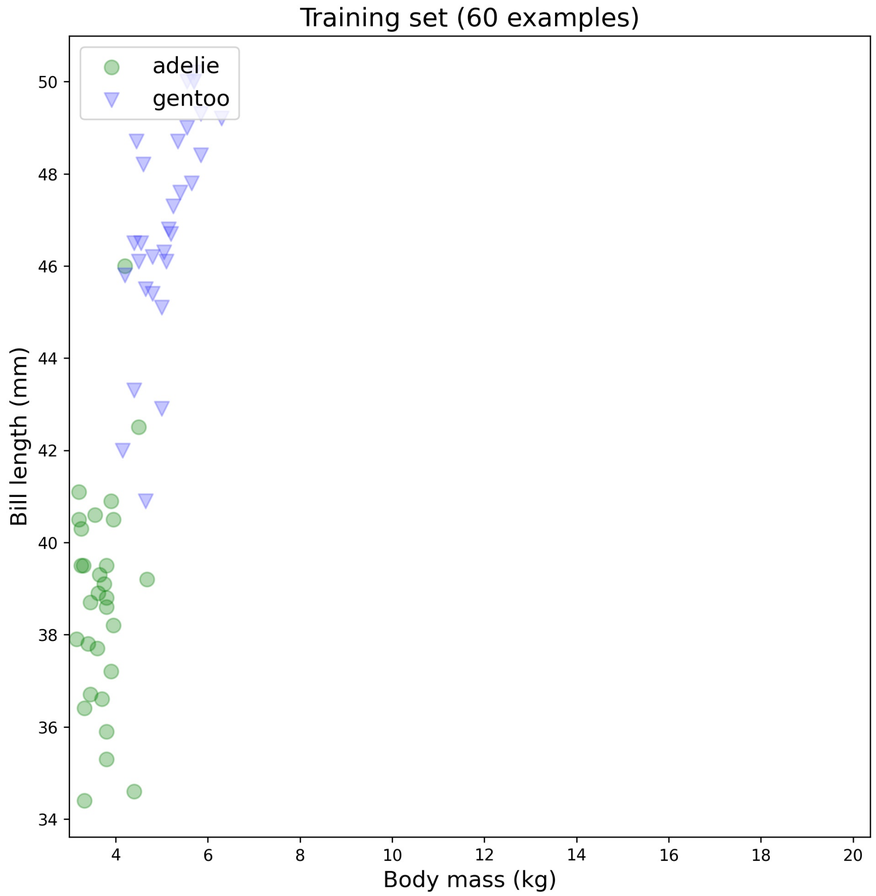
- If body mass in **g** and bill length in **m**

  → **body mass** matters more

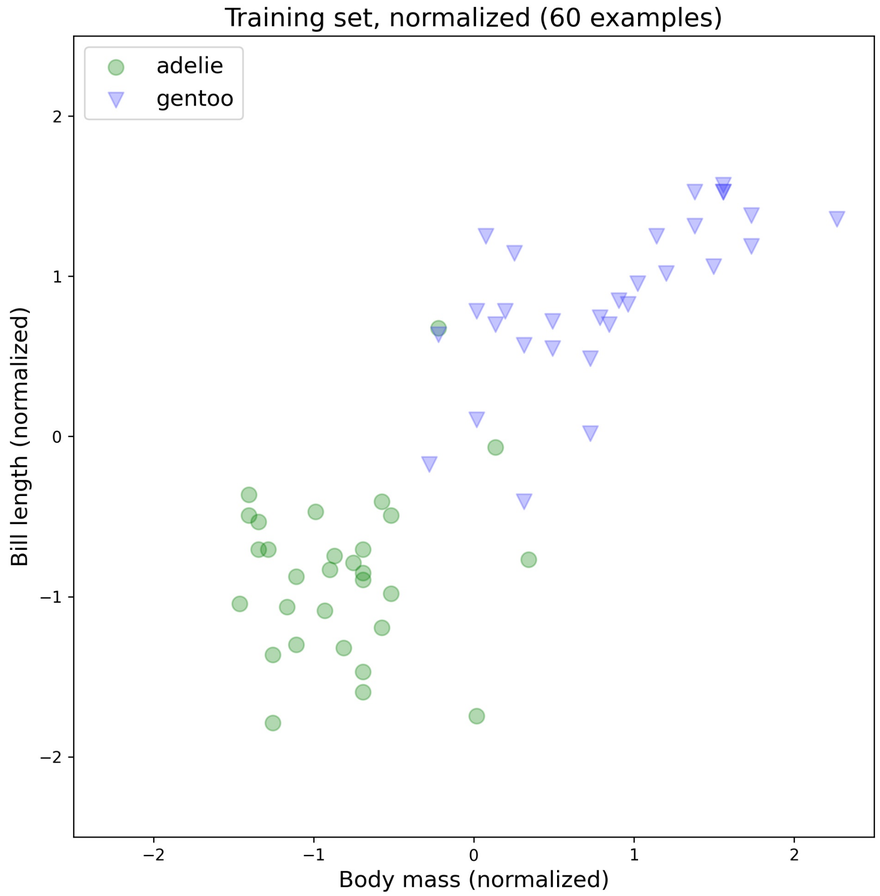
With normalization, the units of the features stop playing an important role in the model accuracy



Data normalization - example



Normalization



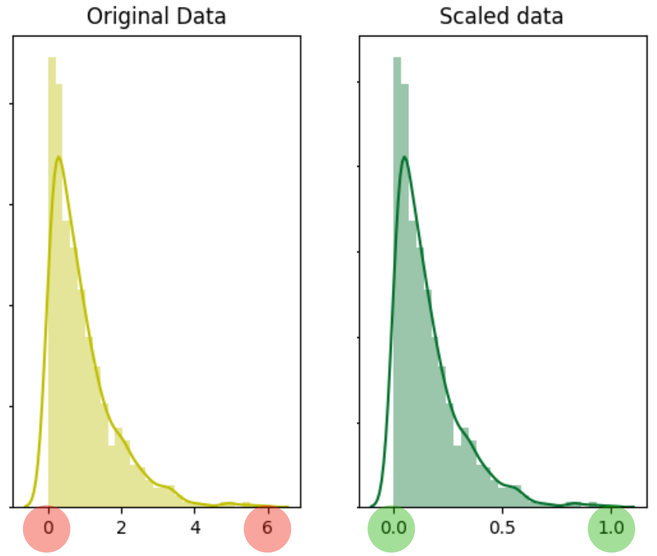
Numerical features scaling approaches

1) **Normalization**

Scale each dimension of data to the range [0, 1]

For each dimension:

$$x^{i}_{norm} = \frac{x^{i} - \min(x^{i})}{\max(x^{i}) - \min(x^{i})}$$

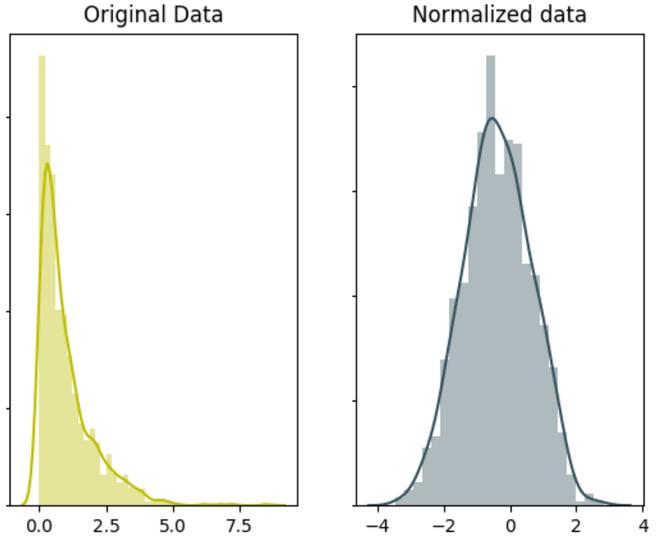


2) **Standardization** (z-score normalization)

Set mean of each dimension (**μ**) to 0, standard deviation (**σ**) to 1

For each dimension:

$$x^{i}_{norm} = \frac{x^{i} - \mu_x}{\sigma_x}$$



Numerical features - outliers

Features may have outliers or follow some heavy-tailed distribution

- Example urban area population:
  - most urban areas have a few thousands inhabitants
  - a handful of urban areas have tens of millions of inhabitants (New York, Tokyo, …)

- With **min-max** scaling:
  - typical values are scaled to a very small interval

How to deal with outliers

- Different scaling effects, Consider $(10,12,13,15,1000)$
  - Normalization: $(0,0.002,0.003,0.005,1)$ (min-max normalization)
  - Standardization: $(-0.51,-0.50,-0.50,-0.49,+2.00)$ (z-score)
  - Scaling using quantiles: $(-1.00,-0.33,0.00,+0.67,+329.00)$

no need to know this formula

$$\frac{x^i - x^{I_1}}{x^{I_3} - x^{I_1}}$$

I_1 first quartile

  - Logarithmic scaling: $(2.30,2.49,2.57,2.71,6.91)$

Logarithmic scaling

Logarithmic scaling: take the log of the feature

We use it when a feature: has strictly positive value and spans several order of magnitude

- Example, let feature be lot area for housing and you want to predict prices based on that

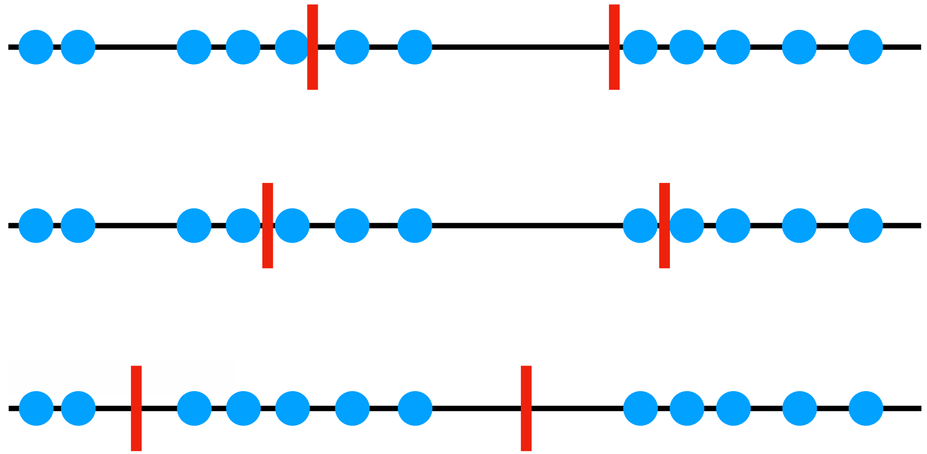
  $\boldsymbol{x} = [300\ 400\ 500\ 8000\ 25000]$

  $\rightarrow \log(\boldsymbol{x}) = [2.48\ 2.60\ 2.70\ 3.90\ 4.40]$
- After log scaling, you can then use normalization/standardization

Numerical features

Binning / discretization: partition continuous features into discrete values

- *Example: study hours:* 0-3 very low, 4-10 low, 11-15 medium, 16-25 high, 25+ very high
  - Handles outliers
  - More interpretable models
  - Can be helpful (data noisy or some threshold effects)



Common types of binning:

- **Uniform:** all bins have identical widths
- **Quantile:** all bins have the same number of points
- **Clustered:** a clustering algorithm (seen later in this course) is used to separate values

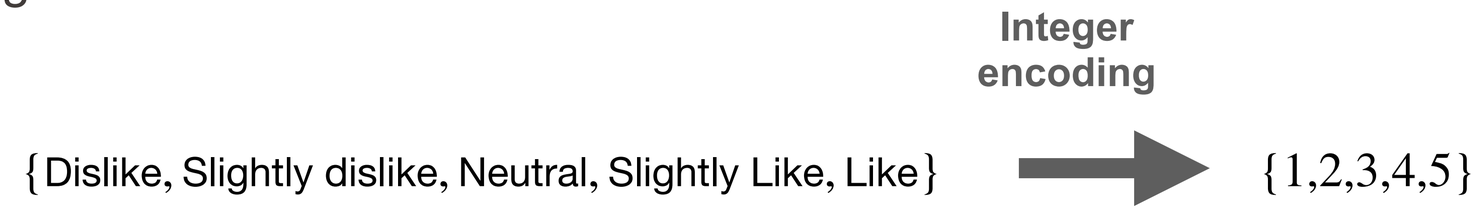
Ordinal features

Ordinal variables: **finite** set of discrete values, with a **clear ordering** between them

- Examples:
  - Satisfaction *(like, slightly like, neutral, slightly dislike, dislike)*
  - Education level *(high school, BS, MS, PhD)*

Integer encoding: map values to integers ranging from 1 to n

- n = number of unique values of the feature
- Mapping from 0 to n-1 is also used



Categorical features

Categorical variables: **finite** set of discrete values, with **no intrinsic ordering** between them

- Examples:
  - Animal species *(cat, dog, iguana, penguin, …)*
  - Eye color *(blue, green, brown, …)*
  - Sex *(male / female)*

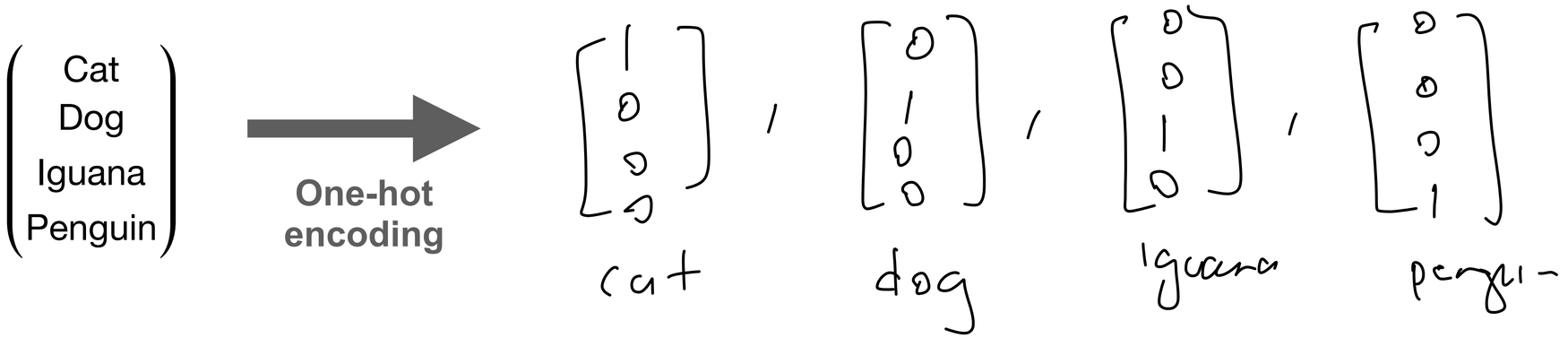
As there is no order between these values, integer encoding may not be appropriate

- e.g. assume [cat, dog, iguana, penguin] is encoded to [1, 2, 3, 4]
  - It wouldn't make sense to imply that a cat is less than an iguana, or that a penguin is closer to an iguana than it is to a cat

Instead, use **one-hot encoding**

Categorical features
One-hot encoding

One-hot encoding: Each category is represented by a binary variable



- One-hot encoding removes any assumption of order between categories
- If there are many categories, the resulting feature vector can be very large and sparse (e.g. suppose you one-hot encode every word in the dictionary)

Exercise - feature encoding

- You are analyzing data from a first-year university course to predict whether a student will pass or fail the final exam (1 = pass, 0 = fail) using a logistic regression model. The data is as follows.

features

- Hours studied per week - a number in the range [0,40] , $x_1 \in \mathbb{R}$
- Academic program: engineering, biology, math , $\begin{bmatrix}1\\0\\0\end{bmatrix}$ eng, $\begin{bmatrix}0\\1\\0\end{bmatrix}$ bio, $\begin{bmatrix}0\\0\\1\end{bmatrix}$ math
- Sleep quality: poor (0), good (1) (ordinal)
- Lecture attendance: rarely, sometimes, often, always , $\{1, 2, 3, 4\}$
- Exam outcome: pass (1) , fail (0). → target / label

- What is x (feature input) and what is y (the target/label) in this problem?
- How would you encode each type of feature?
- After encoding, how many weights and biases will the logistic regression model need to learn? since $x \in \mathbb{R}^6$, 6 weights & 1 bias needs to be determined.

Exercise solution

- Features and target
  - Features: hours studied,academic program,sleep quality,lecture attendance
  - Target: {pass, no pass} (binary)
- Encoding
  - Hours studied: numeric (normalization or standardization)
  - Academic program: one-hot encoding -> engineering: $\begin{bmatrix}1\\0\\0\end{bmatrix}$ biology: $\begin{bmatrix}0\\1\\0\end{bmatrix}$ math: $\begin{bmatrix}0\\0\\1\end{bmatrix}$
  - Sleep quality: ordinal {0,1}
  - Lecture attendance: ordinal ~~{0,1,2}~~ $\{1, 2, 3, 4\}$
- So, the feature vector becomes $x = ~~(x_1, x_2, x_3, x_4, x_5)~~$ $(x_1, x_2, x_3, x_4, x_5, x_6) \in \mathbb{R}^6$

example : $\begin{bmatrix}11\\0\\1\\0\\0\\2\end{bmatrix}$

means : 11 hour study

major : biology

sleep quality poor

lecture attendance : sometimes

Missing values

Raw data isn't always clean, it is quite common for some values to be missing

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	NaN	41.0	6.984127	NaN	322.0	2.555556	37.88	-122.23
1	NaN	21.0	NaN	0.971880	2401.0	NaN	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	NaN	1.073059	558.0	NaN	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
5	4.0368	52.0	NaN	1.103627	413.0	NaN	37.85	-122.25
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25
7	NaN	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25

Each row is block group of population
Each column is defined as:

- MedInc median income in block
- HouseAge median house age in block
- AveRooms average number of rooms
- AveBedrms average number of bedrooms
- Population block population
- AveOccup average house occupancy
- Latitude house block latitude
- Longitude house block longitude

Missing values approaches

Many ML algorithms cannot work with missing values

→ handling missing values properly is very important

How to handle missing data?

Deletion (simplest solution)

- If only a few features have missing values, delete these features (i.e. delete the column)
- If only a few rows have missing values, delete these rows

Imputation: Replace missing values by some reasonable estimate

- Many different data imputation techniques exist

Missing values imputation

- **Constant imputation:** Impute with constant value different from all other values in the feature (such as 0)
- **Mean imputation for a numerical feature:** Impute with mean of feature
- **Median imputation for ordinal feature:** impute the mode of the categorical/ordinal features
- **Mode imputation for categorical feature:** impute the mode of the categorical/ ordinal features
- **KNN imputation:** Impute with mean/mode of k nearest neighbors to data point (we will discuss KNN approach)

Missing values
Imputation - Numerical

Example: Mean imputation for housing dataset

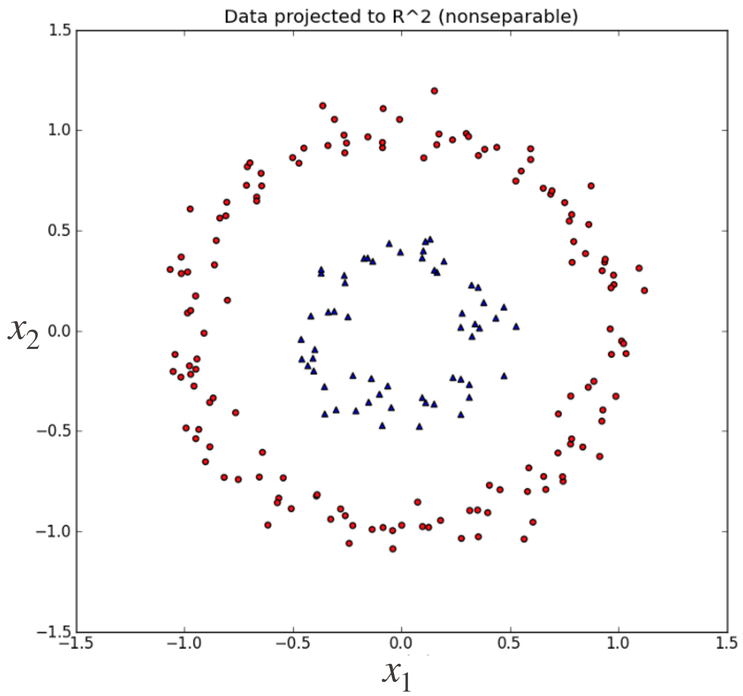
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
0	NaN	41.0	6.984127	NaN	322.0	2.555556
1	NaN	21.0	NaN	0.971880	2401.0	NaN
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260
3	5.6431	52.0	NaN	1.073059	558.0	NaN
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467
5	4.0368	52.0	NaN	1.103627	413.0	NaN
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405
7	NaN	52.0	4.797527	1.061824	1157.0	1.788253

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
0	4.88852	41.0	6.984127	1.045183	322.0	2.555556
1	4.88852	21.0	6.256710	0.971880	2401.0	2.291188
2	7.25740	52.0	8.288136	1.073446	496.0	2.802260
3	5.64310	52.0	6.256710	1.073059	558.0	2.291188
4	3.84620	52.0	6.281853	1.081081	565.0	2.181467
5	4.03680	52.0	6.256710	1.103627	413.0	2.291188
6	3.65910	52.0	4.931907	0.951362	1094.0	2.128405
7	4.88852	52.0	4.797527	1.061824	1157.0	1.788253

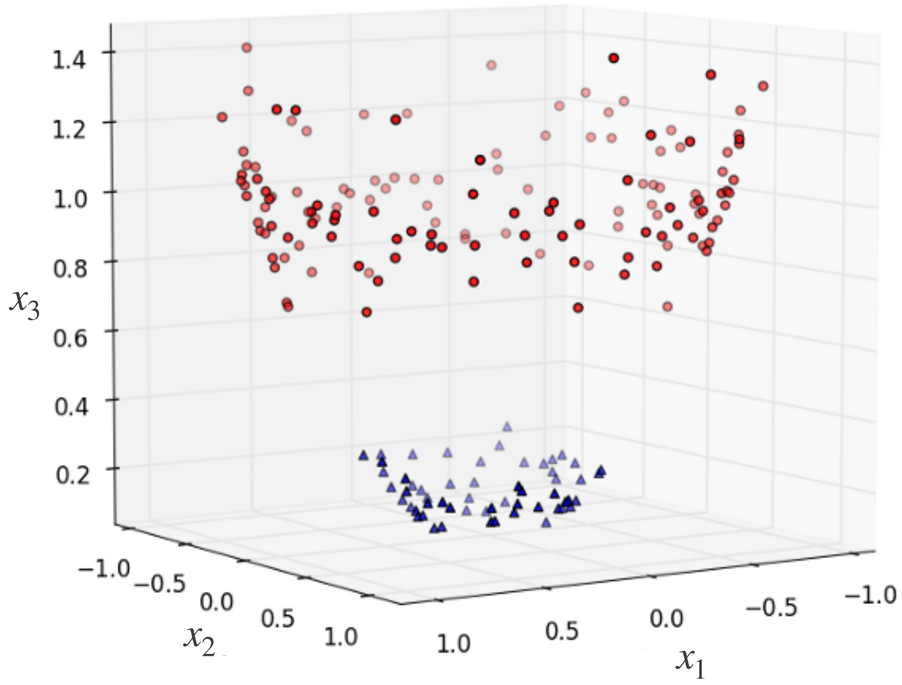
Feature expansion

Creating new features based on existing ones

Example: Let's add a synthetic feature: $x_3 = x_1^2 + x_2^2$



$$\boldsymbol{x}' = (x_1, x_2, x_3)$$
$$= (x_1, x_2, x_1^2 + x_2^2)$$



x_3 encodes a non-linearity in the feature space

By adding x_3, the data becomes linearly separable!

Feature expansion review

Feature expansion is the process of creating derived features from the input data

- Adds complexity at the benefit of
  - Reduces underfitting
  - Can significantly improve performance

Popular feature expansion techniques:

- **Feature crosses:** Multiply features together
- **Binning:** Transform a numerical feature into several categorical or ordinal features
- **Applying a non-linear function to each feature** (*e.g.,* sin, log, polynomials)

Neural networks

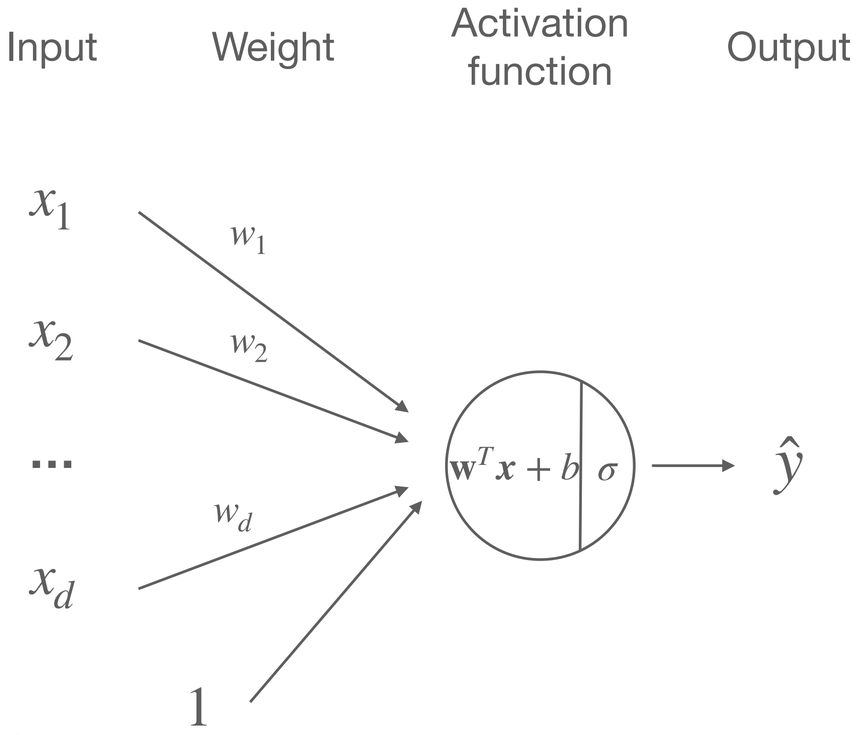
Why deep learning?
Logistic regression review

Logistic regression for d-dimensional data:

Input: $x \in \mathbb{R}^d$

Weights: $w \in \mathbb{R}^d$

Bias: $b \in \mathbb{R}$

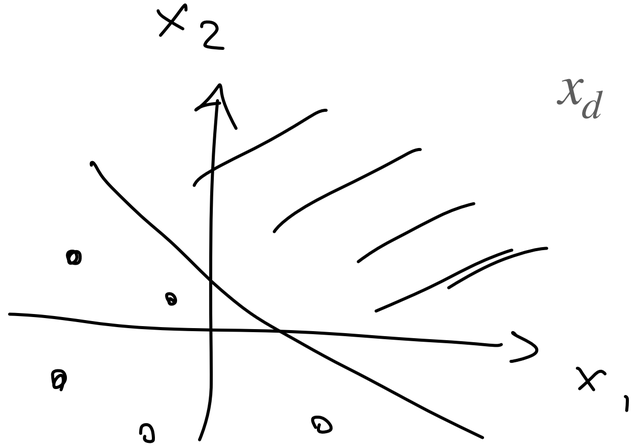


Predicting a half space

$$z = w^T x + b$$

$z \geq 0 \quad \hat{y} = 1$

$z < 0 \quad \hat{y} = 0$



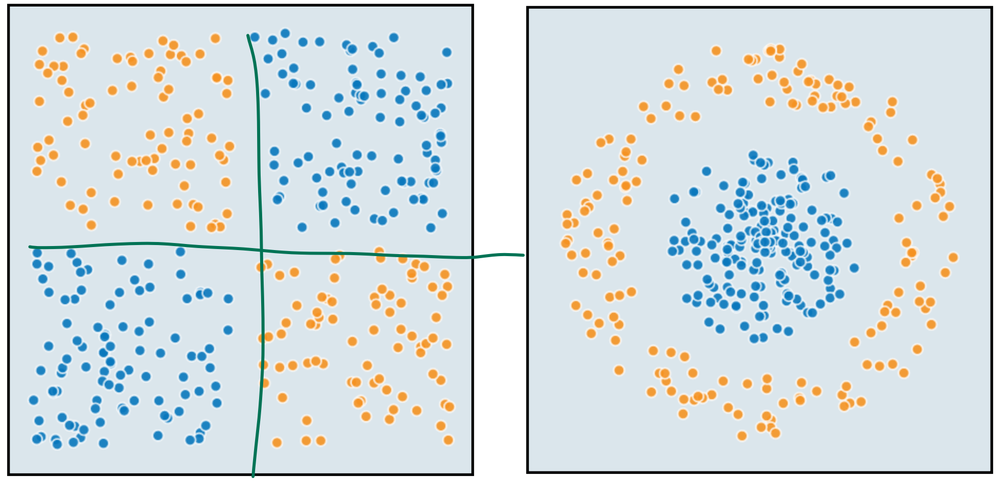
Why deep learning?
Limitations of logistic regression

Issue: Logistic regression performs badly on non-linearly separable data

Potential fix: Use feature engineering to make data linearly separable, then use logistic regression

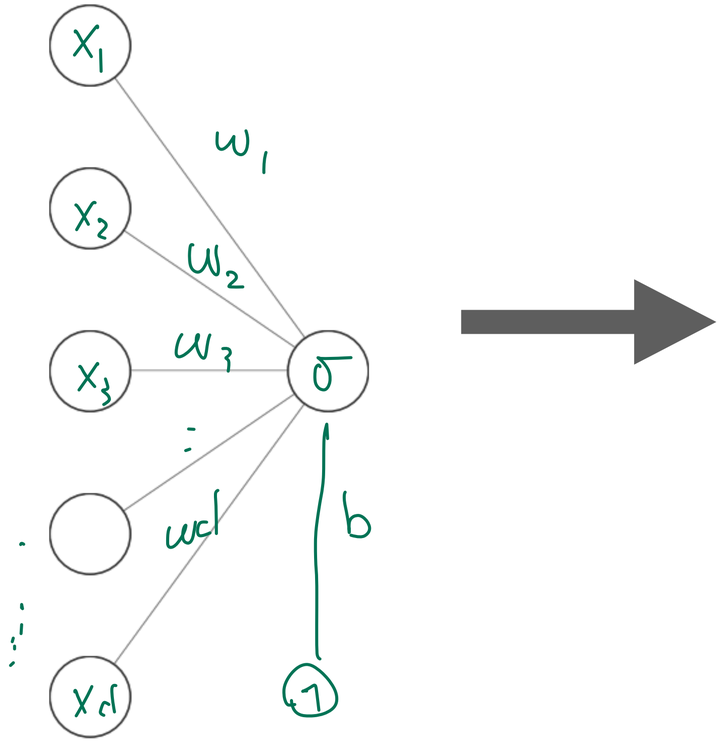
However:

- Features that linearly separate the data can be hard to find manually, specially in high dimension

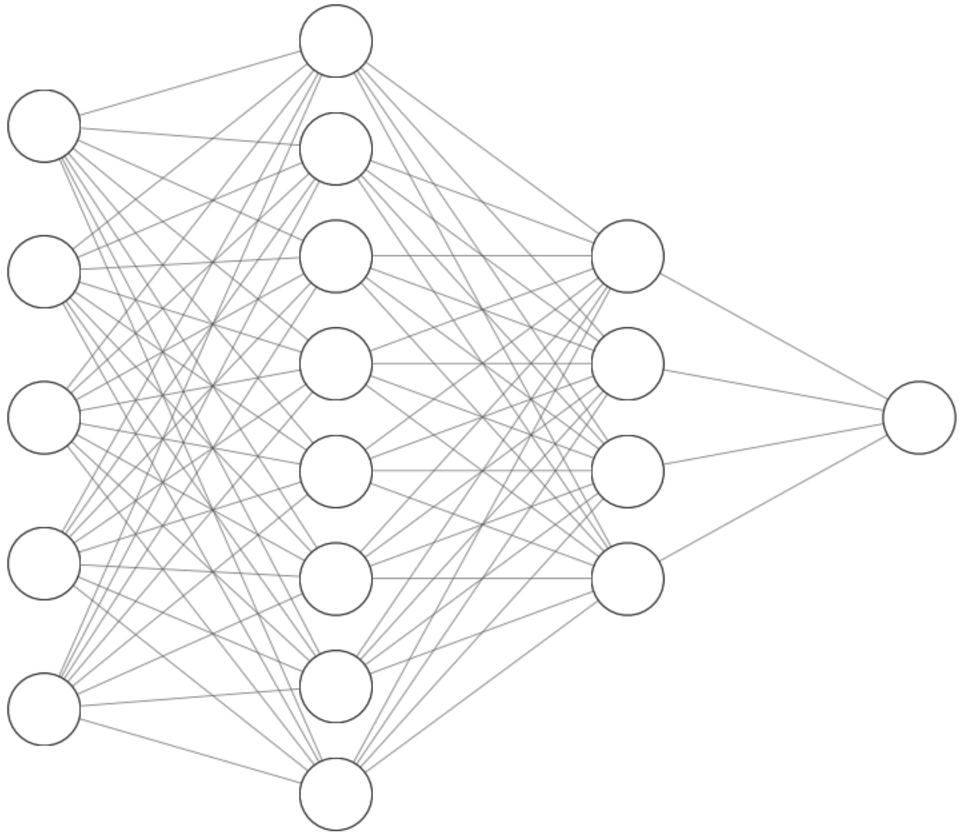


Why deep learning?
From logistic regression to neural networks

Neural networks have been successful in learning complex, non-linear functions

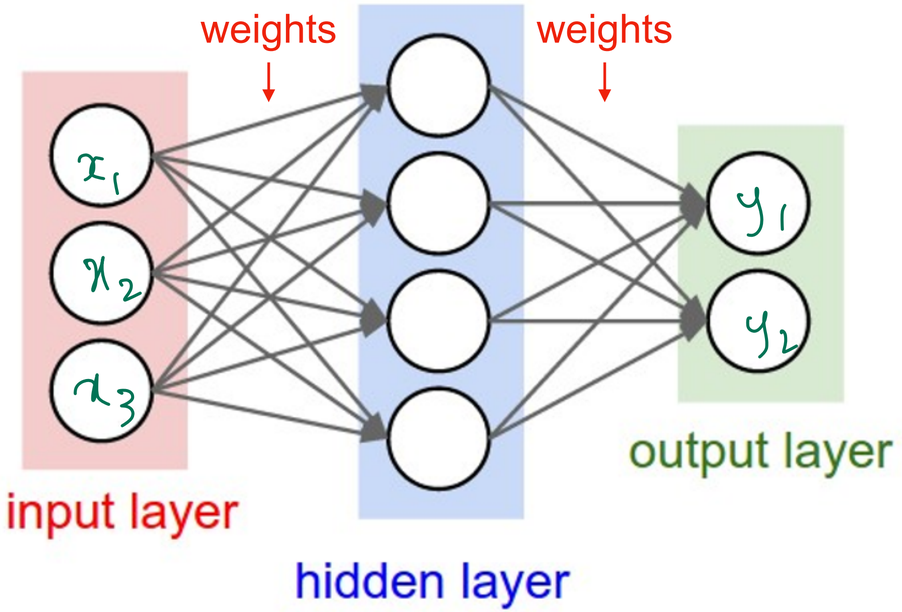


Input Layer $\in \mathbb{R}^5$ Output Layer $\in \mathbb{R}^1$

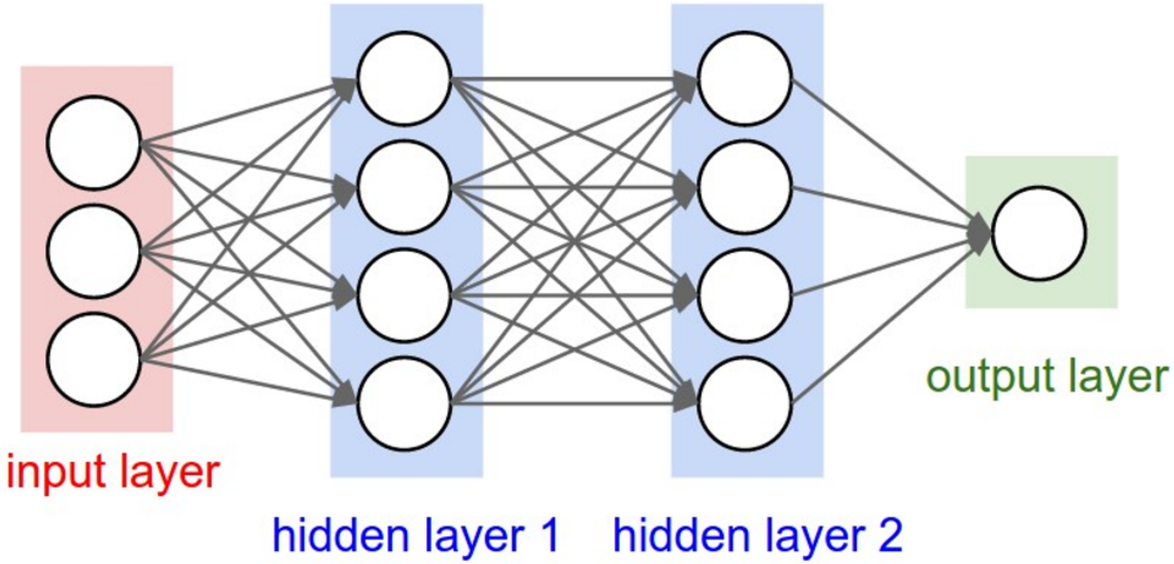


Input Layer $\in \mathbb{R}^5$ Hidden Layer $\in \mathbb{R}^8$ Hidden Layer $\in \mathbb{R}^4$ Output Layer $\in \mathbb{R}^1$

Neural networks
Representation



"2-layer Neural Net", or
"1-hidden-layer Neural Net"



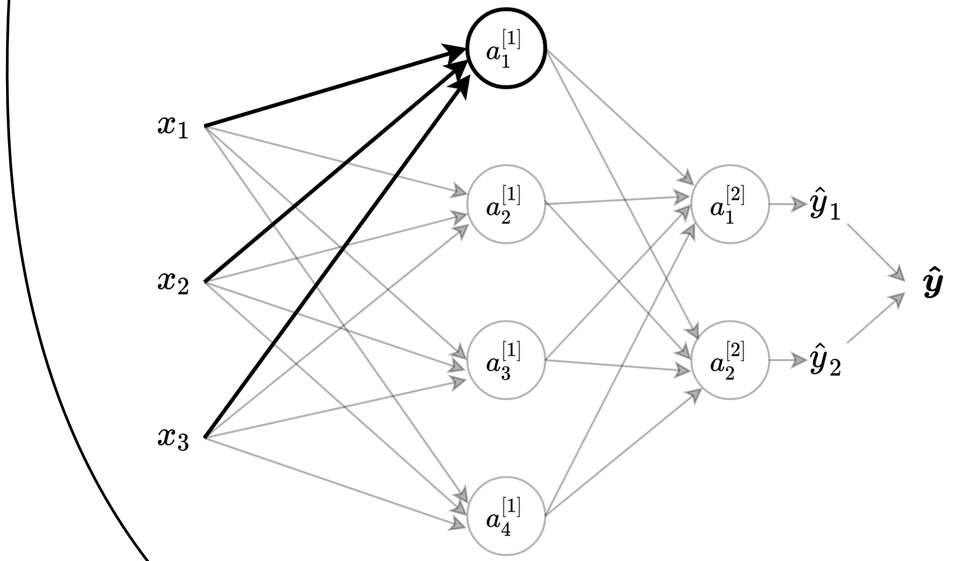
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

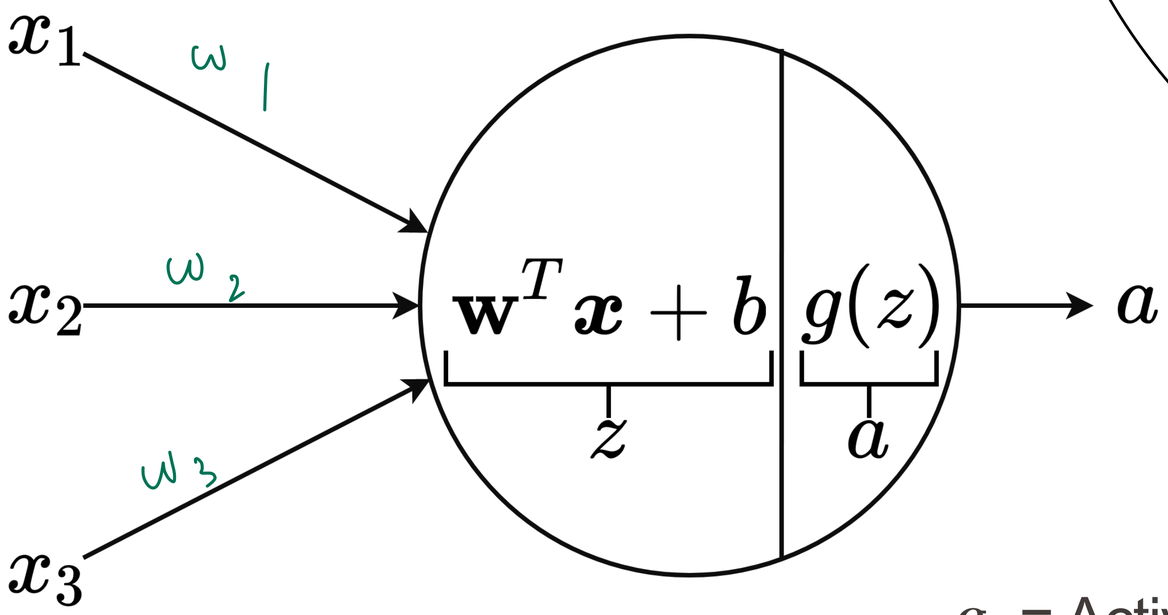
"Fully-connected" layers

Each neuron of a layer is connected
to all neurons of the following layer

Neural networks

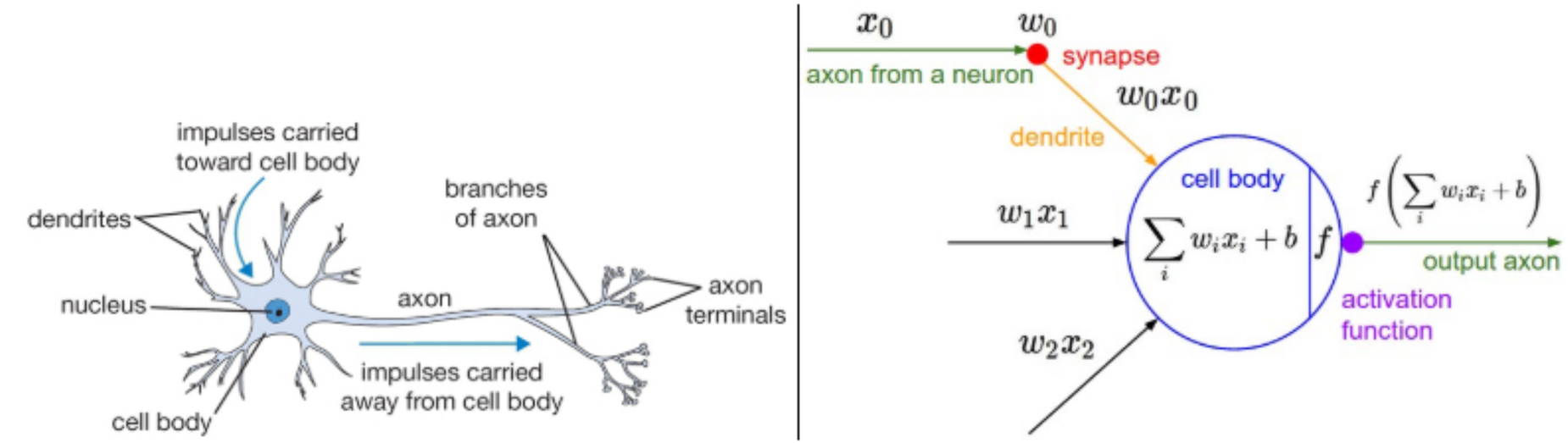
Inside a neuron





g = Activation function

Connections to biological neurons



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

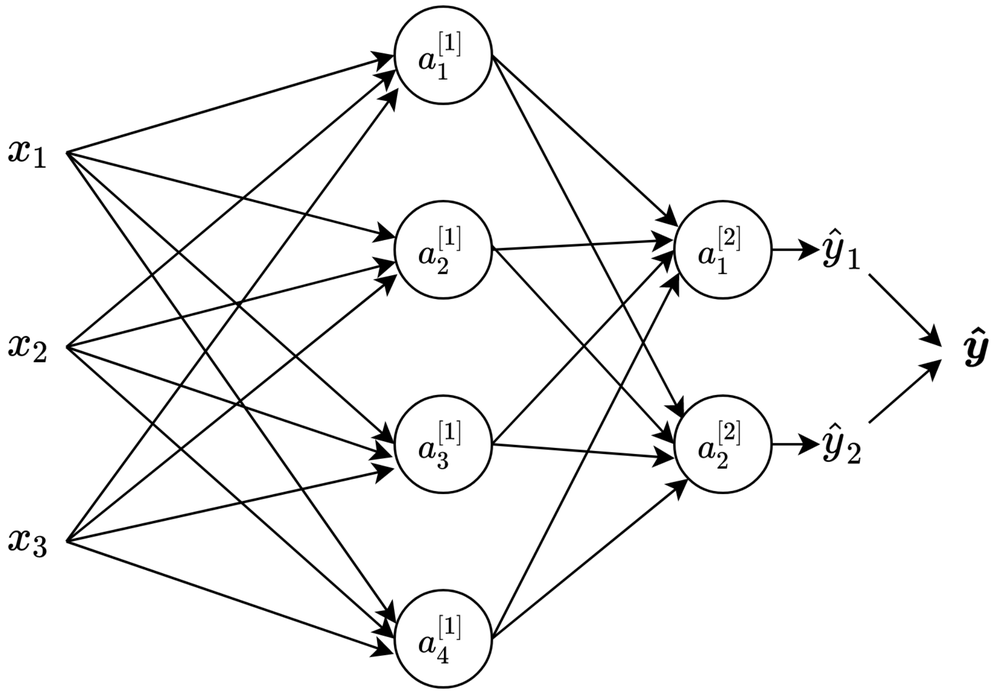
Source: towardsdatascience.com

Neural networks
Representation

$a_i^{[l]}$ ← Layer l

$a_i^{[l]}$ ← Node in layer i

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$ ← Shape (3, 1)



Neural networks
Representation

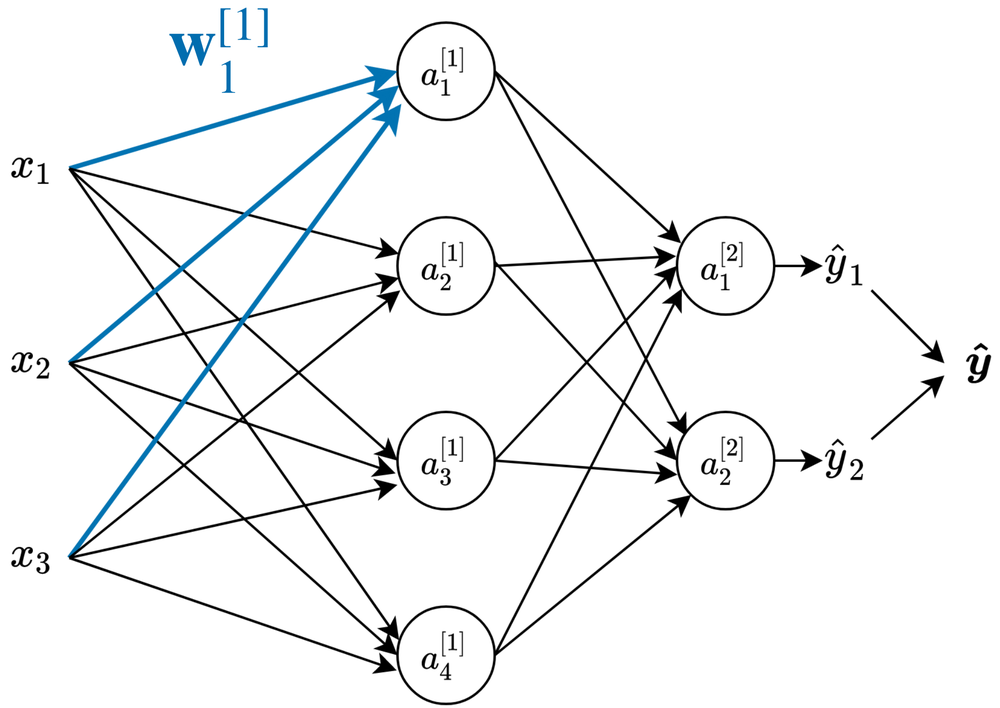
$a_i^{[l]}$ ← Layer l

$a_i^{[l]}$ ← Node i in layer l

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Weight vector for first node of first layer:

$$\mathbf{w}_1^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} \\ w_{1,2}^{[1]} \\ w_{1,3}^{1]} \end{bmatrix}$$



Neural networks
Representation

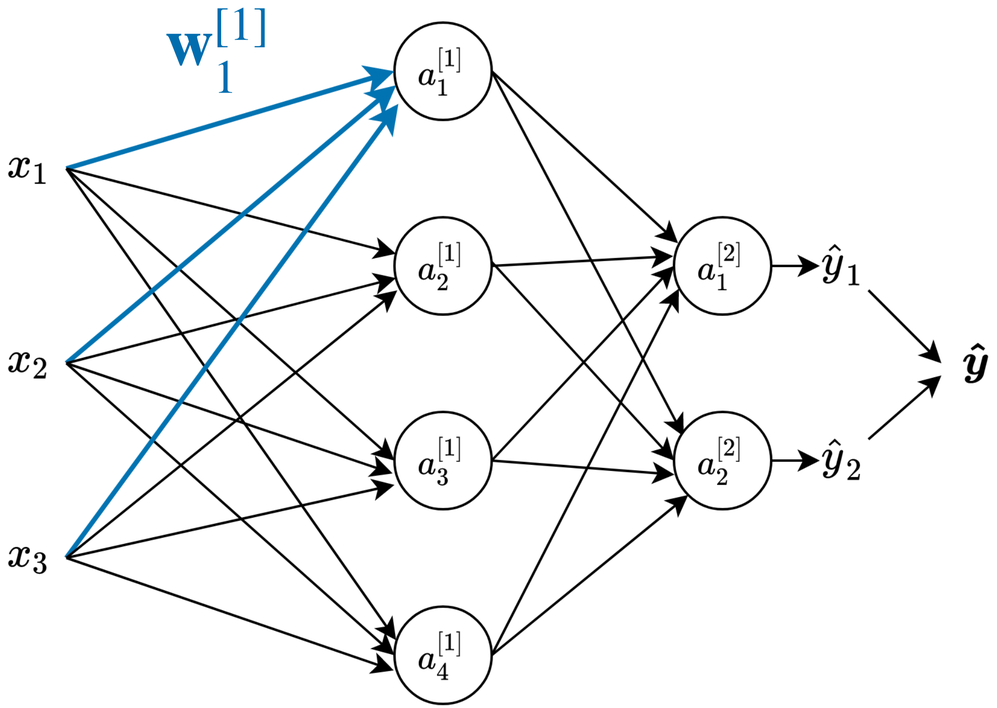
$a_i^{[l]}$ ← Layer

$a_i^{[l]}$ ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Weight vector for first node of first layer:

$$\mathbf{w}_1^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} \\ w_{1,2}^{[1]} \\ w_{1,3}^{1]} \end{bmatrix}$$



$$z_1^{[1]} = \mathbf{w}_1^{[1]T}\boldsymbol{x} + b_1^{[1]}$$
$$z_2^{[1]} = \mathbf{w}_2^{[1]T}\boldsymbol{x} + b_2^{[1]}$$
$$z_3^{[1]} = \mathbf{w}_3^{[1]T}\boldsymbol{x} + b_3^{[1]}$$
$$z_4^{[1]} = \mathbf{w}_4^{[1]T}\boldsymbol{x} + b_4^{[1]}$$

Neural networks
Representation

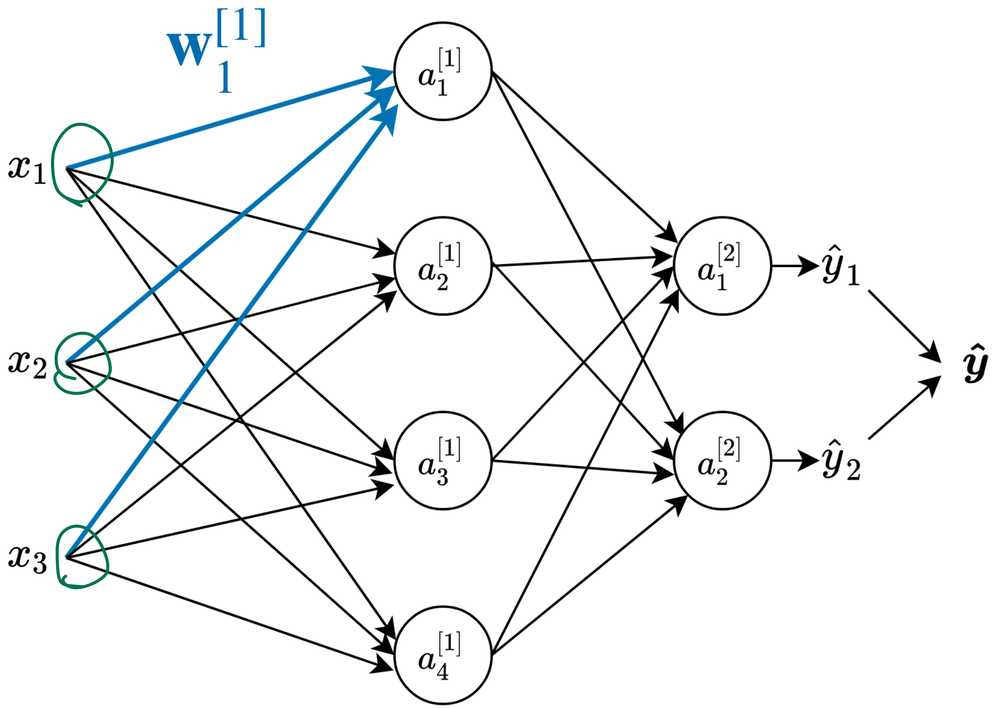
$a_i^{[l]}$ ← Layer

$a_i^{[l]}$ ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Weight vector for first node of first layer:

$$\mathbf{w}_1^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} \\ w_{1,2}^{[1]} \\ w_{1,3}^{1]} \end{bmatrix} \leftarrow \text{Shape (3, 1)}$$



$$z_1^{[1]} = \mathbf{w}_1^{[1]T}\boldsymbol{x} + b_1^{[1]}$$
$$z_2^{[1]} = \mathbf{w}_2^{[1]T}\boldsymbol{x} + b_2^{[1]}$$
$$z_3^{[1]} = \mathbf{w}_3^{[1]T}\boldsymbol{x} + b_3^{[1]}$$
$$z_4^{[1]} = \mathbf{w}_4^{[1]T}\boldsymbol{x} + b_4^{[1]}$$

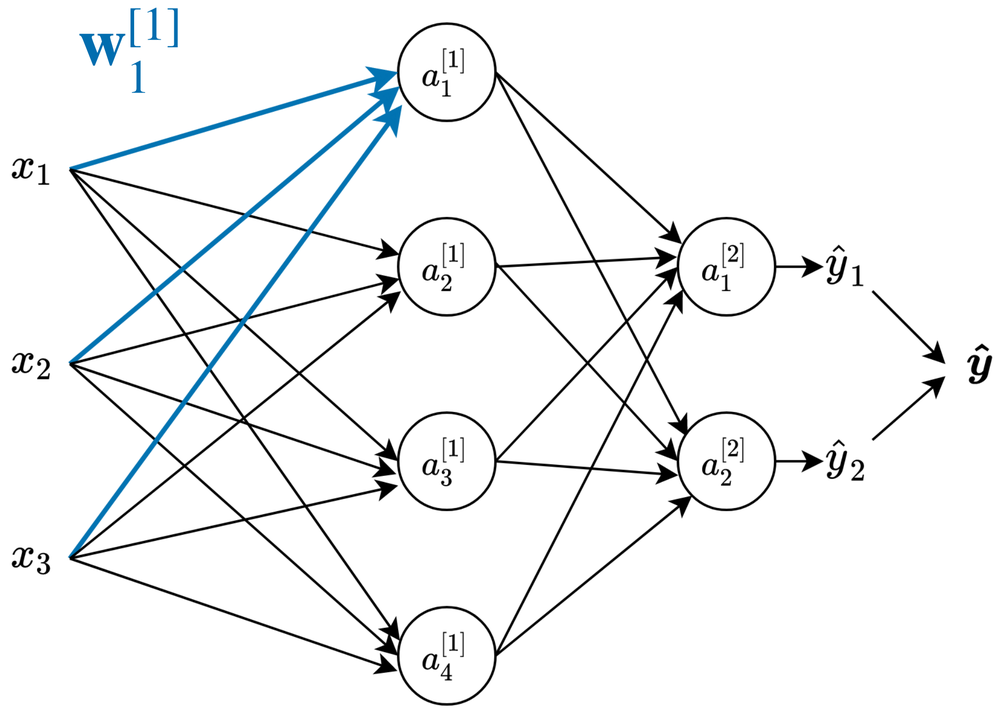
Apply activation →

$$a_1^{[1]} = g^{[1]}(z_1^{[1]})$$
$$a_2^{[1]} = g^{[1]}(z_2^{[1]})$$
$$a_3^{[1]} = g^{[1]}(z_3^{[1]})$$
$$a_4^{[1]} = g^{[1]}(z_4^{[1]})$$

Neural networks
Representation

$a_i^{[l]}$ ← Layer; ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$



Vector notation:

$$\mathbf{W}^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \mathbf{w}_1^{[1]} & \mathbf{w}_2^{[1]} & \mathbf{w}_3^{[1]} & \mathbf{w}_4^{[1]} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \in \mathbb{R}^{3\times 4} \qquad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} \in \mathbb{R}^4$$

Shape (3, 4)

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]} \in \mathbb{R}^4 \xrightarrow{\text{Apply activation}} \mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]}) \in \mathbb{R}^4$$

coordinate-wise application.

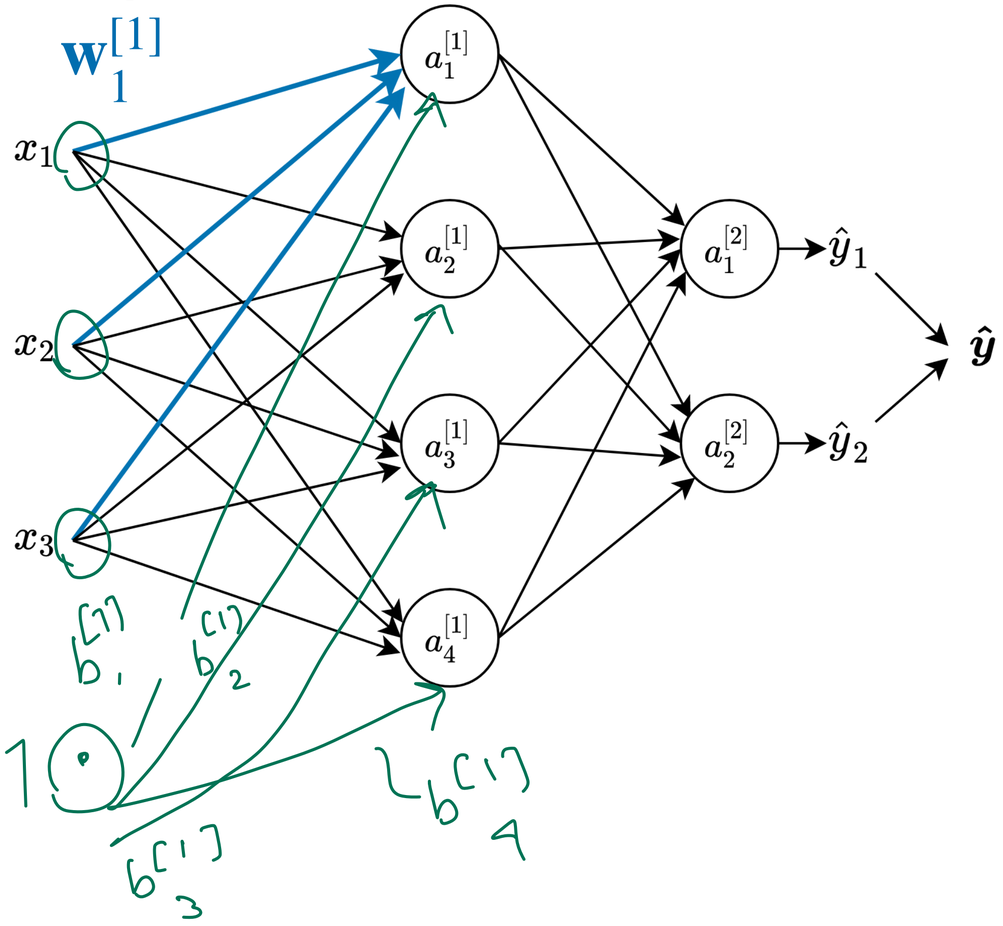
Neural networks

Note: we can write the constant input as a node

$a_i^{[l]}$ ← Layer

$a_i^{[l]}$ ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$



equivalent representation

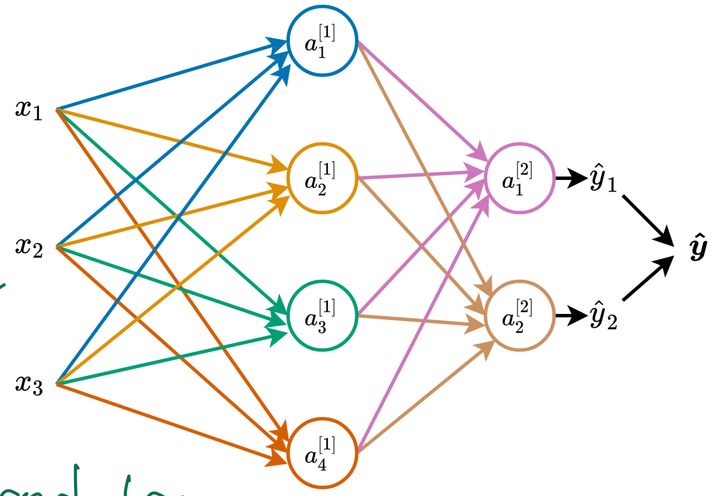
$$a_i^{[1]} = W_i^{[1]T} x + b_i^{[1]}$$

NN - Activation Function

Introduction

$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T} g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$

input to the second layer



What happens if we remove the activations?

$$\hat{\mathbf{y}} = \mathbf{W}^{[2]T}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]}$$

input to second layer

$$\hat{\mathbf{y}} = \mathbf{W}^{[2]T}\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{W}^{[2]T}\mathbf{b}^{[1]} + \mathbf{b}^{[2]}$$

output of second layer

Define $\mathbf{W}'^{T} = \mathbf{W}^{[2]T}\mathbf{W}^{[1]T}$ Define $\mathbf{b}' = \mathbf{W}^{[2]T}\mathbf{b}^{[1]} + \mathbf{b}^{[2]}$

$$\hat{\mathbf{y}} = \mathbf{W}'^{T}\boldsymbol{x} + \mathbf{b}'$$

We end up with a linear classifier

example

$g^{[1]}$ is sigmoid

$g^{[2]}$ is tanh ..

$$f_{w,b}(x) = \tanh\left(W^{[2]T}\,\sigma\left(W^{[1]T}x + b^{[1]}\right) + b^{[2]}\right)$$

Neural network - Activation functions

Introduction

To model a nonlinear predictor:

- Pass the output of each neuron through a nonlinear function, called activation function
- Connection to neuron firing in brain

Some well-known activation functions:

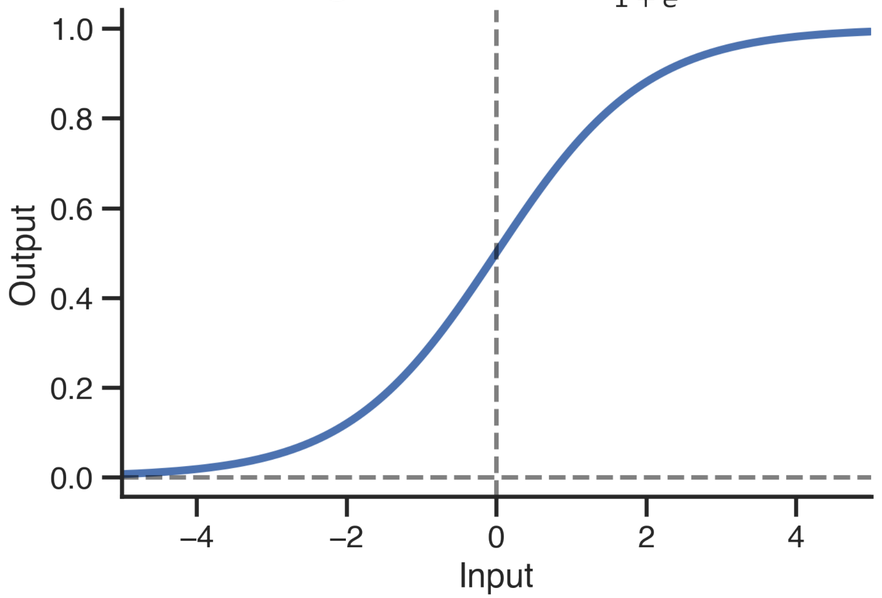
- Sign (used in theory but not in practice)
- Sigmoid
- Tanh
- ReLU/Leaky ReLU

$$\text{sign}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

we will look at now

NN - Activation functions
Overview

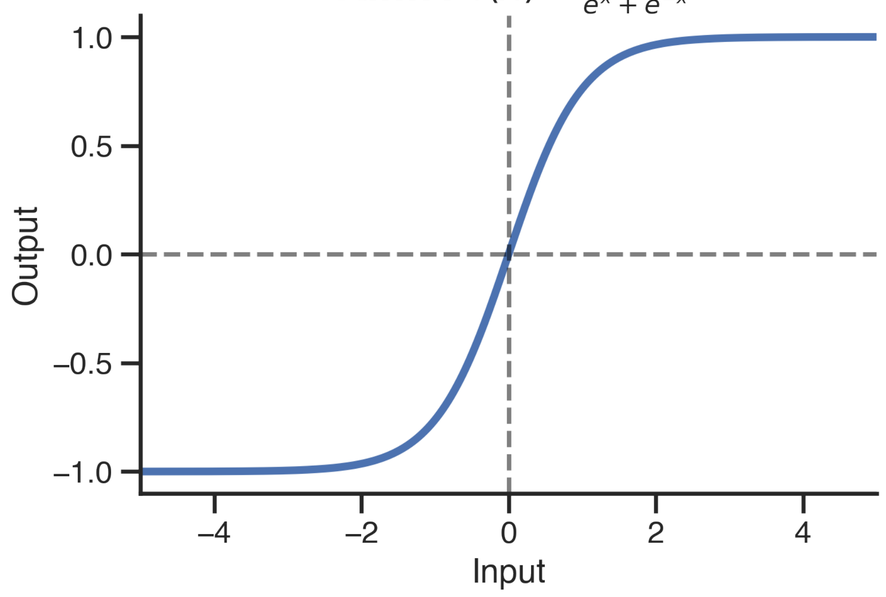
Sigmoid : $f(x) = \frac{1}{1+e^{-x}}$



Sigmoid (σ):

- Squashes input in a [0, 1] range
- Approximately nullifies gradient (for “large” positive or negative inputs) -> **vanishing gradient problem**
- rarely used except for final layer of binary classification network

Tanh : $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

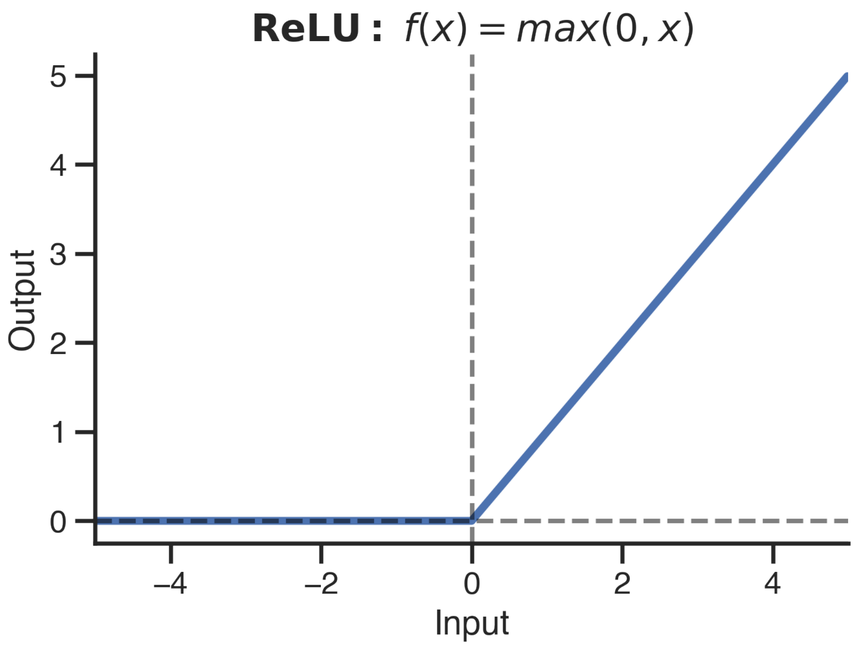


Tanh:

- Squashes input in a [-1, 1] range
- Like sigmoid, nullifies gradient (for “large” positive or negative inputs)
- Zero-centered, preferable over sigmoid as an activation
- Rarely used in practice (ReLU is more popular)

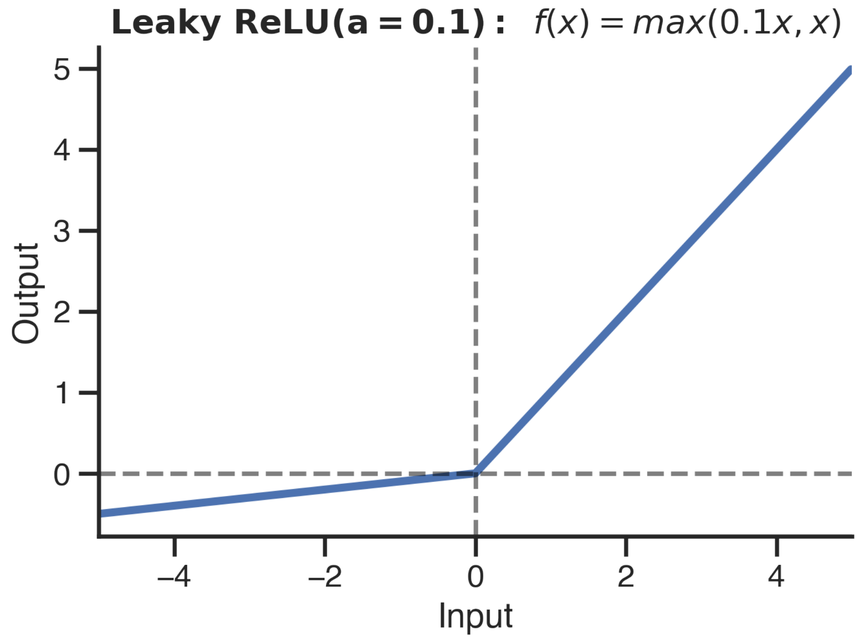
NN - Activation functions

Overview



Rectified Linear Unit (ReLU):

- Easily computed, simple gradient
- Greatly accelerates convergence of gradient descent
- Saturates in only one direction, suffers less from vanishing gradient problem
- commonly used



Leaky ReLU:

- Attempts to fix “dying ReLU” problem by having a small negative slope for x < 0.
- Leaky ReLU and other ReLU variants (ELU, SELU, GELU, Swish, etc…) are sometimes used over ReLU

NN - Activation functions

Derivatives

(verify)

Sigmoid:

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

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Tanh:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^{2}(x)$$

Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} = \max(0,\, x)$$

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Note: Derivative of ReLU is undefined for $x = 0$. By convention, it is set to 0.

Neural network predictor

- Once we determine the number of layers, the width and the activation function of each layer, we get the predictor:
  - Note: in a given layer, we use same activation function. Across layers, often we use same activation function
- For regression with $y \in \mathbb{R}^m$, often the last layer does not have an activation function
  - $z^L = W^{L^T} a^{L-1} + b^L \in \mathbb{R}^m$, where $W^L \in \mathbb{R}^{n_{L-1} \times m}$
- For classification in K categories, often the last layer is set to softmax
  - $z^L = \text{softmax}(a_1^{L-1}, \ldots, a_{n_{L-1}}^{L-1})$, where $n_{L-1} = K$

Training: Determining the neural network predictor

- Choose the neural network architecture

  # layers, # nodes per layer, activation function per layer

- Choose a loss function
  - For classification cross-entropy loss, $L(w,b) = \frac{-1}{N} \sum_{i=1}^{N} \sum_{c} 1\{y^i = c\} \log \frac{e^{z_c^i}}{\sum_{j=1}^{K} e^{z_j^i}}$
  - For regression mean-squared error loss, $L(w,b) = \frac{1}{N} \sum_{i=1}^{N} (y^i - \hat{y}^i)^2$
- Minimize the loss function $\min_{w,b} L(w,b)$

Training neural networks

Forward pass of 2 layer NN (for a single example):

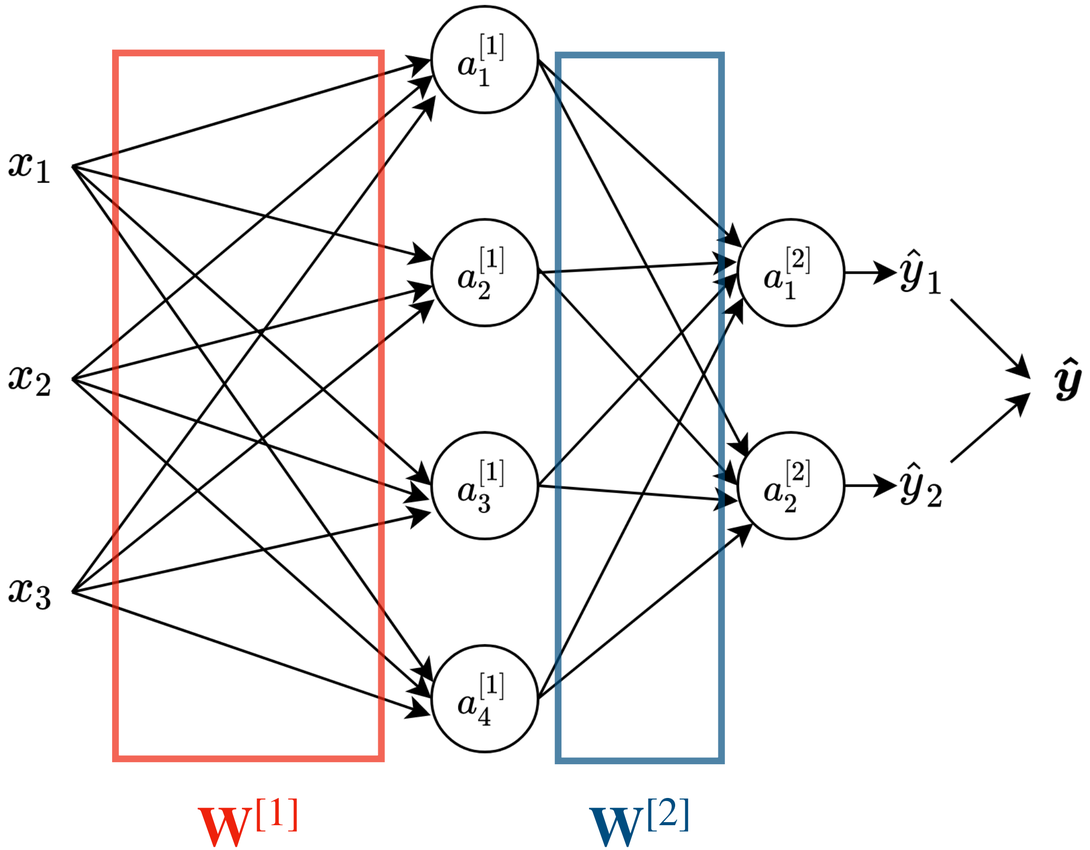
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$

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

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

$$\hat{\mathbf{y}} = \mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$

$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T} g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$



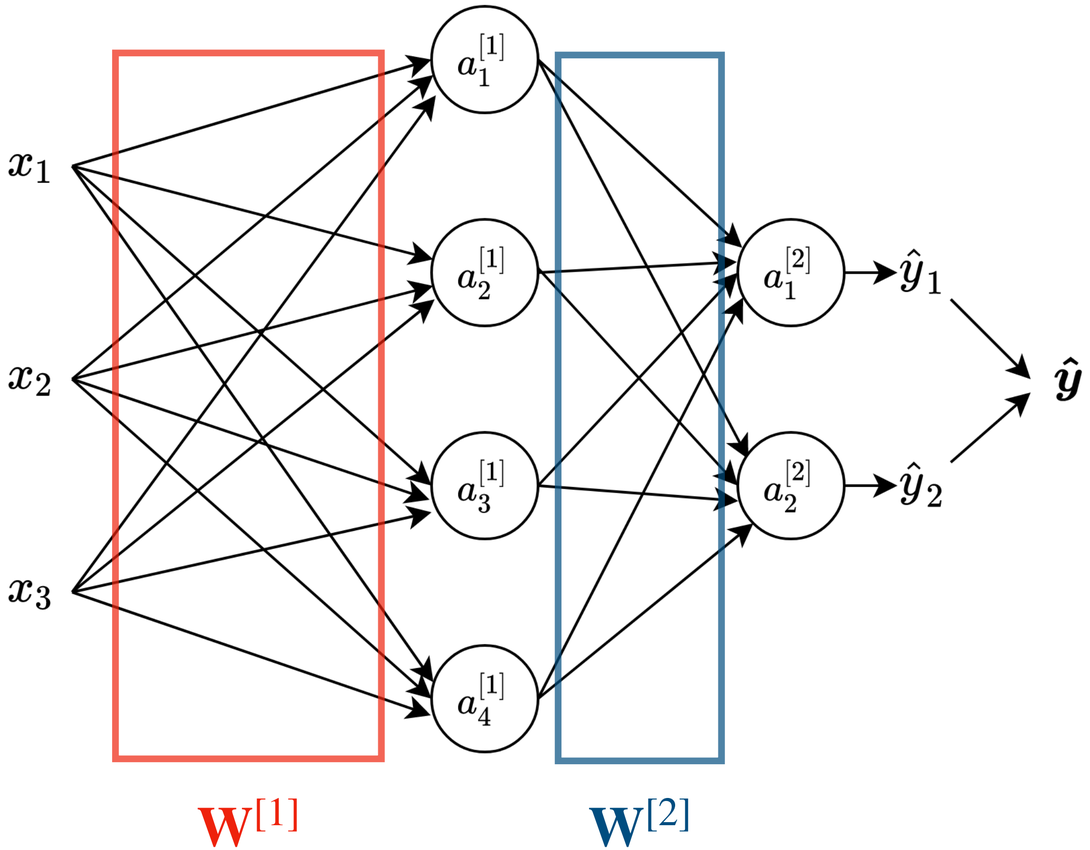
Neural networks
Training

Forward pass of 2 layer NN (for a single sample):

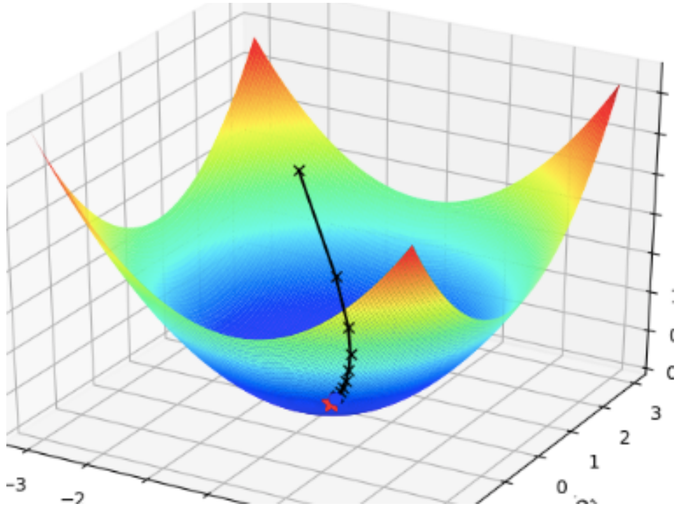
$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T} g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$

To train, we need a **loss function**: $L(\hat{\mathbf{y}}, \mathbf{y})$

Using that loss function, we want to update
$\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{W}^{[2]}, \mathbf{b}^{[2]}$
using **gradient descent**.



Gradient descent



Need to compute: $\frac{\partial L}{\partial \mathbf{W}^{[i]}}, \frac{\partial L}{\partial \mathbf{b}^{[i]}}$

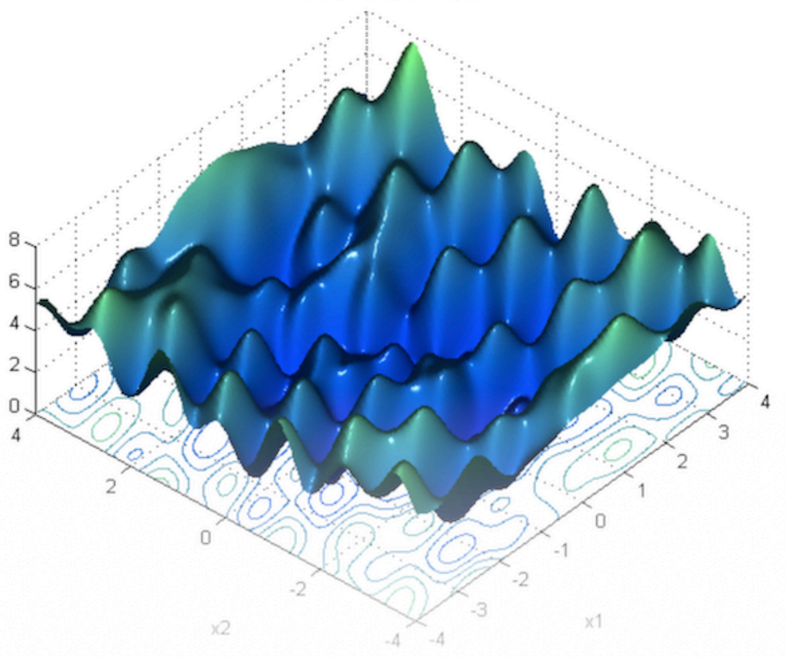
=> Gradient of loss with respect to weights and biases of each layer

Once gradients are computed,

update weights with:

- $\mathbf{W}_{t+1}^{[i]} := \mathbf{W}_{t}^{[i]} - \alpha_t \frac{\partial L}{\partial \mathbf{W}^{[i]}}(\mathbf{W}_t, \mathbf{b_t})$
- $\mathbf{b}_{t+1}^{[i]} := \mathbf{b}_{t}^{[i]} - \alpha_t \frac{\partial L}{\partial \mathbf{b}^{[i]}}(\mathbf{W}_t, \mathbf{b_t})$

where α_t is the learning rate



Neural networks
Forward / Backward pass

Forward pass: Compute the output of a neural network for a given input

Backward pass: Compute derivatives of the network parameters given the output

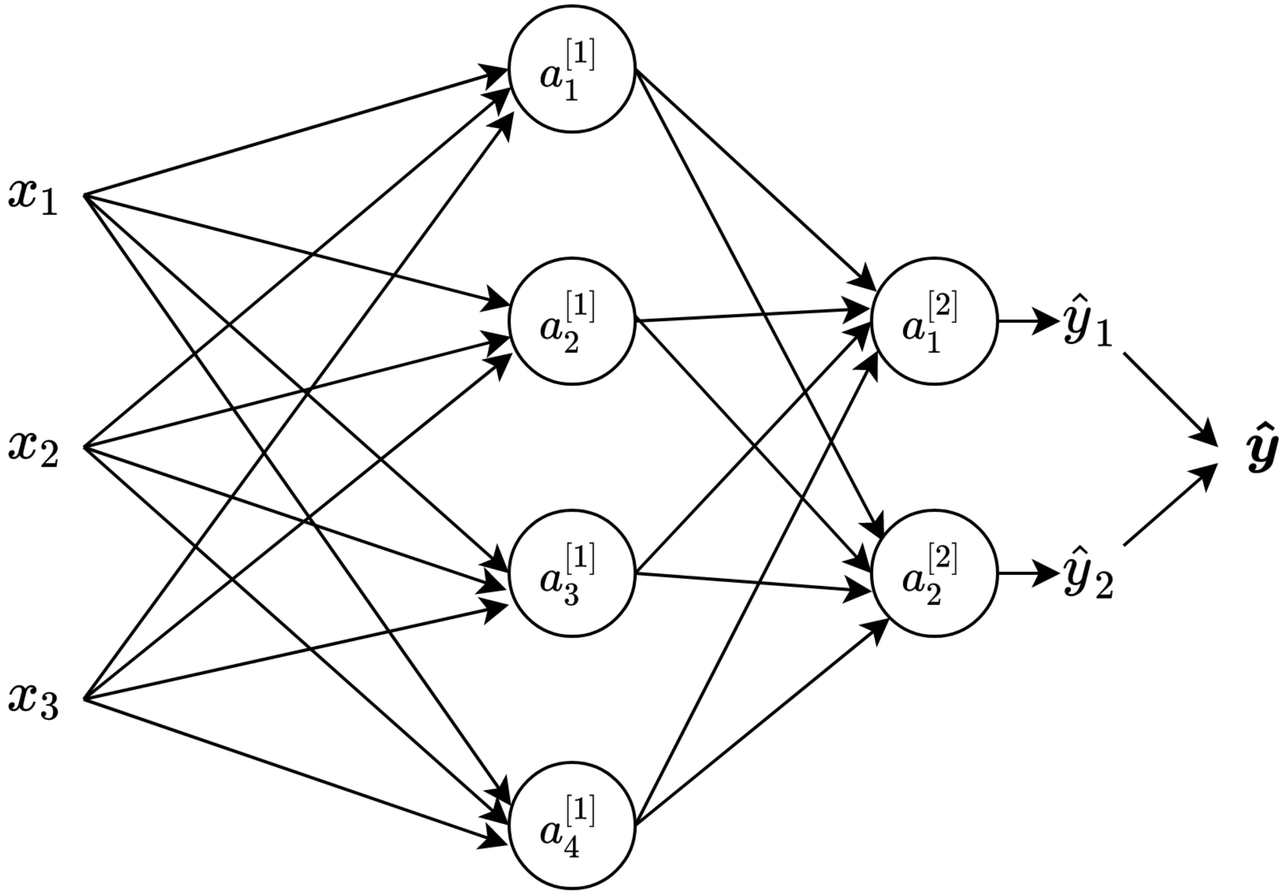
During **training**, you need both the **forward** pass and the **backward** pass.

During **prediction** (inference**)**, you only need the **forward** pass.

Inference: the process of using a trained machine learning model for prediction

Neural networks

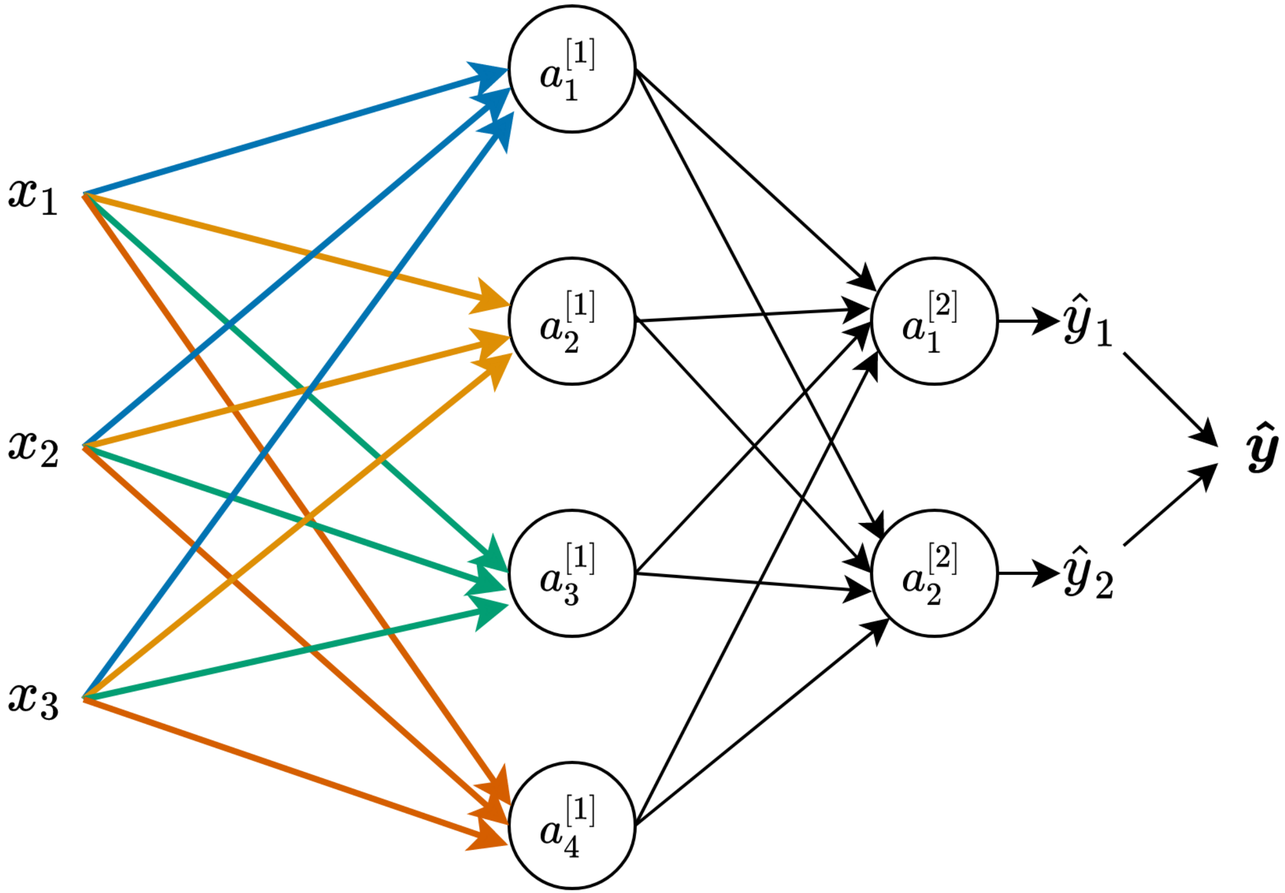
Forward pass



Neural networks

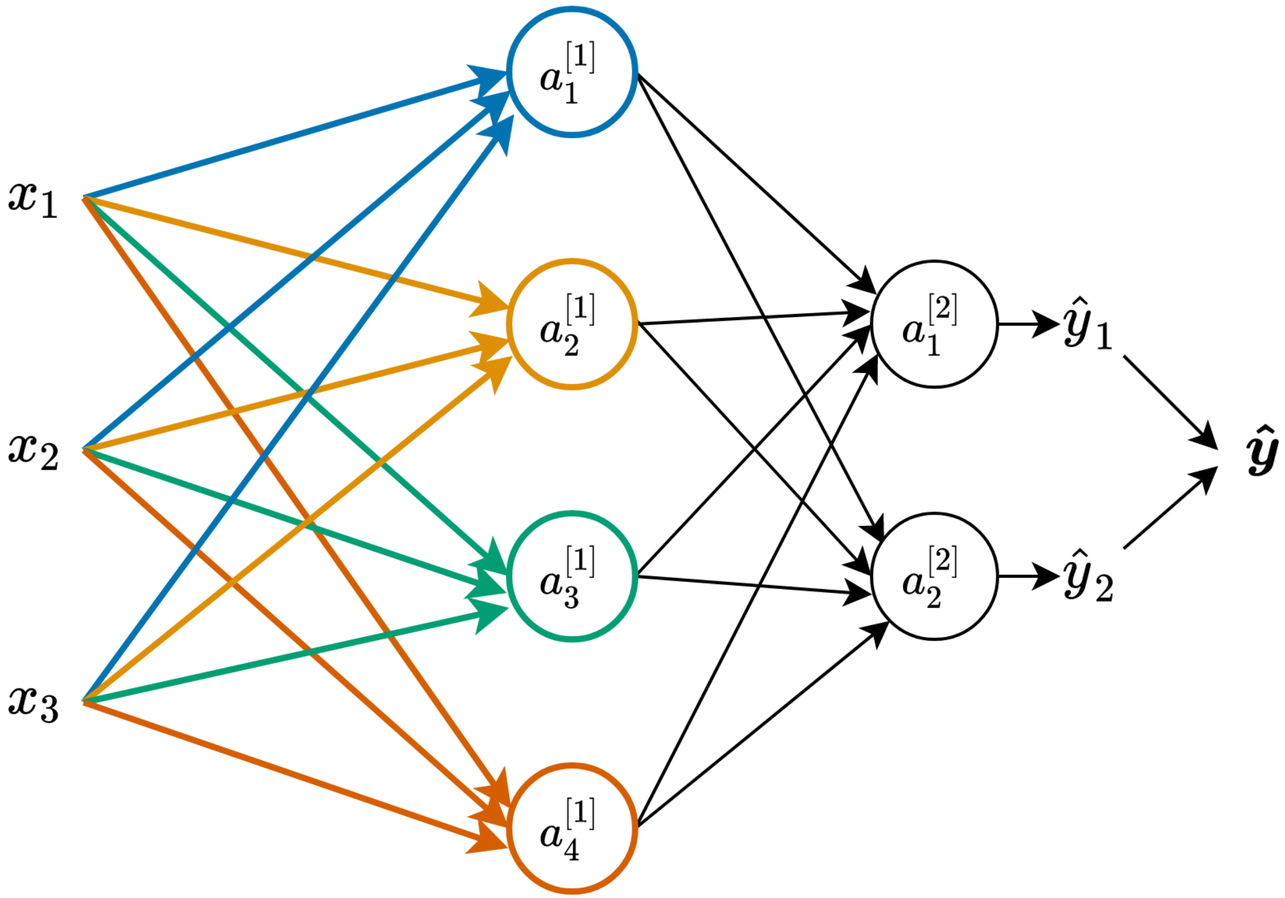
Forward pass

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$



Neural networks
Forward pass

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$



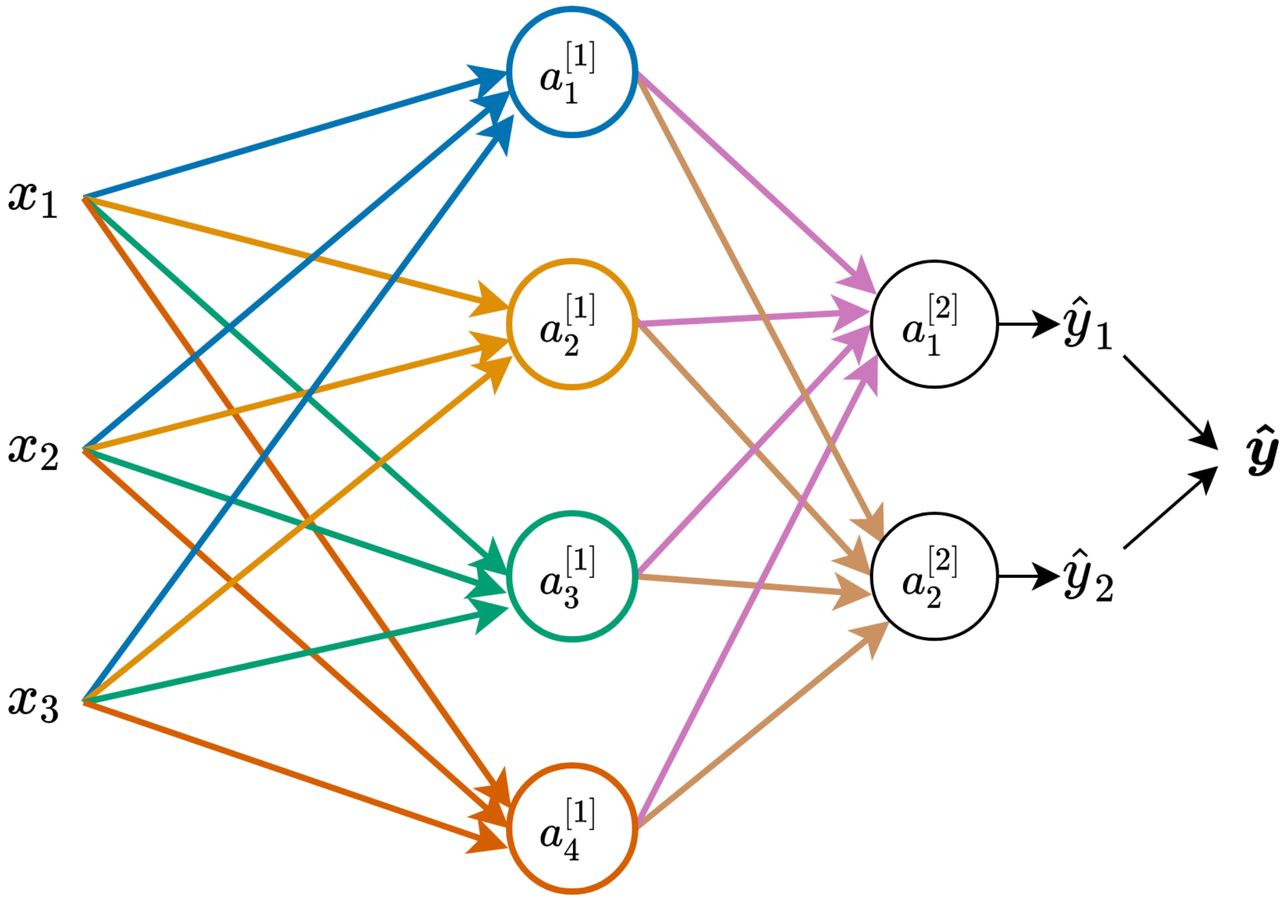
Neural networks

Forward pass

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$

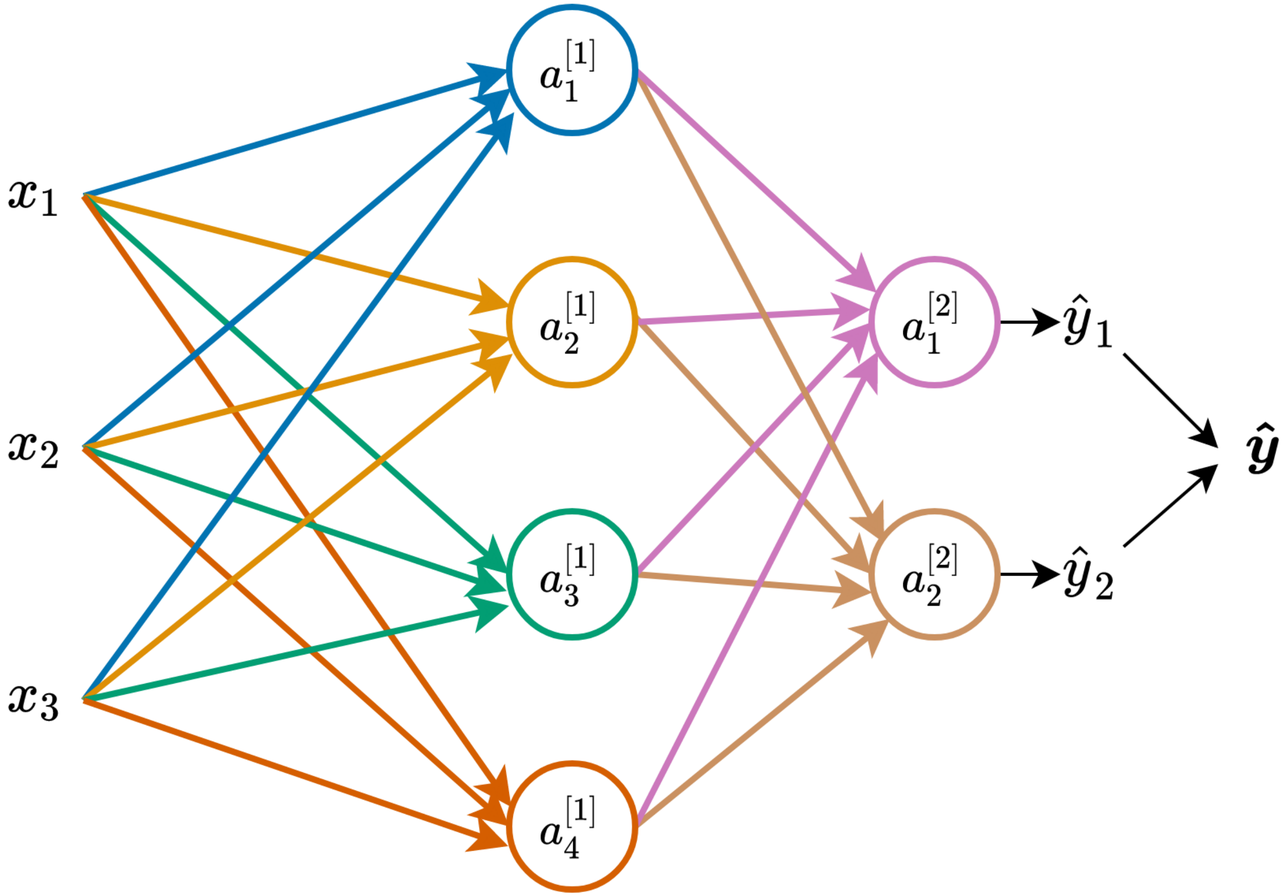
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

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



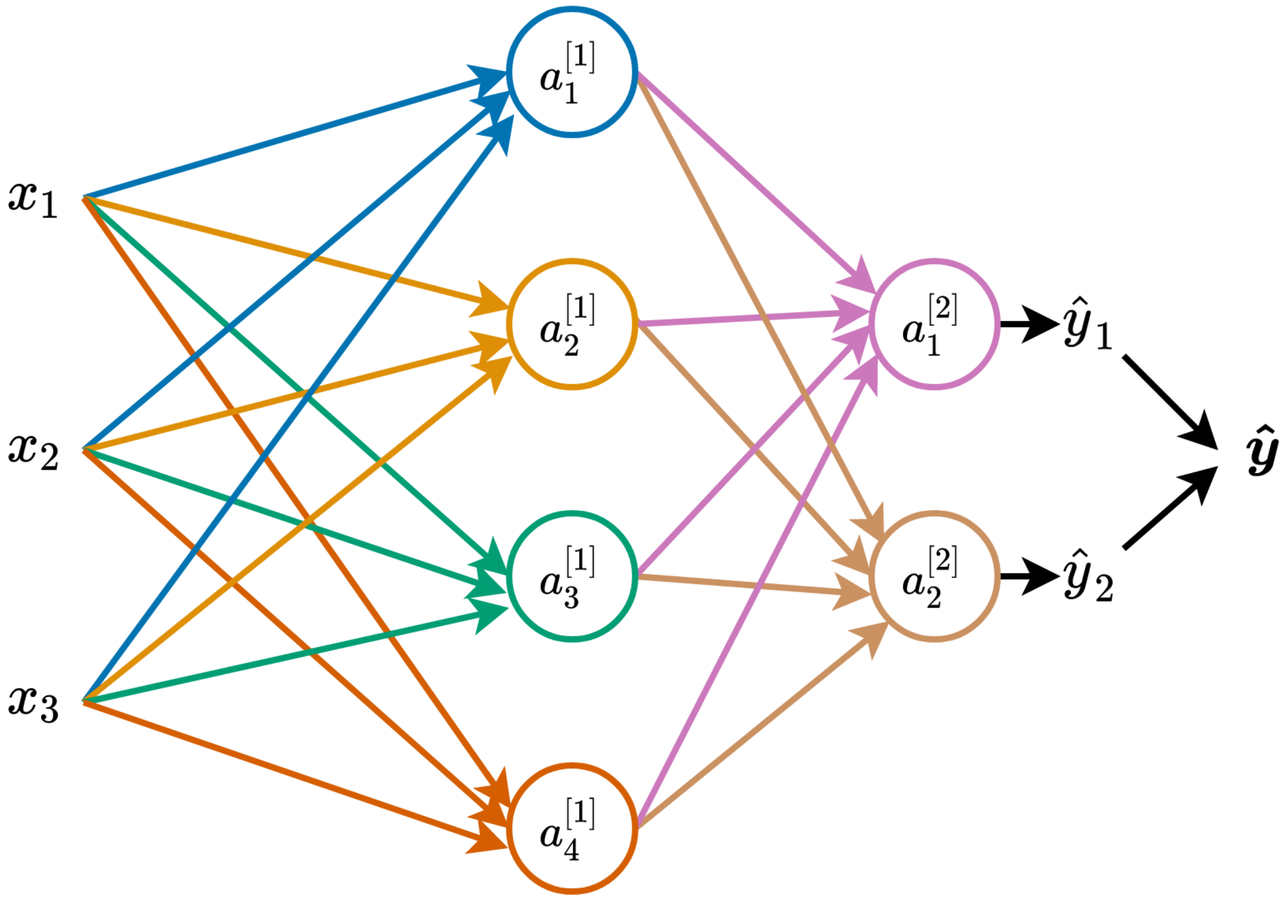
Neural networks
Forward pass

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$
$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]T}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$



Neural networks
Forward pass

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$
$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]T}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$
$$\hat{\mathbf{y}} = \mathbf{a}^{[2]}$$



Neural networks
Forward pass

Forward pass of this 2-layer NN:

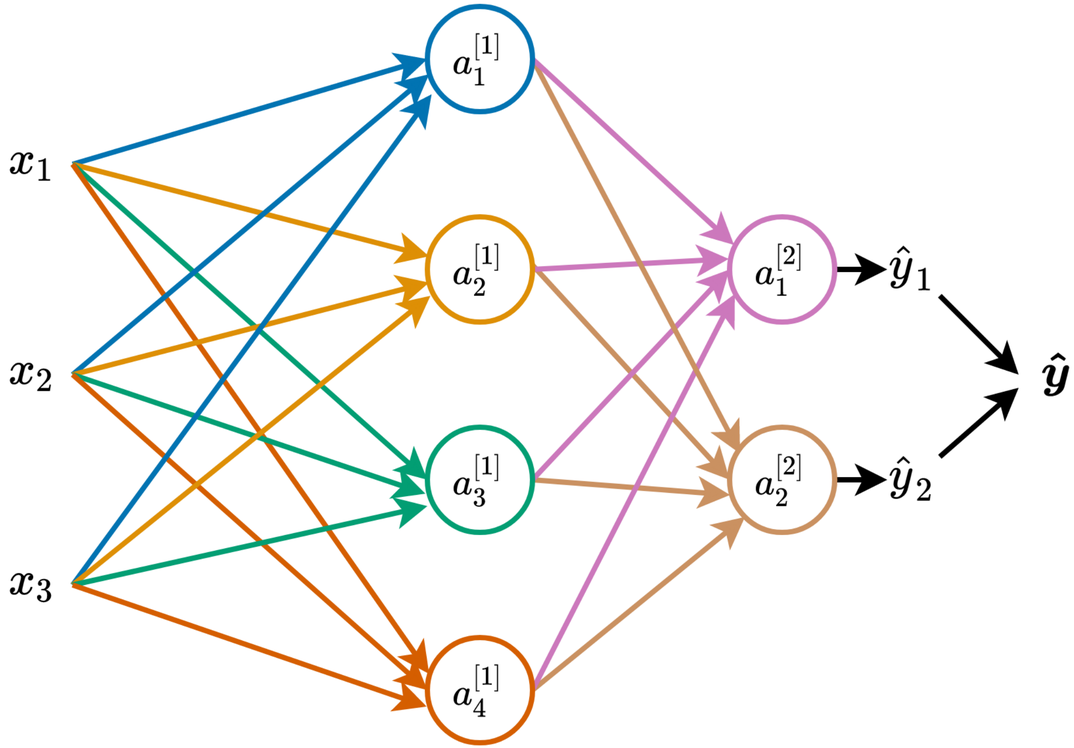
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$

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

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

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

$$\hat{\mathbf{y}} = \mathbf{a}^{[2]}$$



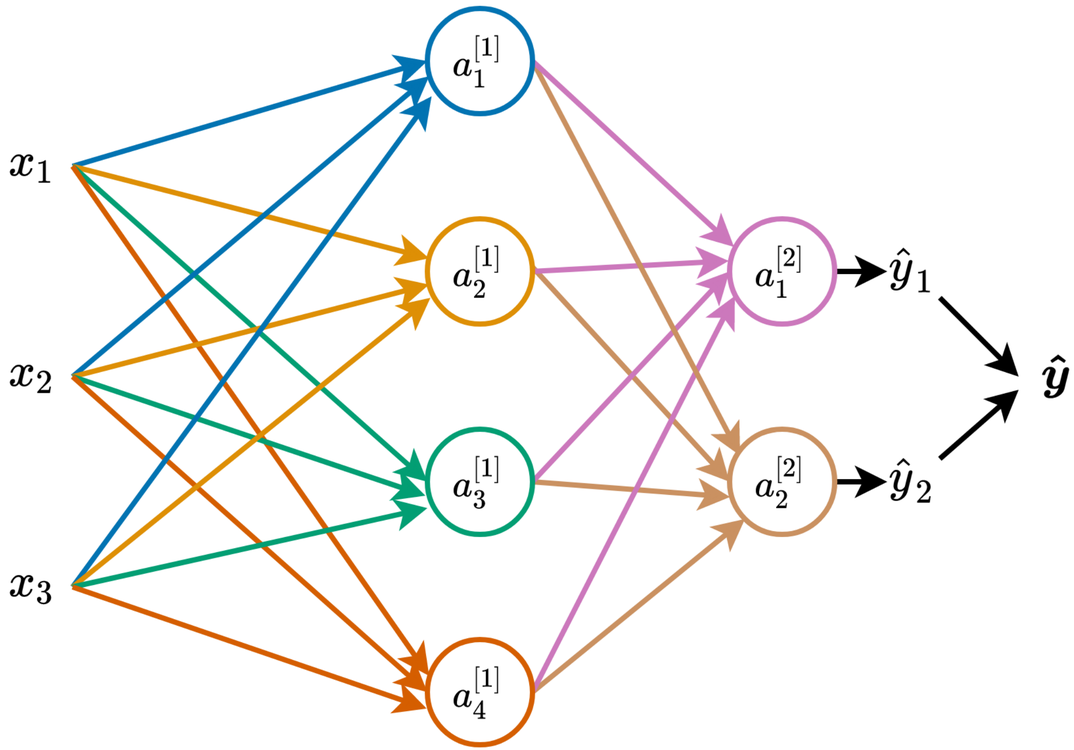
Rewriting it in one equation:

$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T}g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$

Neural networks
Backward pass

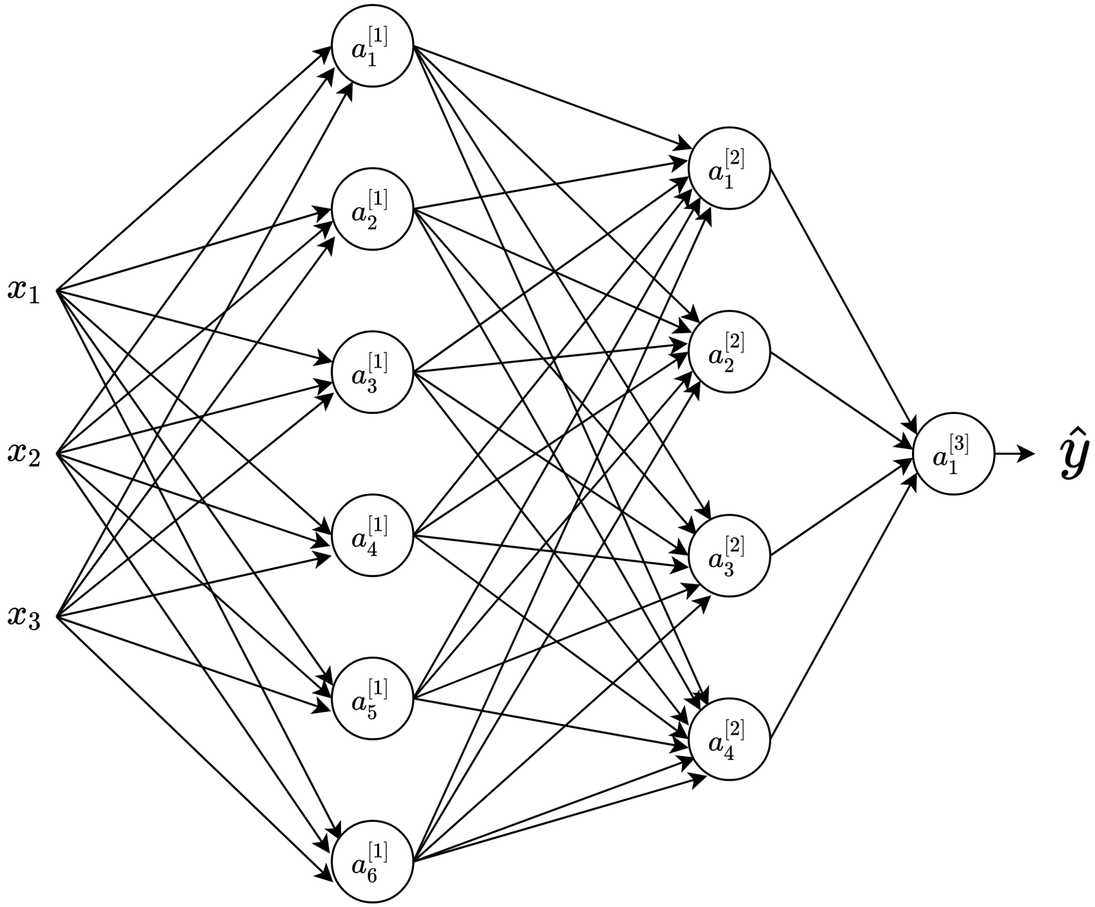
Compute gradients starting from the last layer and propagating backwards

Read: example computation in Section 5.2 of the book Understanding Machine Learning (see Moodle)



Deep learning frameworks

Implementing a simple neural network in PyTorch




```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(3, 6)
        self.fc2 = nn.Linear(6, 4)
        self.fc3 = nn.Linear(4, 1)

    def forward(self, x):
        # First layer
        x = self.fc1(x)
        x = F.relu(x)
        # Second layer
        x = self.fc2(x)
        x = F.relu(x)
        # Output layer
        x = self.fc3(x)
        return x
```

Summary and exercise hour announcements

Feature engineering

- How to best represent your numerical/ordinal/categorial data for the computer
- Need some insight into the data to address it

Neural networks

- Definition, activation functions
- Loss function and training

Exercise hour

- This week: problem set on logistic regression and neural networks
- 15.10 exercise hour: quiz 1 (optional)