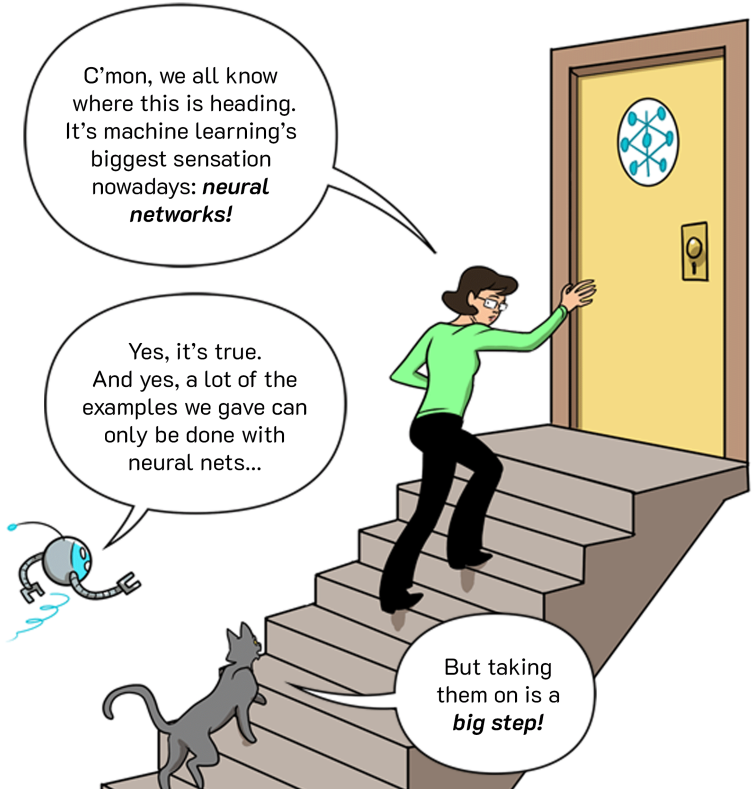
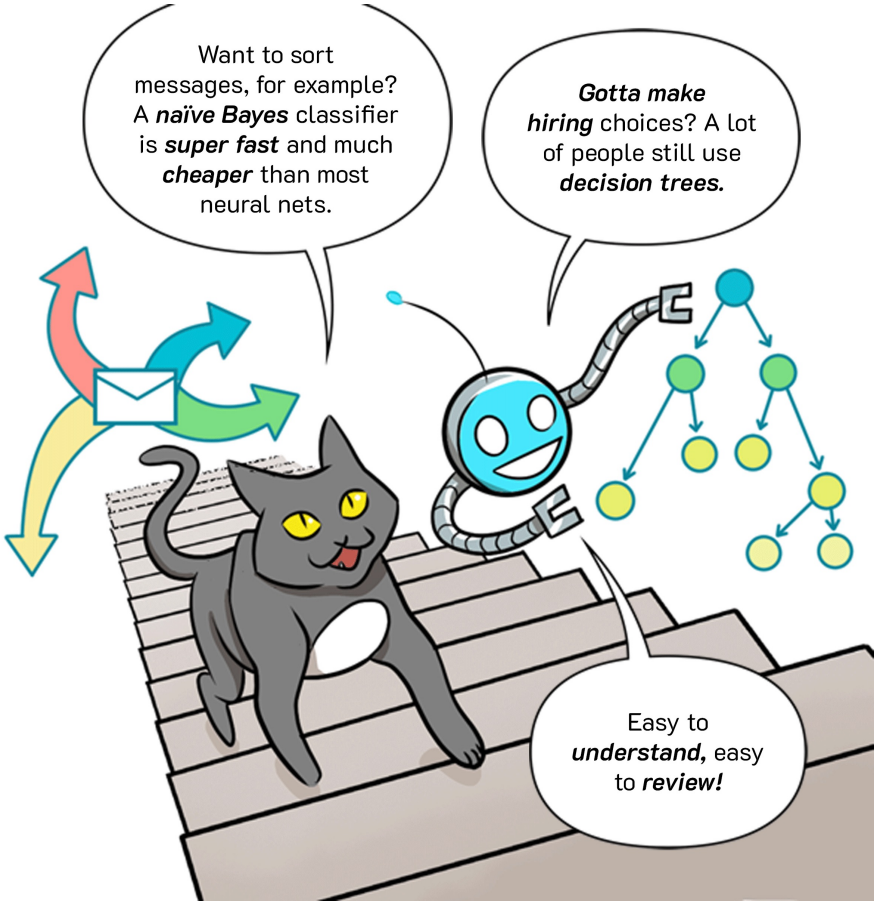
Introduction to
Artificial Neural Networks
Part I

Dr. Andrea Santamaria Garcia, Chenran Xu

Institute of Beam Physics and Technology (KIT)





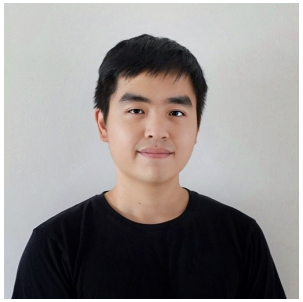
Comic source: https://cloud.google.com/products/ai/ml-comic-1

Comic source: https://cloud.google.com/products/ai/ml-comic-1

AI4Accelerators Team at KIT

@ansantam

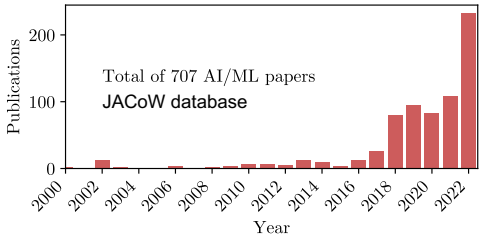
Dr. Andrea Santamaria Garcia
andrea.santamaria@kit.edu
Team leader



Chenran Xu
chenran.xu@kit.edu
Doctoral student

We are accelerator physicists researching machine learning applications for particle accelerators

https://www.ibpt.kit.edu/team_AI.php



Increased interest in ML methods in the accelerator world!

Karlsruhe Institute of Technology

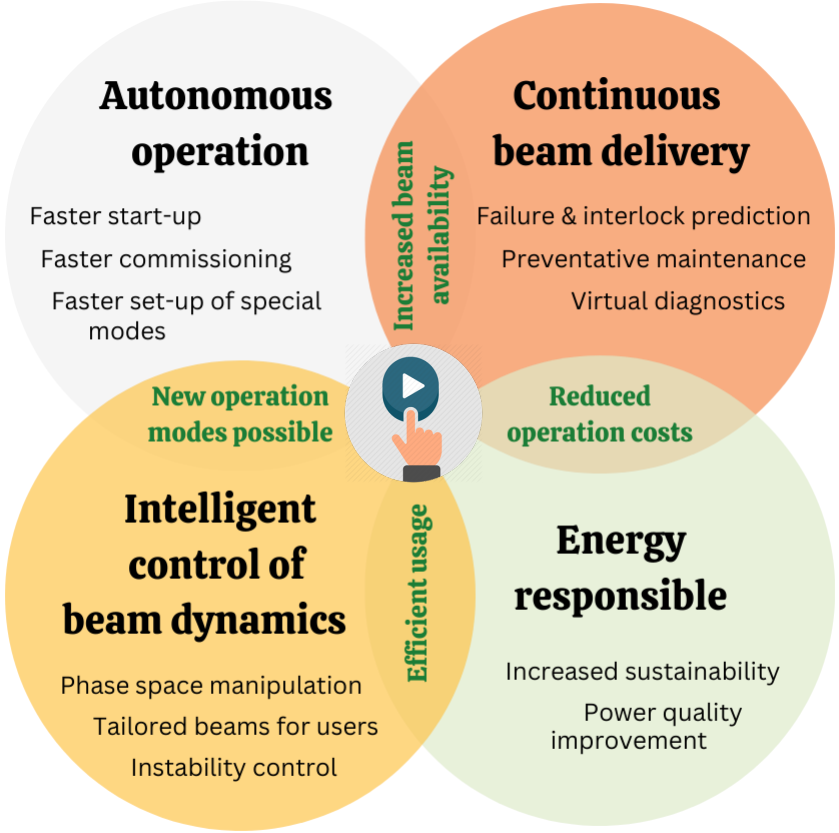
IBPT
Institute for Beam Physics and Technology

Task	Goal	Methods/Concepts	Examples[1]
Detection	Detect outliers and anomalies in accelerator signals for interlock prediction, data cleaning	• Anomaly detection • Time series forecasting • Clustering	• Collimator alignment • Optics corrections • SRF quench detection
Prediction	Predict the beam properties based on accelerator parameters	• Virtual diagnostics • Surrogate models • Active learning	• Beam energy prediction • Accelerator design • Phase space reconstruction
Optimization	Achieve desired beam properties or states by tuning accelerator parameters	• Numerical optimizers • Bayesian optimization • Genetic algorithm	• Injection efficiency • Radiation intensity
Control	Control the state of the beam in real time in a dynamically changing environment	• Reinforcement learning • Bayesian optimization • Extremum Seeking	• Trajectory steering • Instability control

[1] non-exhaustive

A. Santamaria Garcia et al, FLS23-TH3D3

A vision for future accelerators, driven by ML



Karlsruhe Institute of Technology

IBPT
Institute for Beam Physics and Technology

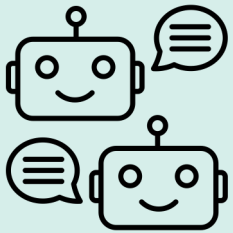
A. Santamaria Garcia et al, FLS23-TH3D3

Introduction

ARTIFICIAL INTELLIGENCE (AI)

Computers mimic human behaviour

- First chatbots
- Robotics
- Expert systems
- Natural language processing
- Fuzzy logic
- Explainable AI



Narrow AI

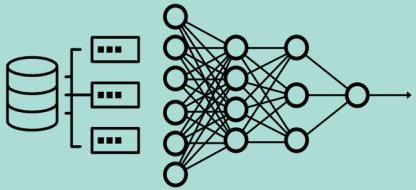
MACHINE LEARNING (ML)

Computers learn without being explicitly programmed to do so and improve with experience

Collection of **data-driven** methods / algorithms

Focused on **prediction** / **optimization** / **control** based on properties learned from data

Tries to **generalize** to unseen scenarios



DEEP LEARNING (DL)

Multi-layered neural networks perform certain tasks with high accuracy

- Speech/handwriting recognition
- Language translation
- Recommendation engines
- Computer vision

What are neural networks, conceptually?

They are combinations of mathematical functions that implement more complicated functions

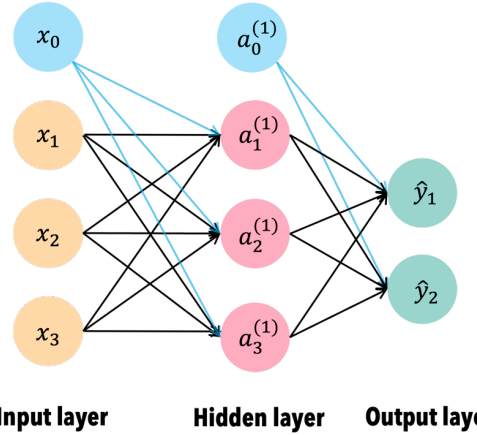
target data

$$y = f(x)$$ the model of the data

output input

$$\hat{y} = g^{(1)}(\boldsymbol{w}^{(1)} g^{(0)}(\boldsymbol{w}^{(0)} \boldsymbol{X}))$$

Nonlinear function weight



The structure of the network, the functions it uses, and its multiplicative parameters define the spaces of multivariate functions that can be implemented by a neural network

NN = a nonlinear function of (usually) many variables that depends on (usually) many parameters

Why neural networks?

Universal Approximation Theorem

A feedforward network with a single hidden layer and an unbounded nonlinear activation function is sufficient to approximate, to an arbitrary precision, any continuous function

Is this too good to be true 🧐?

of course, there are some caveats

The number of neurons might be infeasibly large

The resulting model might not generalize

The approximated function does not extrapolate outside of the approximation region

Thanks to machine-learning algorithms,
the robot apocalypse was short-lived.

https://www.smbc-comics.com/comic/rise-of-the-machines

Limitations of neural networks

NNs are excellent function approximators!

… when they have training data 🤗

and a lot of it

https://www.explainxkcd.com/wiki/index.php/1838:_Machine_Learning

- The universal approximation theorem does not tell you how to construct your network parameters.
- Neural networks are trained on data, but they might not converge on the correct parameters.

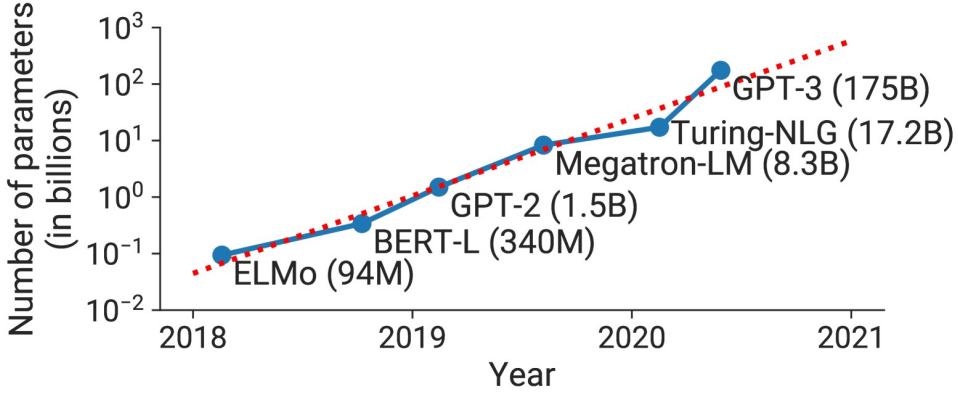
Machine learning ≠

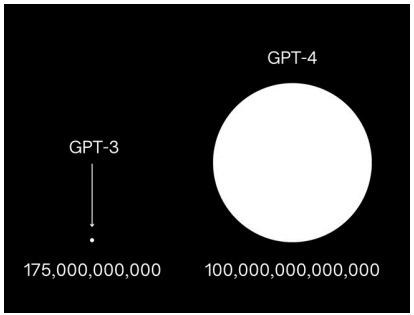
Although nowadays it starts looking like it!

A data hungry, very large parameter example: LLMs

In deep networks, performance keeps increasing with data quantity and network size

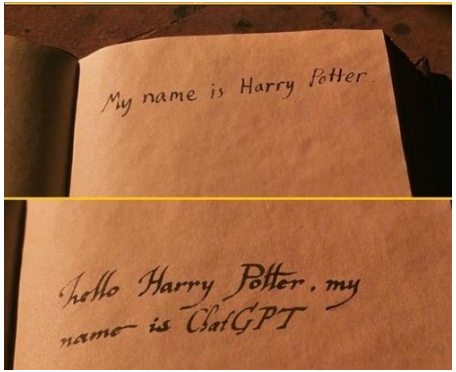
https://arxiv.labs.arxiv.org/html/2104.04473





Not sorcery, just a lot of resources!

Still impressive



Limitations of neural networks

+ Is there a scenario where linear regression outperforms neural networks?

- yes, when the linear regression assumptions are satisfied

The No Free Lunch theorem

Our model is a simplification of reality

⇩

Every learning algorithm must possess a certain bias

⇩

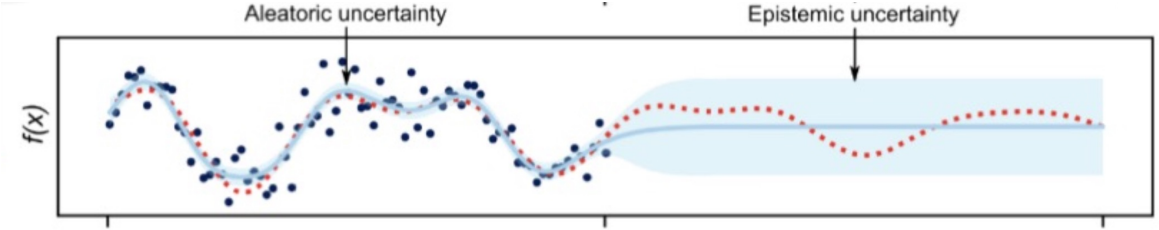
No single machine learning algorithm is universally the best-performing algorithm for all problems

- There are no universal solutions for ML problems
- With no strong assumptions on the input data, ML approaches perform similarly

https://arxiv.org/pdf/2202.04513.pdf
https://arxiv.org/pdf/2007.10928.pdf

Limitations of neural networks

Types of uncertainty



Aleatoric uncertainty

Data uncertainty

- Describes the confidence in the input data
- Is high when input data is noisy
- Cannot be reduced by adding more data
- Can be learned directly using neural networks
  - ➢ Probabilistic outputs, ensembling

Epistemic uncertainty

Model uncertainty

- Describes the confidence of the prediction
- Is high when missing training data
- Can be reduced by adding more data
- Challenging to estimate
  - ➢ Bayesian NN, evidential deep learning

from “Introduction to Deep Learning” 6.S19 (MIT)

MACHINE LEARNING

SUPERVISED LEARNING

Classification, prediction, forecasting

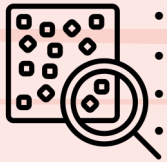
computer learns by example

- Spam detection
- Weather forecasting
- Housing prices prediction
- Stock market prediction

UNSUPERVISED LEARNING

Segmentation of data

computer learns without prior information about the data

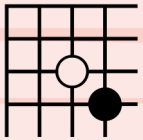


- Medical diagnosis
- Fraud (anomaly) detection
- Market segmentation
- Pattern recognition

REINFORCEMENT LEARNING

Real-time decisions

computer learns through trial and error



- Self-driving cars
- Make financial trades
- Gaming (AlphaGo)
- Robotics manipulation

Why machine learning now?

1943 — Perceptron
W. McCulloch, W. Pitts

1951 — "A Stochastic Approximation Method"
H. Robbins, S. Monro

1958 — Perceptron implementation
F. Rosenblatt

1959 — "Machine Learning"
A. Samuel

1969 — Perceptron (book)
M. Minsky, S. Papert

1973 — The Lighthill report
J. Lighthill

Pessimistic reviews & abandonment of connectionism

Connectionism: solving complex problems by combining many simple, generic elements

1974 — AI winter
Collapse of the billion-dollar AI industry

1993 — Handwritten digit recognition
Y. LeCun et al

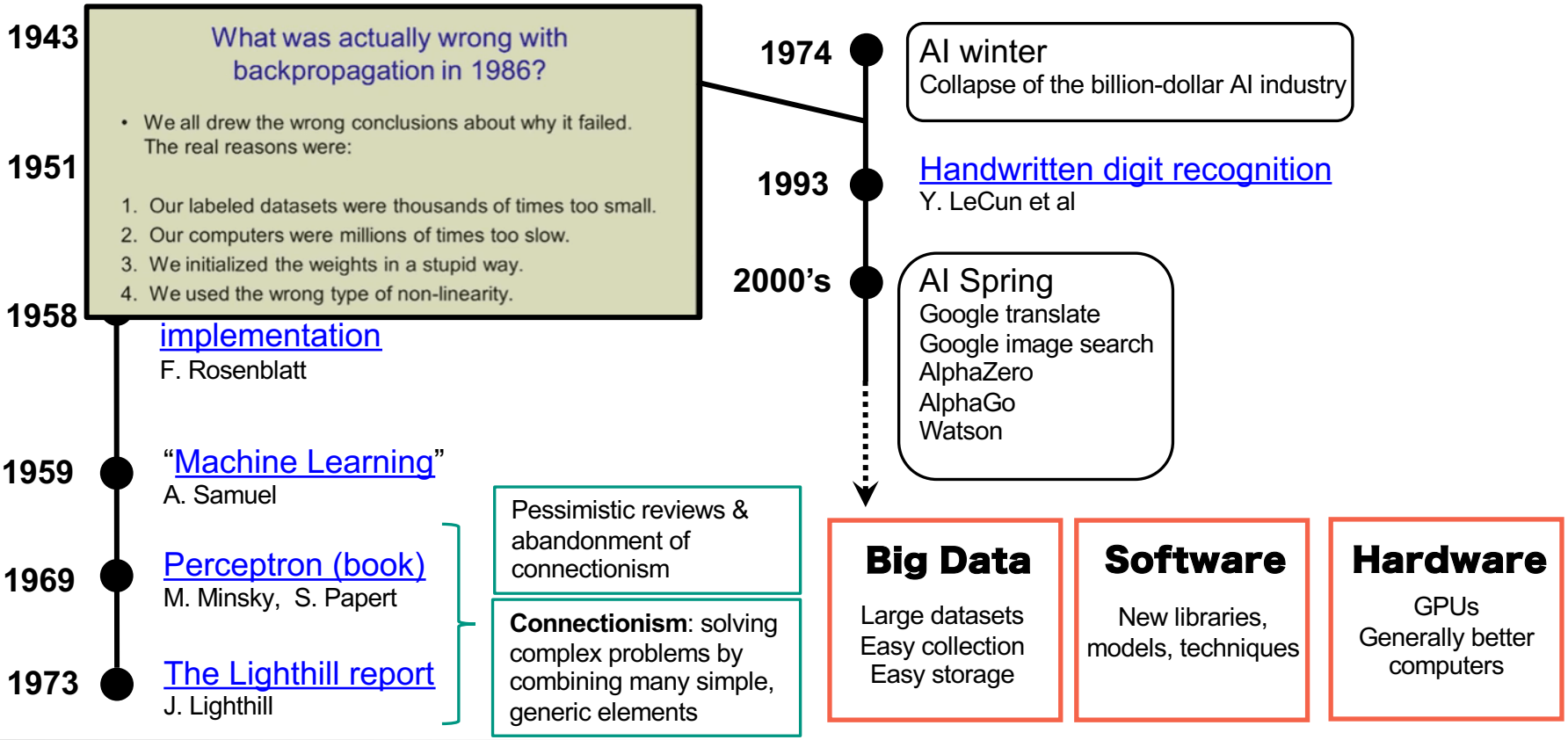
2000's — AI Spring
Google translate
Google image search
AlphaZero
AlphaGo
Watson

Big Data
Large datasets
Easy collection
Easy storage

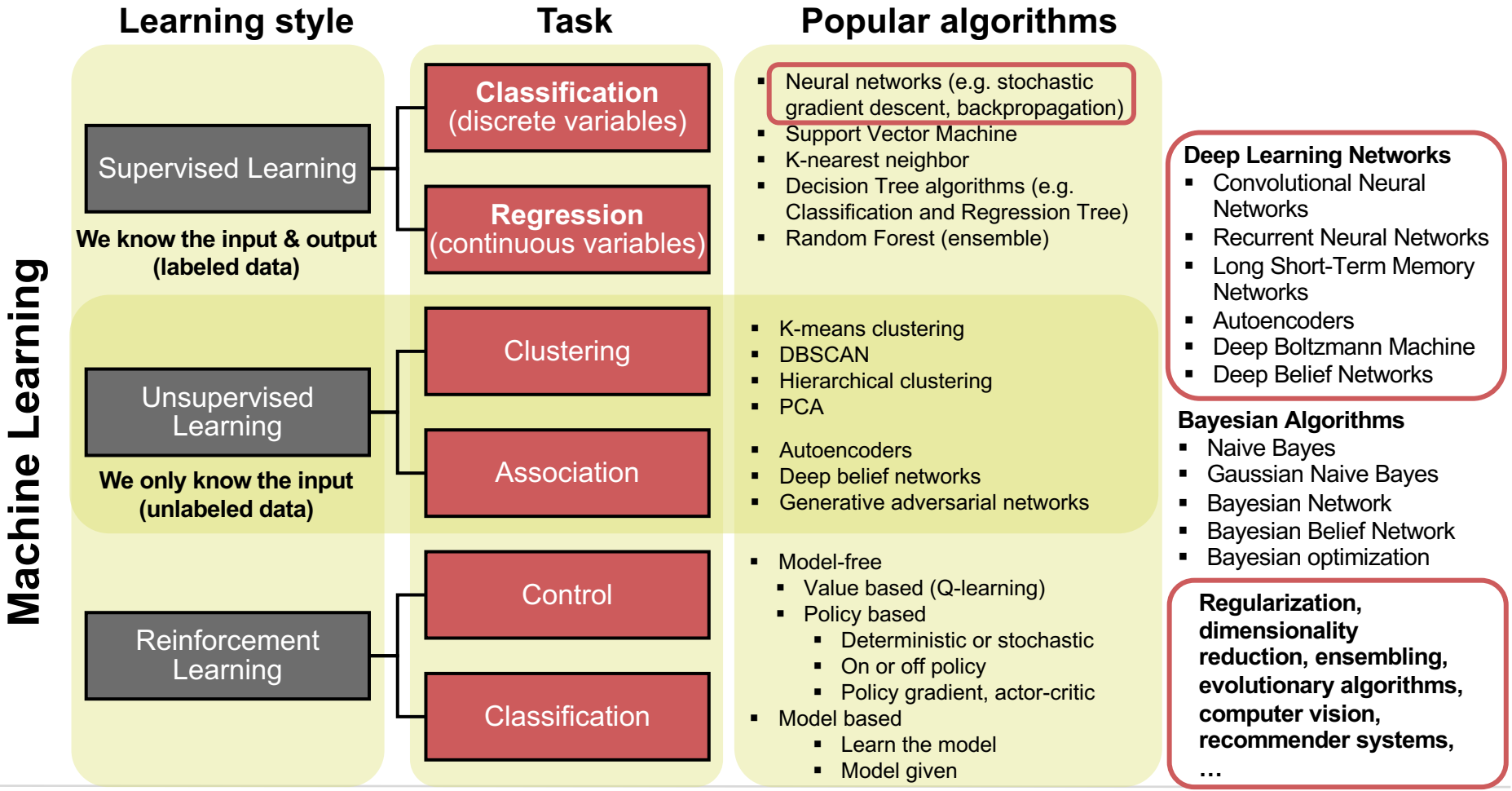
Software
New libraries, models, techniques

Hardware
GPUs
Generally better computers

Why machine learning now?



1986 slide by Geoff Hinton


Machine Learning
Learning style
Task
Popular algorithms
Supervised Learning
We know the input & output (labeled data)
Classification (discrete variables)
Regression (continuous variables)
Neural networks (e.g. stochastic gradient descent, backpropagation)
Support Vector Machine
K-nearest neighbor
Decision Tree algorithms (e.g. Classification and Regression Tree)
Random Forest (ensemble)
Unsupervised Learning
We only know the input (unlabeled data)
Clustering
Association
K-means clustering
DBSCAN
Hierarchical clustering
PCA
Autoencoders
Deep belief networks
Generative adversarial networks
Reinforcement Learning
Control
Classification
Model-free
Value based (Q-learning)
Policy based
Deterministic or stochastic
On or off policy
Policy gradient, actor-critic
Model based
Learn the model
Model given
Deep Learning Networks
Convolutional Neural Networks
Recurrent Neural Networks
Long Short-Term Memory Networks
Autoencoders
Deep Boltzmann Machine
Deep Belief Networks
Bayesian Algorithms
Naive Bayes
Gaussian Naive Bayes
Bayesian Network
Bayesian Belief Network
Bayesian optimization
Regularization, dimensionality reduction, ensembling, evolutionary algorithms, computer vision, recommender systems, …

Supervised learning vs unsupervised learning

Supervised learning

Labeled data: we know x and y

*Learns the **mapping** between input and output*

*Focuses on **predicting** output for new data*

- **Classification**
- **Regression**
- **Image tasks:** image classification, object detection, semantic segmentation.
- **Sequence tasks**: text classification, sentiment analysis, text translation, time-series forecasting, sequence generation.

Unsupervised learning

Unlabeled data: we know x, we don't know y

*Models the **underlying structure** or distribution in the data to learn more about it*

*Focuses on **structure**, **distribution**, and **relationships** within the data*

- **Clustering**: dataset divided into groups (similarity).
- **Dimensionality reduction**: selection of dimensions that best explain the variabilty in the data.
- **Association rule learning**: find relationships between variables in big databases.
- **Density estimation**: estimate the distribution of the data (anomaly detection, data generation).

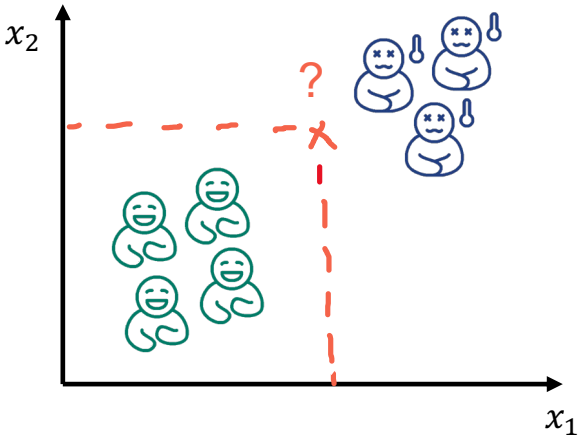
Supervised learning vs unsupervised learning

Supervised learning

Labeled data: we know x and y

*Learns the **mapping** between input and output*

*Focuses on **predicting** output for new data*

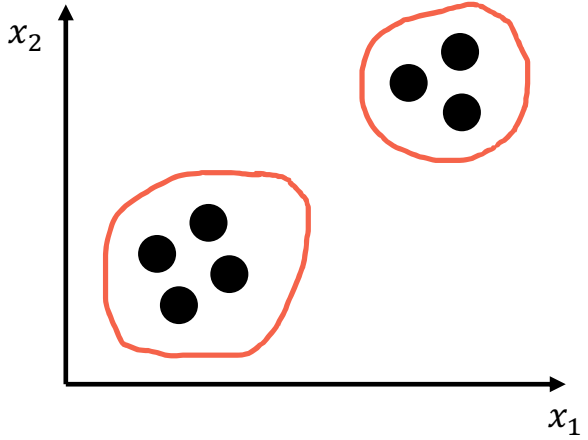


Unsupervised learning

Unlabeled data: we know x, we don't know y

*Models the **underlying structure** or distribution in the data to learn more about it*

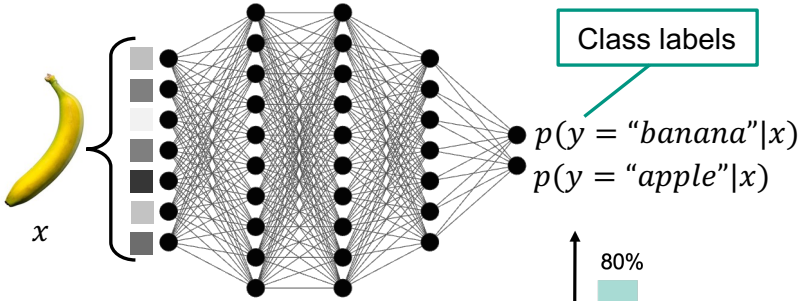
*Focuses on **structure**, **distribution**, and **relationships** within the data*



Classification vs regression in supervised learning

Classification task

y is discrete (can take certain values only)



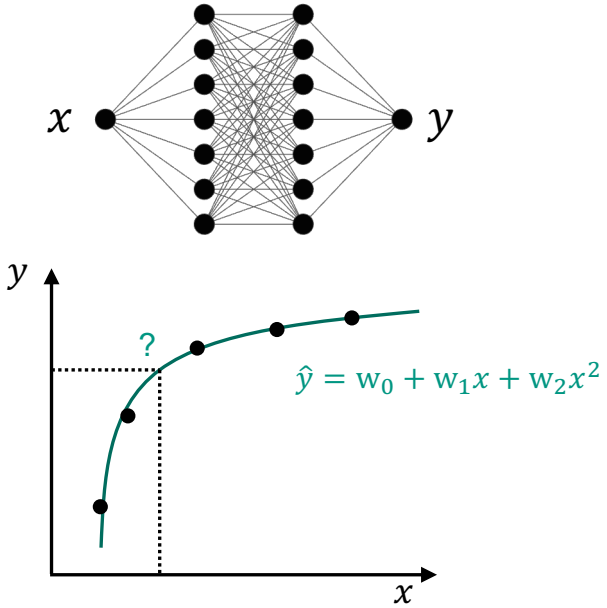
Likelihood function → categorical

$$L(w|x,y) = p_x(y)$$

Probability of a label y given the inputs x

Regression task

y is continuous (can take any value)



Examples of continuous targets

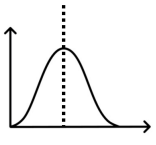
Product demand forecasting

Prices (t)
Seasonality
Promotions
Competitors
Economy
Sales data
Ads

μ
σ^2

Model's best guess for average demand

Model's uncertainty in the demand prediction



Likelihood function → gaussian

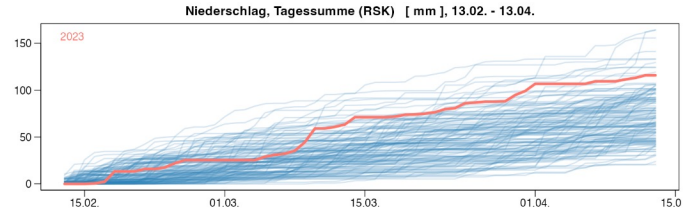
- Inventory management
- Risk assessment

$$L(w|x,y) = \text{normal}(\mu, \sigma^2)$$

Weather prediction

Temperature
Humidity
Air pressure
Wind speed
Precipitation
Solar radiation
Dew point
Cloud cover
Date

Rainfall
Snowfall
Air quality
Storm events
Temperature
Wind



https://github.com/brry/rdwd

Performance metrics

Classification task

y is discrete

- **Accuracy**: proportion of correctly classified instances out of all instances.
- **Precision**: proportion of true positive results divided by the number of all positive results (including false positives).
- **Recall (Sensitivity)**: proportion of true positive results divided by the number of positives that should have been identified.
- **F1 Score**
- **Confusion Matrix**
- **ROC Curve and AUC**

Regression task

y is continuous

- **Mean Absolute Error (MAE)**: average of the absolute differences between the predicted values and the actual values.

$$MAE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n}$$

- **Mean Squared Error (MSE)** or **Root Mean Squared Error (RMSE)**: average of the squared differences between the predicted values and the actual values.

$$MSE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|^2}{n} \quad RMSE = \sqrt{\frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|^2}{n}}$$

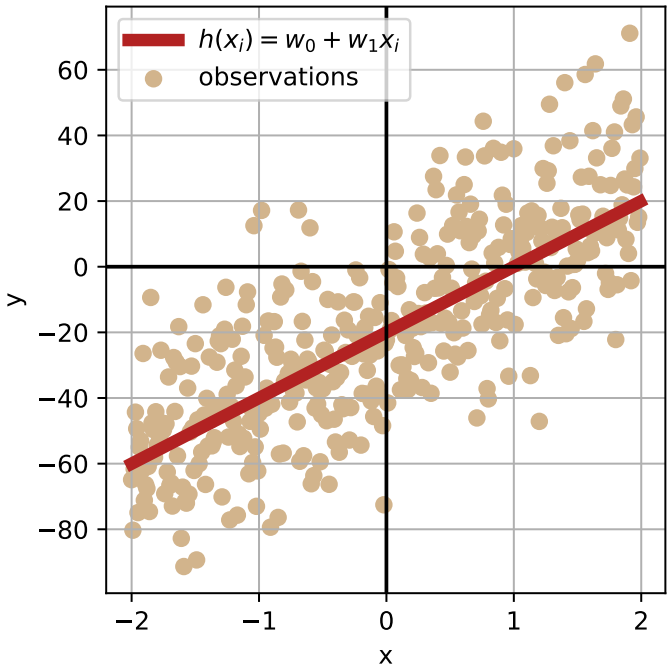
- **R-squared**: A statistical measure of how close the data are to the fitted regression line.

Let’s review regression

Univariate linear regression (one prediction)

Simple (one feature)

We want to fit linearly a set of points $(x_i,\ y_i)$



Hypothesis function

$$h: x \rightarrow \hat{y}$$

Space of input variables = **feature**

independent variable

Space of output variables = **estimated value** = **prediction**

dependent variable

$$h_w(x_i) = w_0 + w_1 x_i$$

- regression coefficients
- parameters of the model
- estimated from data

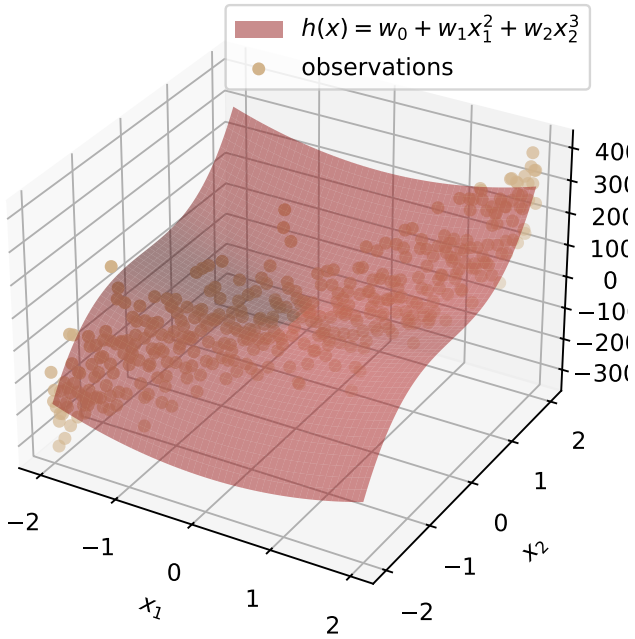
weights

$i = 1, \ldots, n$ data points

Univariate regression (one prediction)

Multiple (several features)

We want to fit a set of points $(x_i,\ y_i)$



Hypothesis function

$$h_w(x_1, x_2) = w_0 + w_1x_1^2 + w_2x_2^2$$

more generally

bias

$$h_w(x) = \underbrace{w_0}_{= b} + \sum_{i,\,k=1}^{n,\,p} w_k\phi_k(x_i)$$

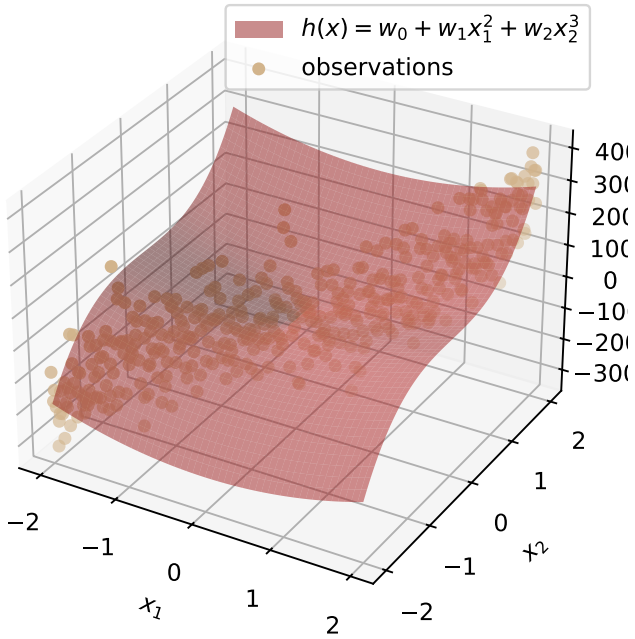
$i = 1, \dots, \boldsymbol{n}$	data points
$k = 1, \dots, \boldsymbol{p}$	features
$x_{0i} = 1$	pseudo-variable
$\phi_k(x_i) = x_{ki}$	basis function

Univariate regression (one prediction)

$i = 1, \dots, n$ data points
$k = 1, \dots, p$ features

Multiple (several features)

We want to fit a set of points $(x_i,\ y_i)$



Hypothesis function

$$h_w(x_1, x_2) = w_0 + w_1 x_1^2 + w_2 x_2^2$$

more generally

bias

$$h_w(x) = w_0 + \sum_{i,\,k=1}^{n,\,p} w_k \phi_k(x_i)$$

basis function

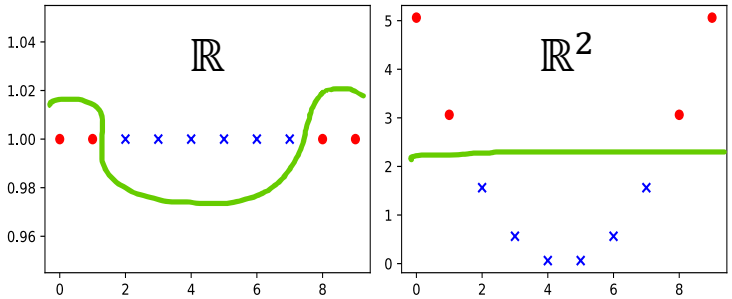
Allows to model nonlinearity in the data while keeping linearity in the parameters w

$\phi(x) = (x^0, x^1, x^2, \cdots, x^p)$
Polynomial basis function

Brief interlude

Basis functions $\phi(x)$

- Used for ***feature mapping*** $\phi : \mathbb{R} \rightarrow \mathbb{R}^L$
- They transform the input in a nonlinear way to a new space to capture more information about the data
  (e.g. make it linearly separable, easier to model):



They can be combined to fit complex nonlinear functions using ***linear regression***

Kernel functions $k(x_i, x')$

- Also used for feature mapping to higher dimensional spaces, where the number of basis functions would be too high and computationally expensive.
- Prediction for unseen data x' is done by measuring the similarity between x' and the training data x_i
  - *Similarity = inner product* $\begin{cases} x, x' \in \mathcal{X} \\ k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \end{cases}$

Example: prediction for x' in a binary classifier

$$\hat{y} = \text{sgn} \sum_{i=1}^{n} w_i y_i \boldsymbol{k(x_i, x')}$$

Often:

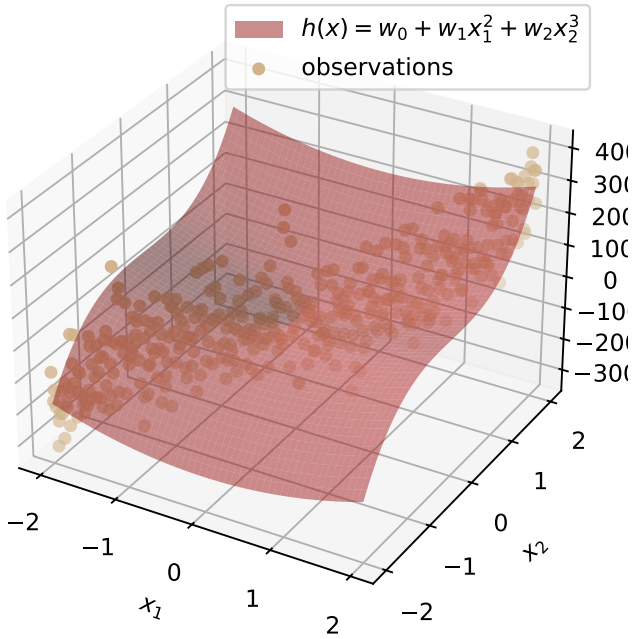
$$k(x, x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x') \quad \phi : \mathcal{X} \rightarrow \mathcal{V}$$

Univariate regression (one prediction)

$i = 1, \dots, n$ data points
$k = 1, \dots, p$ features

Multiple (several features)

We want to fit a set of points $(x_i,\ y_i)$



Matrix notation

$$h_w(x) = W^T X = \boldsymbol{X^T W}$$

=1

$$\boldsymbol{X^T} = \begin{bmatrix} x_{01} & x_{11} & \cdots & x_{p1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{0n} & x_{1n} & \cdots & x_{pn} \end{bmatrix}$$

$[n \times (p+1)]$

bias included

$$\boldsymbol{W} = \begin{bmatrix} w_0 \\ \vdots \\ w_p \end{bmatrix}$$

$[(p+1) \times 1]$

How to iteratively find the correct fit?

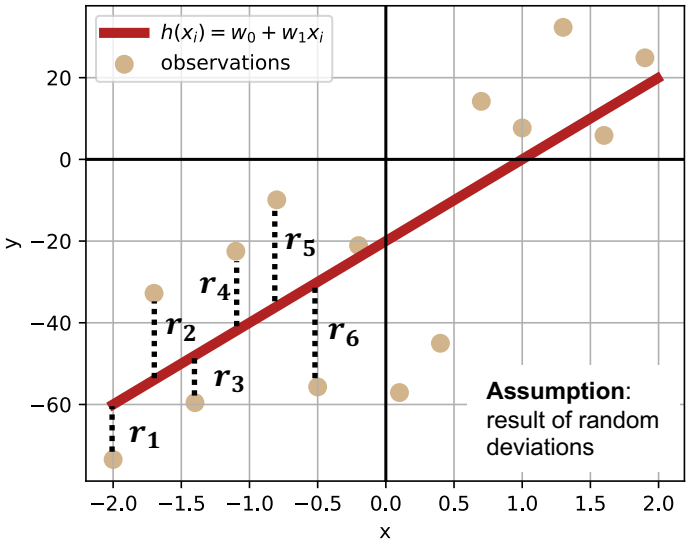
By minimizing the loss function

Goal: choose w_k such that $\boldsymbol{h_w(x_i)}$ is as close to y_i as possible

$$\boldsymbol{h_w(x_i)} - y_i = r_i$$

residual

error



Loss function $J_W(\hat{y}, y)$

Also called cost function or error function

→ Number representing how well your algorithm models your dataset

Quantifies the error between the predicted values $\hat{y}$ by a model and the target values y of the data it is trying to fit or predict given your current weights w

*The goal of training a model is to find the parameters w that **minimize the loss function***

The loss is monitored during the training of NNs

MAE (L1 loss)

MSE (L2 loss)

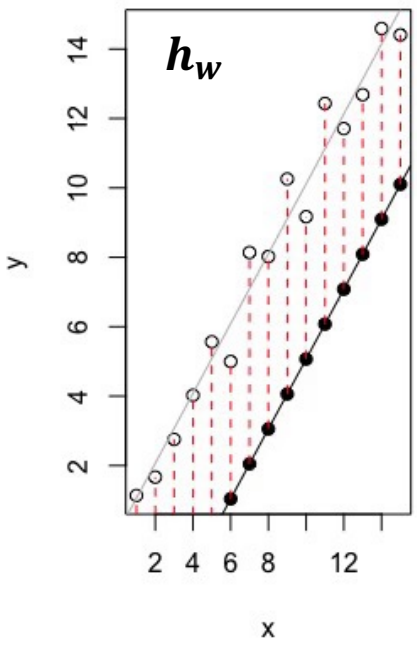
Negative log-likelihood function

Cross-entropy loss

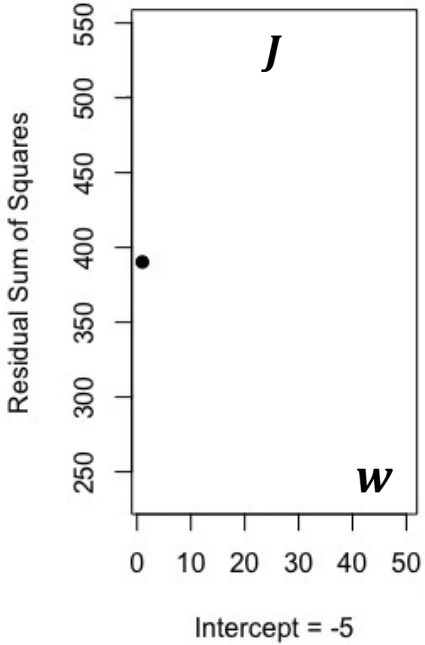
Example: least squares fitting

Common parameter estimation in regression analysis

Scan of hypothesis



Calculation of loss



Animation from: https://yihui.org/animation/example/least-squares/

Goal: find the weights that minimize $\boldsymbol{J}$

$$\sum_{i=1}^{n} \boldsymbol{r_i^2} = \sum_{i=1}^{n} (h_w(x_i) - y_i)^2 = |\boldsymbol{X}w - \boldsymbol{y}|^2 = \boldsymbol{J}(\boldsymbol{w})$$

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, J(\boldsymbol{w})$$

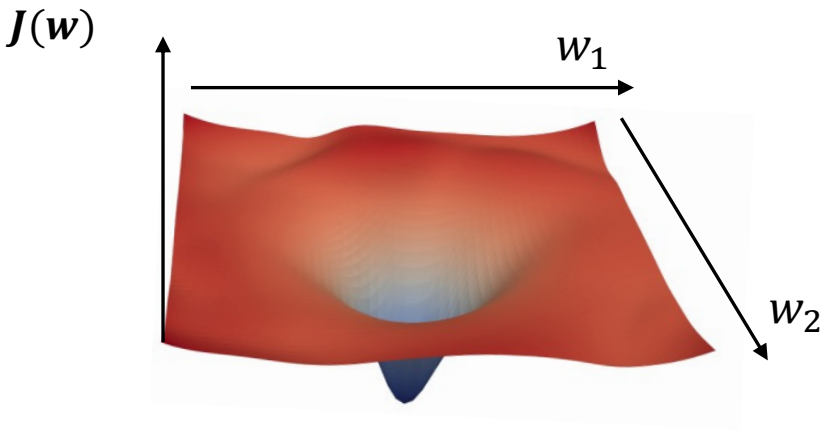
It depends on the weights (parameters) of your model

⚠ **Use least squares when:**

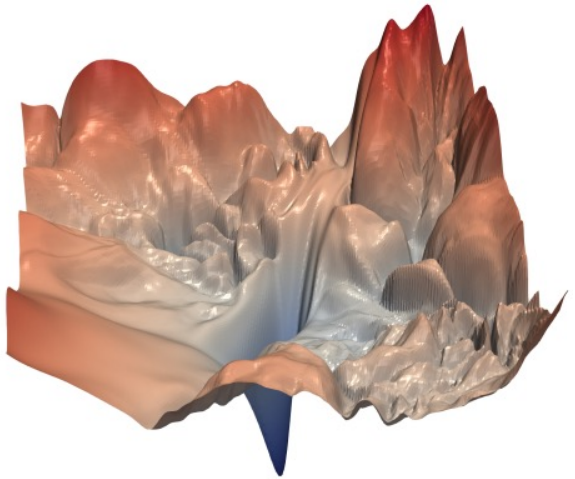
- System is overdetermined
  - Points > features $(n > p + 1)$
- Uncertainties in the data are "controlled" — Why?
  - ➢ Otherwise: maximum likelihood estimation, ridge regression, lasso regression, least absolute deviation, bayesian linear regression, etc.

The “loss landscape”

This is a simple loss landscape for two weights:



Nice and simply convex
Easy to minimize, convergence guaranteed



This is more how it can be with deep NNs
(with no regularization)

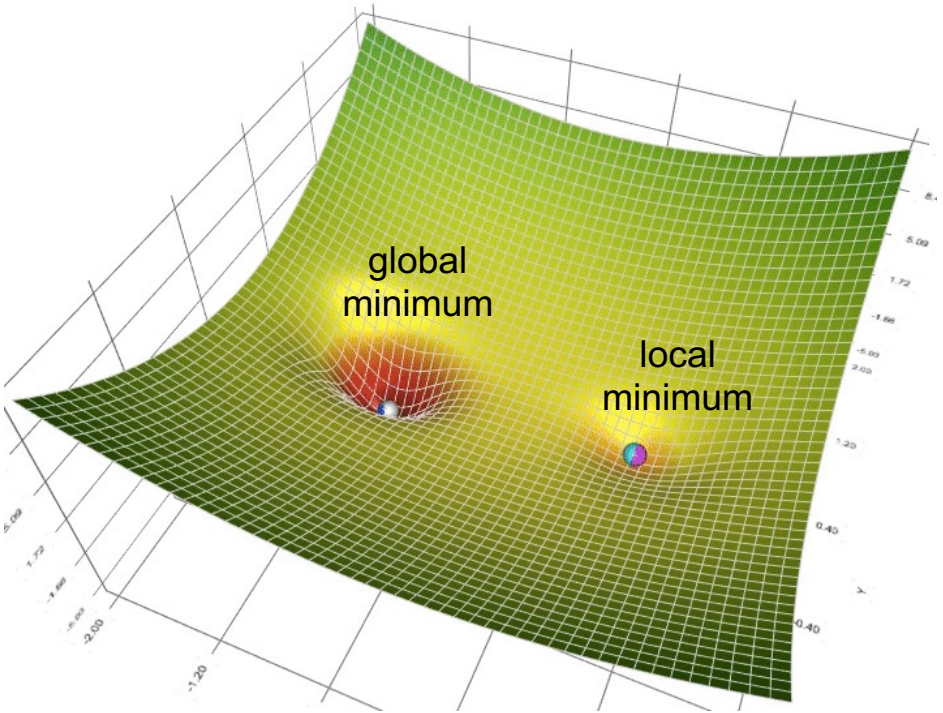
Remarks:

- We never see it like this, we only get one loss function value per algorithm iteration
- This is usually much higher dimensional

https://arxiv.org/pdf/1712.09913.pdf

How do we find our way in the loss landscape?

With optimizers!



Animation from Lili Jiang

Gradient based methods (first order):

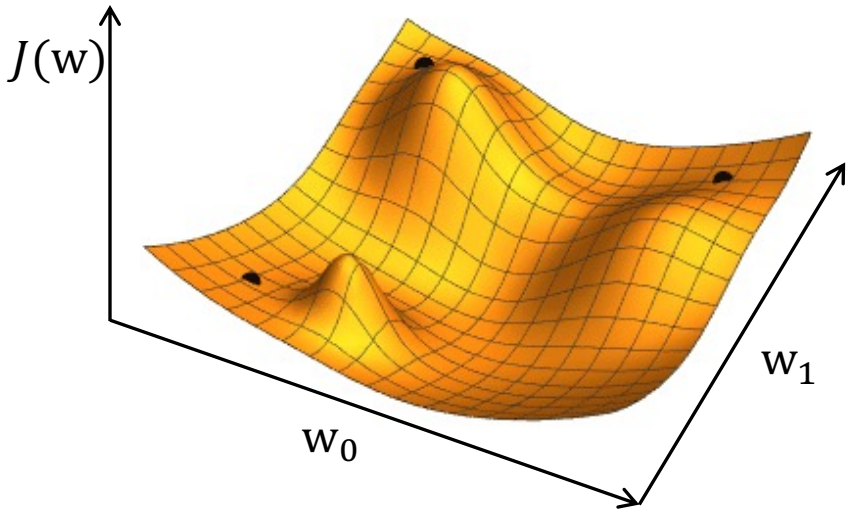
- Gradient descent
- momentum
- AdaGrad
- RMSProp
- Adam

dynamic adjustment of algorithm parameters

More tomorrow!

Gradient descent

Algorithm to iteratively solve $\underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$



Animation from Wikimedia commons

Update rule:

step size learning rate

first order

$$\boldsymbol{w} := \boldsymbol{w} - \alpha \boldsymbol{\nabla} J(\mathrm{w}_k)$$

take repeated steps in the opposite direction of the gradient

$$J(\mathrm{w}_k) = \frac{1}{2n}\sum_{i=1}^{n}(h_{\mathrm{w}}(x_i) - y_i)^2$$

Loss function

$$h_{\mathrm{w}}(x) = \mathrm{w}_0 + \mathrm{w}_1 x$$

Hypothesis

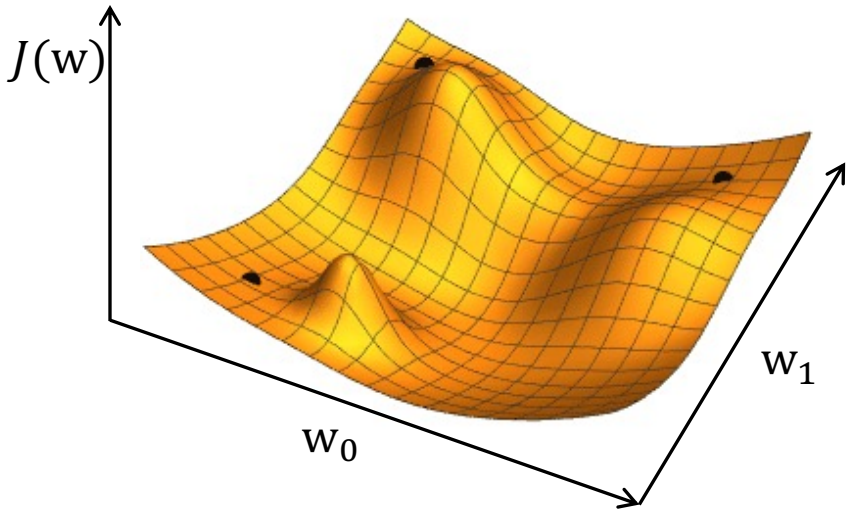
$$\vec{\nabla} = \left(\frac{\partial}{\partial w_1}, \ldots, \frac{\partial}{\partial w_k}\right)$$

$$\frac{\partial J(\mathrm{w}_0, \mathrm{w}_1)}{\partial \mathrm{w}_0} = \frac{1}{n}\sum_{i=1}^{n}(\mathrm{w}_0 + \mathrm{w}_1 x_i - y_i)$$

$$\frac{\partial J(\mathrm{w}_0, \mathrm{w}_1)}{\partial \mathrm{w}_1} = \frac{1}{n}\sum_{i=1}^{n} x_i(\mathrm{w}_0 + \mathrm{w}_1 x_i - y_i)$$

Gradient descent

Algorithm to iteratively solve $\underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$



Animation from Wikimedia commons

Update rule:

step size learning rate

first order

$$\boldsymbol{w} := \boldsymbol{w} - \alpha \boldsymbol{\nabla} J(\mathrm{w}_k)$$

take repeated steps in the opposite direction of the gradient

$$\frac{\partial J(\mathrm{w}_0, \mathrm{w}_1)}{\partial \mathrm{w}_0} = \frac{1}{n}\sum_{i=1}^{n}(\mathrm{w}_0 + \mathrm{w}_1 x_i - y_i)$$

$$\frac{\partial J(\mathrm{w}_0, \mathrm{w}_1)}{\partial \mathrm{w}_1} = \frac{1}{n}\sum_{i=1}^{n} x_i(\mathrm{w}_0 + \mathrm{w}_1 x_i - y_i)$$

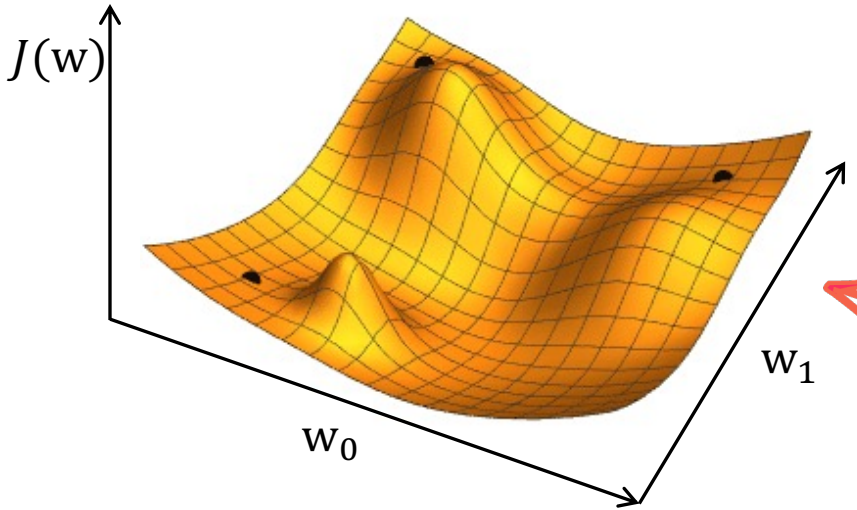
$$\mathrm{w}_0 := \mathrm{w}_0 - \alpha \frac{\partial J(\mathrm{w}_0, \mathrm{w}_1)}{\partial \mathrm{w}_0}$$

$$\mathrm{w}_1 := \mathrm{w}_1 - \alpha \frac{\partial J(\mathrm{w}_0, \mathrm{w}_1)}{\partial \mathrm{w}_1}$$

updated simultaneously

Gradient descent

Algorithm to iteratively solve $\underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$



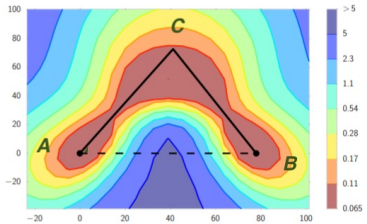
Animation from Wikimedia commons

General remarks

- Works in any number of dimensions.
- Depending on the initial $(\mathrm{w}_0, \mathrm{w}_1)$ optimization can end up at different points.
- It proceeds very slowly near saddle points but can eventually escape them if weights were randomly initialized.
- In highly nonconvex loss landscapes, gradient descent often finds near-optimal solutions.

Recent research shows that simple, low-cost paths connect different optima

→ Low loss connected manifold



Garipov et al. 2018
Draxler et al. 2019

Gradient descent

Algorithm to iteratively solve $\underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$

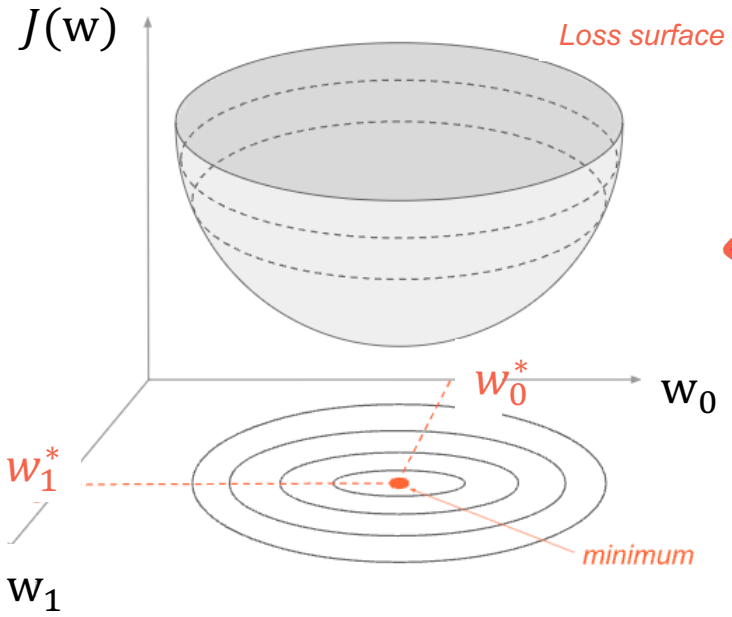


Image from https://allmodelsarewrong.github.io/gradient.html

Matrix notation

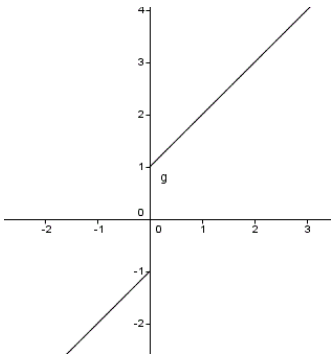
$$J(W) = \frac{1}{2}(XW - Y)^T(XW - Y)$$

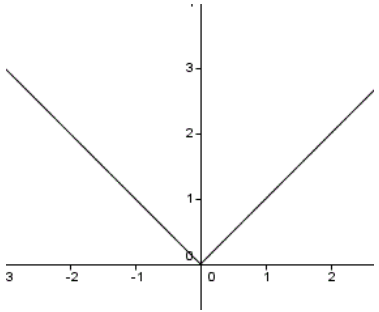
$$= \frac{1}{2}(\underbrace{W^T X^T XW}_{\text{cuadratric}} - \underbrace{2Y^T XW}_{\text{linear}} + \underbrace{Y^T Y}_{\text{constant}})$$

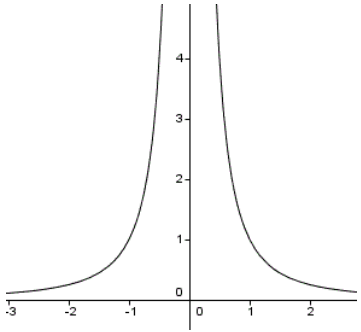
Analytically: $W = (X^T X)^{-1} X^T Y$

Iteratively: $\frac{\partial J}{\partial W} = X^T(X^T W - Y)$

Gradient descent: some limitations







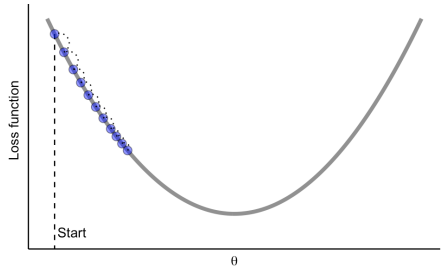
- Sensitivity to learning rate (next slide).
- Slow convergence in areas with small gradients (plateaus).
- Very dependent on weight initialization.
- Computationally intensive for large datasets.
- Only works when the function is continuously differentiable (gradient needs to exist).

Gradient descent: remarks

Step size / learning rate $\boldsymbol{w} \coloneqq \boldsymbol{w} - \alpha \nabla J(\mathrm{w}_k)$

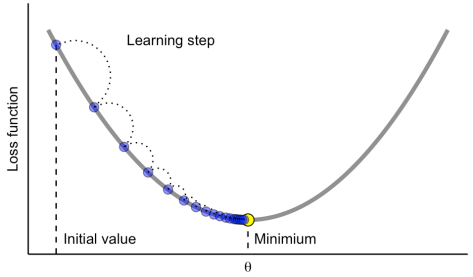
Too small

Converges slowly



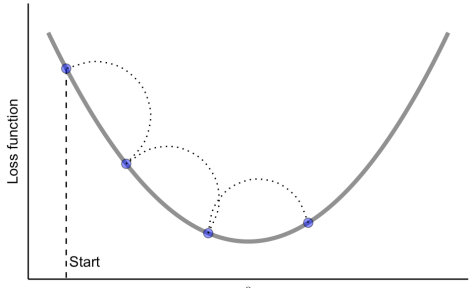
Just right

Converges quickly



Too large

Diverges



Images from https://bradleyboehmke.github.io

Gradient descent: remarks

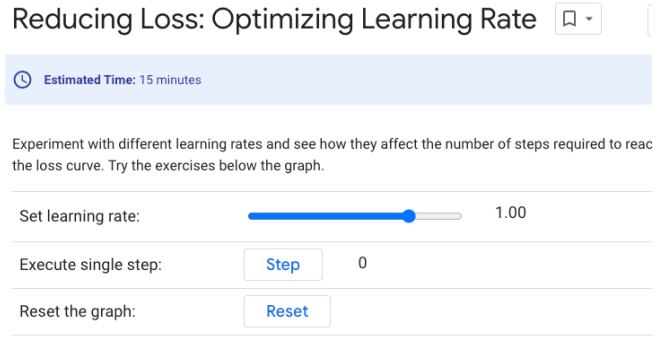
Step size / learning rate $\boldsymbol{w} \coloneqq \boldsymbol{w} - \alpha \nabla J(\mathrm{w}_k)$

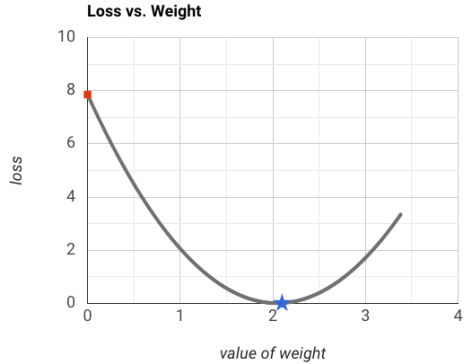
Let's try it out!

https://developers.google.com/machine-learning/crash-course/fitter/graph

Try the following "start learning rate":

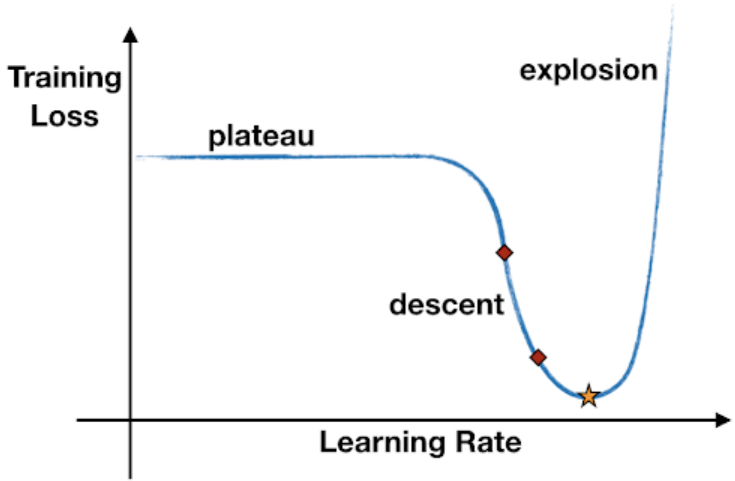
- 0.02
- 0.1
- 0.4
- 1





Gradient descent: remarks

Step size / learning rate $\boldsymbol{w} \coloneqq \boldsymbol{w} - \alpha \nabla J(\mathrm{w}_k)$



Modern algorithms have adaptive learning rates

- The optimizer is working correctly if the loss decreases after every iteration.
- The recommended minimum learning rate is the value where the loss decreases the fastest
- Usual to **declare convergence tests.** (e.g., declare convergence when $J(\theta)$ decreases by less than 10^{-3} in one iteration)

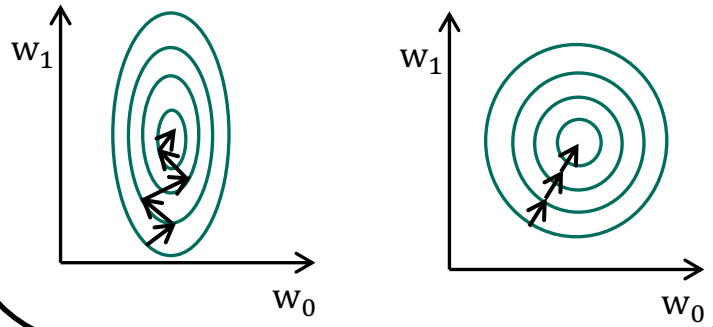
Images from https://blog.dataiku.com/

Gradient descent: remarks

Feature scaling

- Optimizers can converge faster if the features are on a similar scale
- Becomes very important in polynomial regression:

$$w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



Normalization

$$-1 \leq x_i \leq 1 \quad \text{or} \quad 0 \leq x_i \leq 1$$

To rescale between [a, b]:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

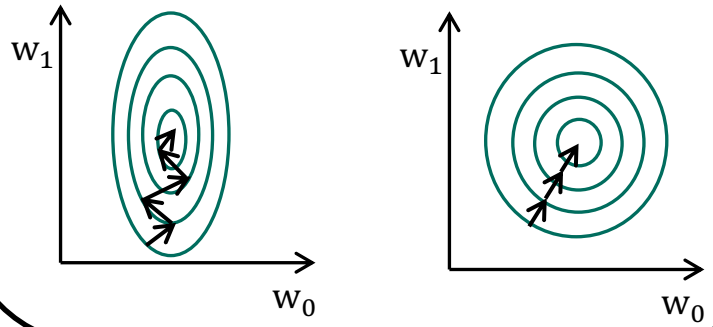
- Geometrically: will scale the n-dimensional data into an n-dimensional hypercube.
- Useful when there are no outliers and distribution is uniform.

Gradient descent: remarks

Feature scaling

- Optimizers can converge faster if the features are on a similar scale
- Becomes very important in polynomial regression:

$$w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



Standardization

$$\mu = 0, \ \sigma^2 = 1$$

$$x' = \frac{x - \mu}{\sigma}$$

- Geometrically: translates the mean of the data to the origin and scales its spread.
- Useful when the data follows a Gaussian distribution.
- Deals well with outliers.

But when are we going to learn about neural networks??

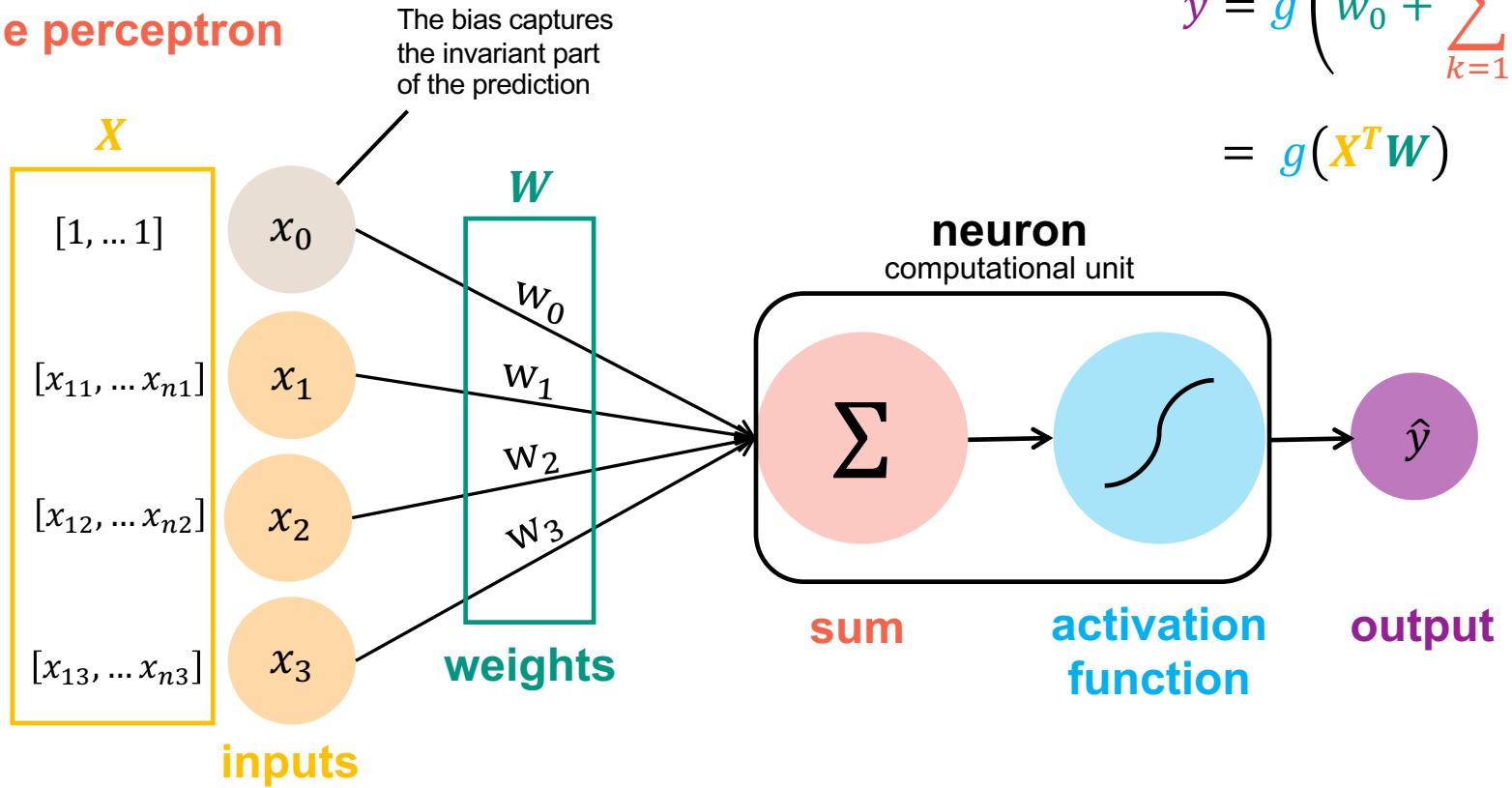

NNs
Linear algebra, multivariate calculus, statistics
NNs

Neural networks

The perceptron

$$\hat{y} = g\left(w_0 + \sum_{k=1}^{p} x_i\, w_i\right)$$

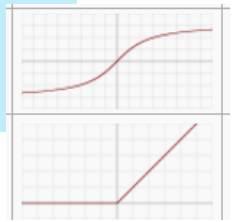
$$= g(X^T W)$$

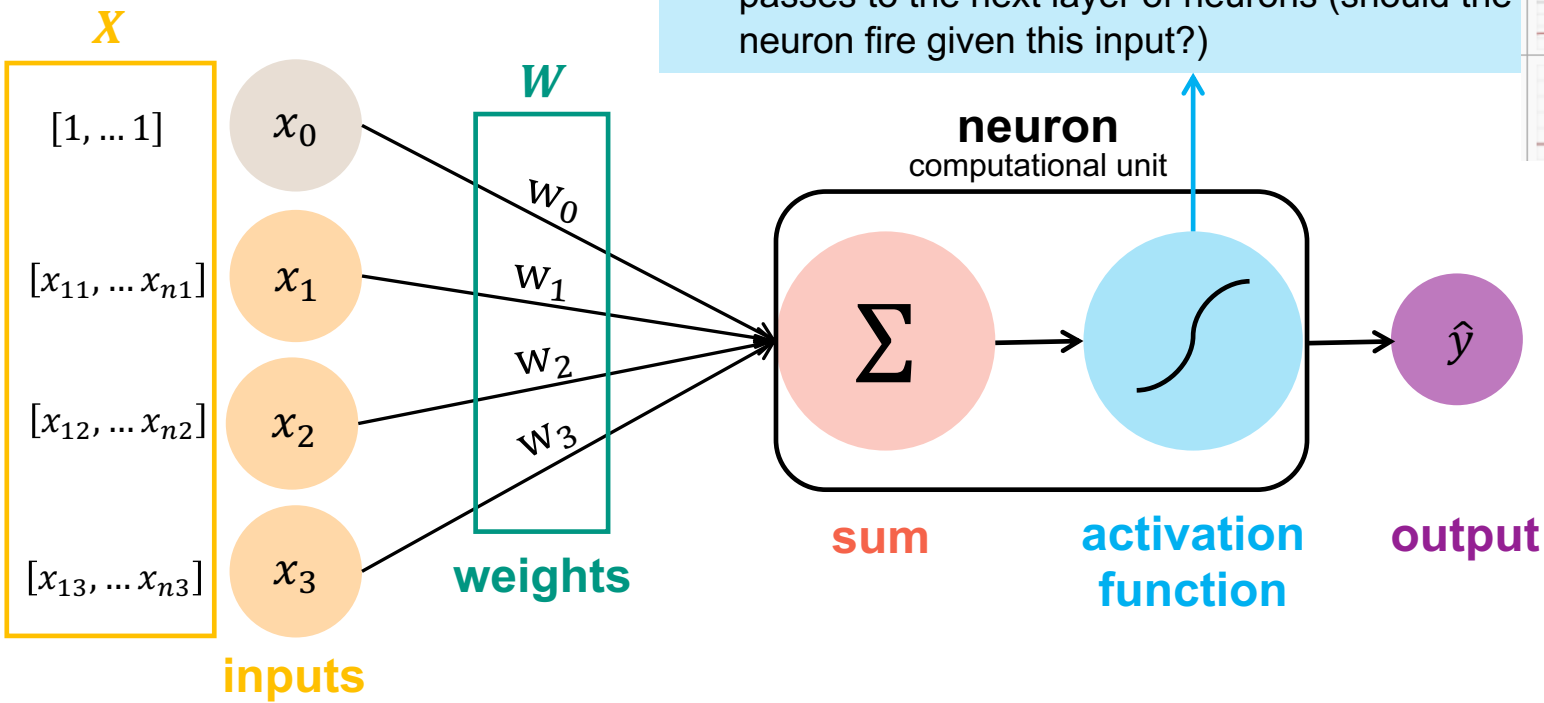


Neural networks

The perceptron

- What makes NNs different from linear regression.
- Defines the output of the neuron given an input.
- "**Mathematical gate**" that filters which data passes to the next layer of neurons (should the neuron fire given this input?)

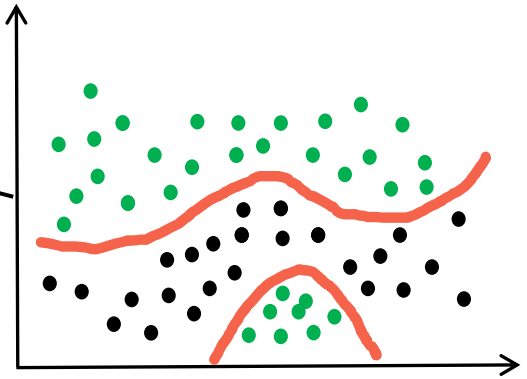


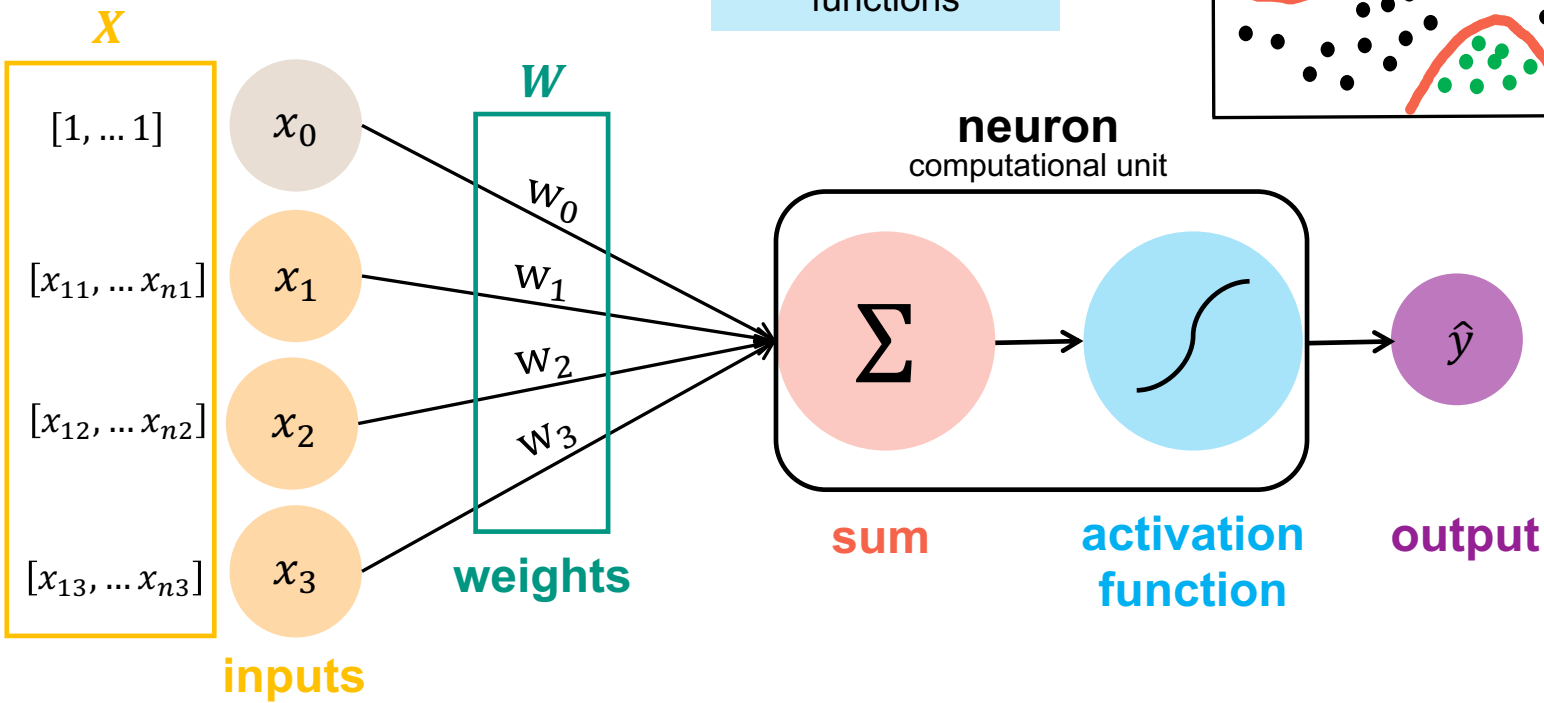


Neural networks

The perceptron

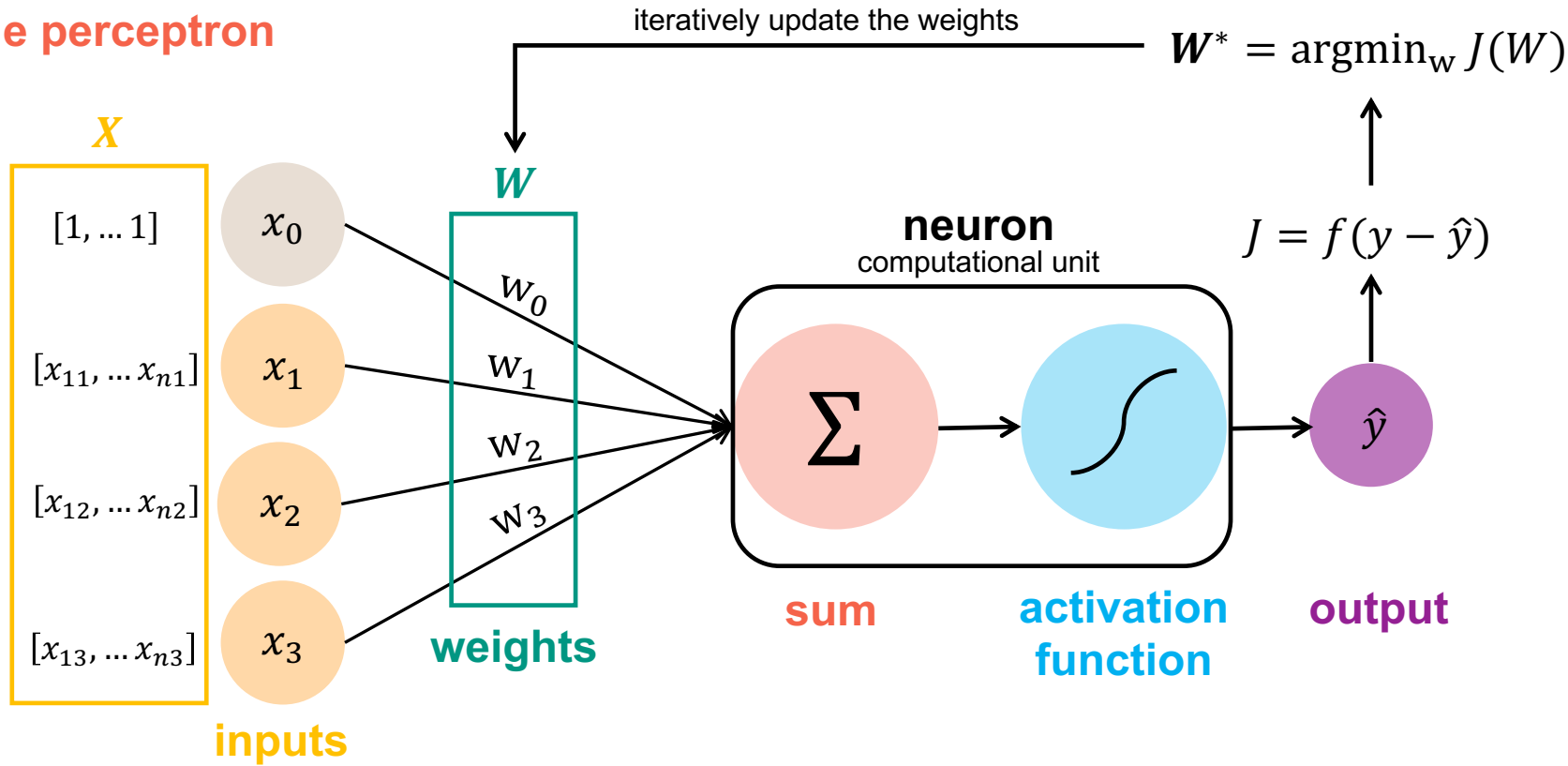
introduces nonlinearities to approximate arbitrarily complex functions





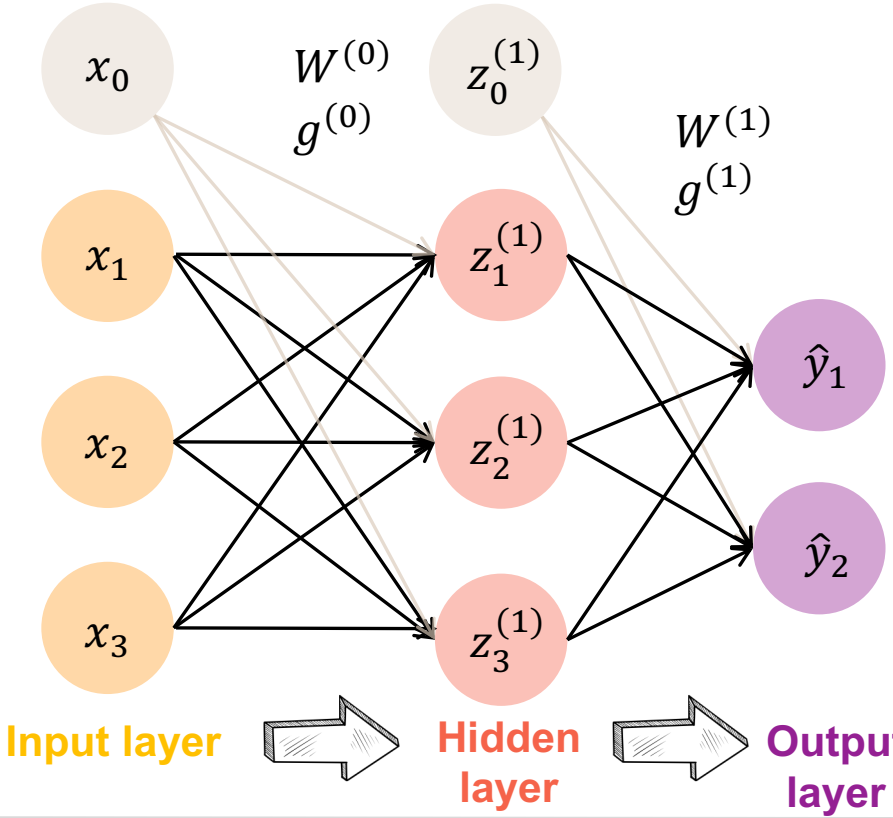
Neural networks

The perceptron



Forward propagation

Single layer neural network



New notation:

- **Superscript** = layer number
- **Subscript** = neuron number
- z = activation vector or unit
- g = activation function

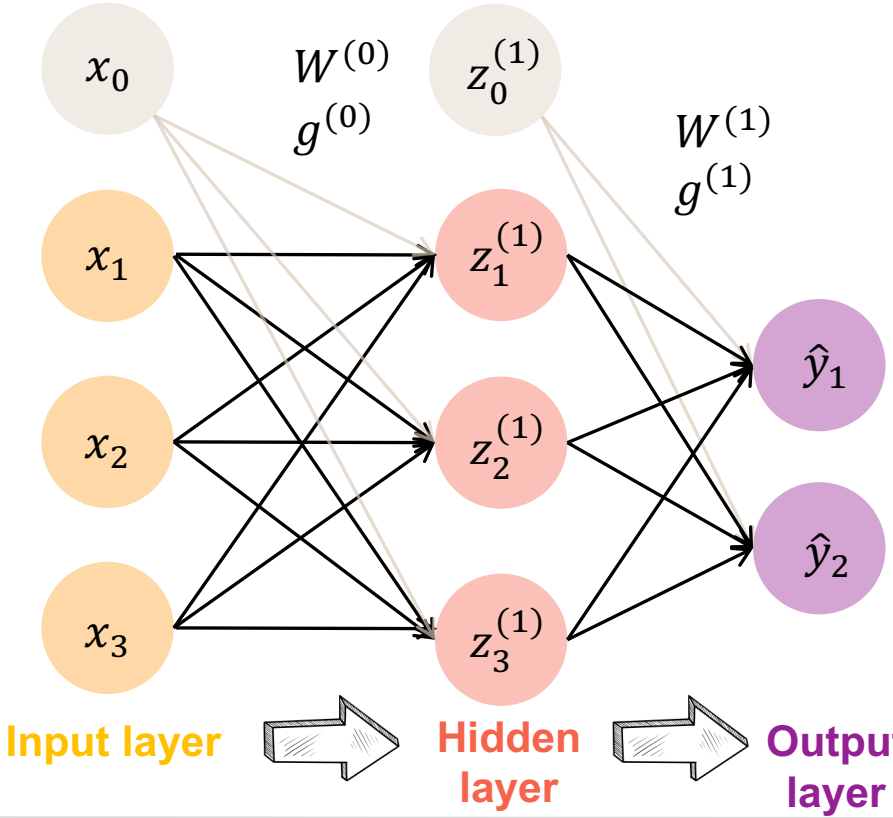
$$\hat{y} = g^{(1)}(\boldsymbol{W}^{(1)}\overbrace{g^{(0)}(\boldsymbol{W}^{(0)}\boldsymbol{X})}^{\boldsymbol{z}^{(1)}})$$

multilayer network evaluates compositions of functions computed at individual neurons

Forward propagation

Single layer neural network

$$\hat{y} = g^{(1)}(\boldsymbol{W}^{(1)} \overbrace{g^{(0)}(\boldsymbol{W}^{(0)}\boldsymbol{X})}^{\boldsymbol{z}^{(1)}})$$



Step 1

$$\boldsymbol{W}^{(0)} = \begin{bmatrix} w_{10}^{(0)} & w_{11}^{(0)} & w_{12}^{(0)} & w_{13}^{(0)} \\ w_{20}^{(0)} & w_{21}^{(0)} & w_{22}^{(0)} & w_{23}^{(0)} \\ w_{30}^{(0)} & w_{31}^{(0)} & w_{32}^{(0)} & w_{33}^{(0)} \end{bmatrix}$$

neurons in hidden layer

neurons in input layer + bias

$$\boldsymbol{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

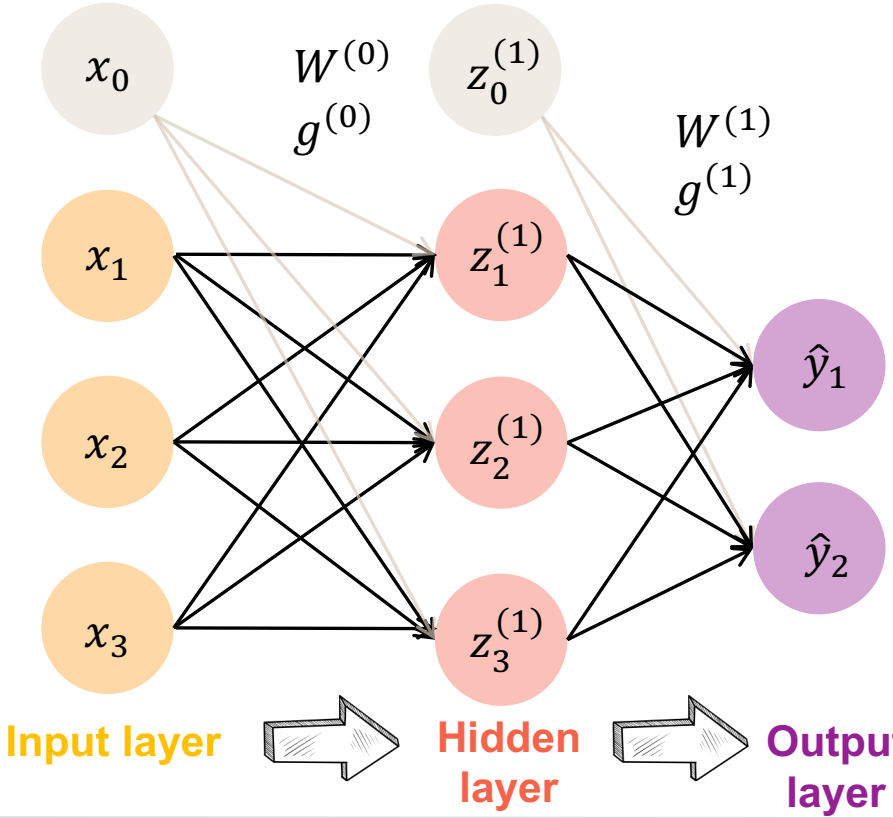
$x_1 = (x_{11}, \ldots, x_{1n})$

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} a_0^{(1)} \\ g^{(0)}(w_{10}^{(0)} x_0 + w_{11}^{(0)} x_1 + w_{12}^{(0)} x_2 + w_{13}^{(0)} x_3) \\ g^{(0)}(w_{20}^{(0)} x_0 + w_{21}^{(0)} x_1 + w_{22}^{(0)} x_2 + w_{23}^{(0)} x_3) \\ g^{(0)}(w_{30}^{(0)} x_0 + w_{31}^{(0)} x_1 + w_{32}^{(0)} x_2 + w_{33}^{(0)} x_3) \end{bmatrix}$$

Forward propagation

Single layer neural network

$$\hat{y} = g^{(1)}(\boldsymbol{W}^{(1)}\overbrace{g^{(0)}(\boldsymbol{W}^{(0)}\boldsymbol{X})}^{\boldsymbol{z}^{(1)}})$$



Step 2

$$w^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}$$

neurons in output layer

neurons in hidden layer + bias

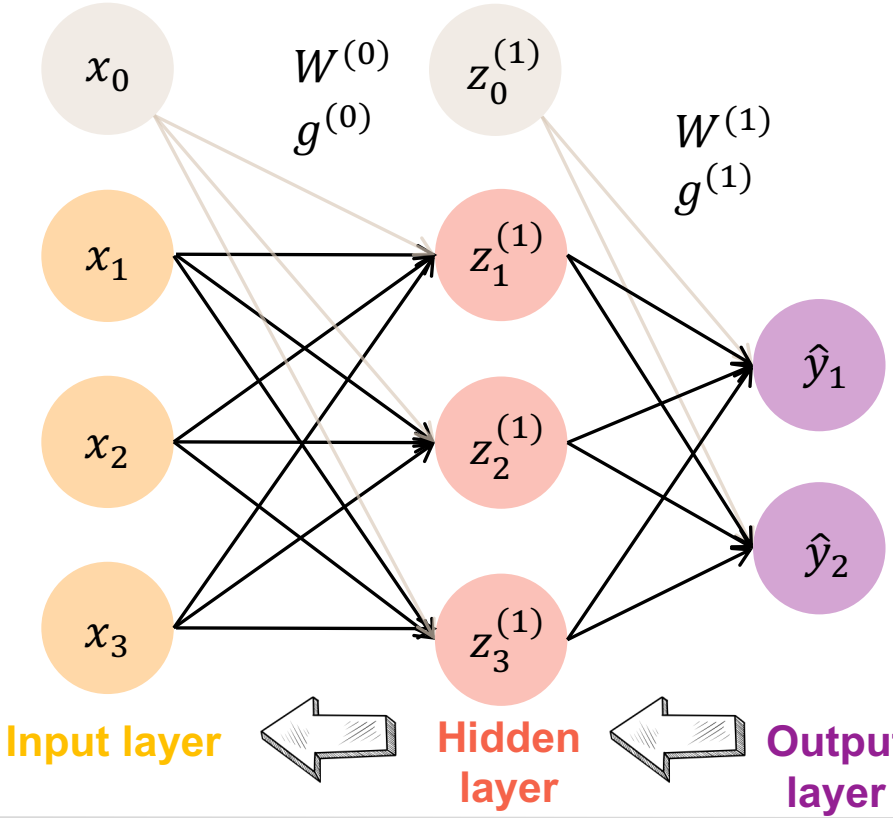
$$z^{(1)} = \begin{bmatrix} z_0^{(1)} \\ z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix}$$

$$z^{(2)} = \hat{y} = \begin{bmatrix} g^{(1)}(w_{10}^{(1)} z_0^{(1)} + w_{11}^{(1)} z_1^{(1)} + w_{12}^{(1)} z_2^{(1)} + w_{13}^{(1)} z_3^{(1)}) \\ g^{(1)}(w_{20}^{(1)} z_0^{(1)} + w_{21}^{(1)} z_1^{(1)} + w_{22}^{(1)} z_2^{(1)} + w_{23}^{(1)} z_3^{(1)}) \end{bmatrix}$$

✅ Forward propagation completed

Backpropagation

Single layer neural network



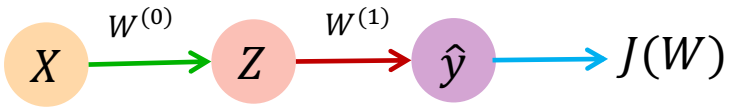
We want to find the network weights that achieve the lowest loss

$$\boldsymbol{W}^* = \text{argmin}_{\text{w}}\, J(\boldsymbol{W});\ \ \boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}\}$$

With gradient descent

$$W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$$

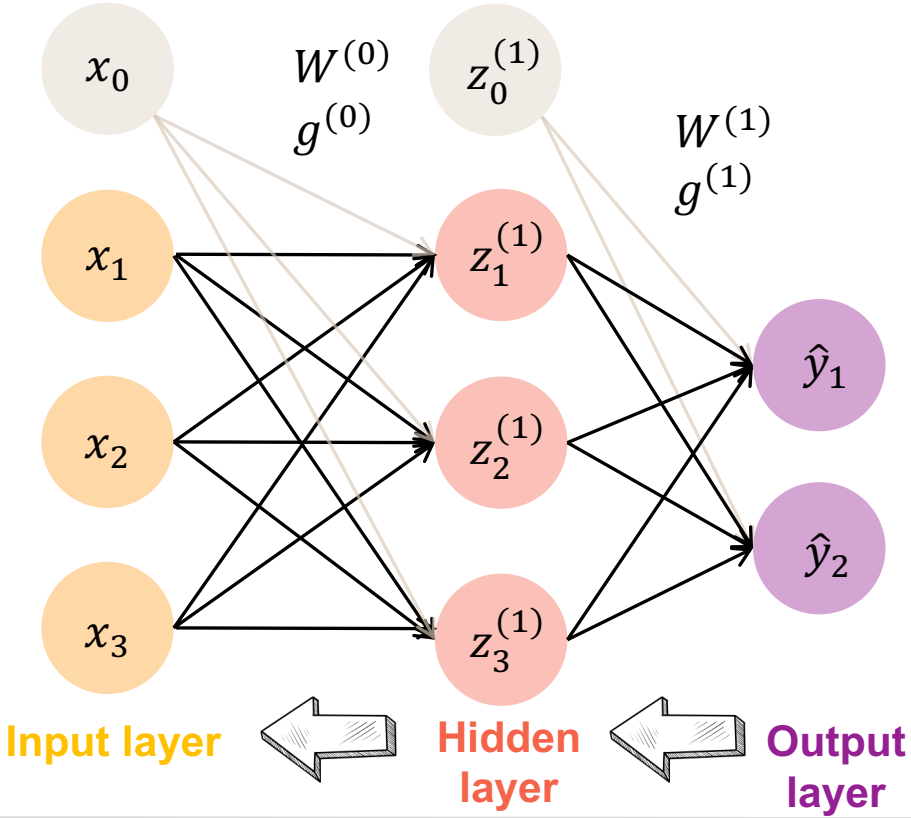
Calculating gradients with the chain rule



- We need to consider the sequential nature of the operations in a neural network (layer by layer) to calculate the new weights, as they depend on each other.
- We can do this by applying the chain rule to the gradients, since we have a composition of functions.

Backpropagation

Single layer neural network



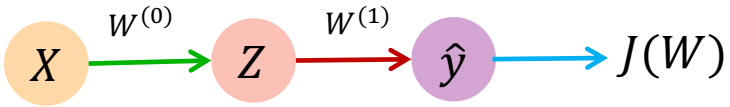
We want to find the network weights that achieve the lowest loss

$$\boldsymbol{W}^* = argmin_{\mathrm{w}} J(\boldsymbol{W});\ \boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}\}$$

With gradient descent

$$W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$$

Calculating gradients with the chain rule



$$\frac{\partial J(W)}{\partial W^{(1)}} = \frac{\partial J(W)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W^{(1)}}$$

$$J(W) = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(\hat{y}^2 + y^2 - 2\hat{y}y)$$

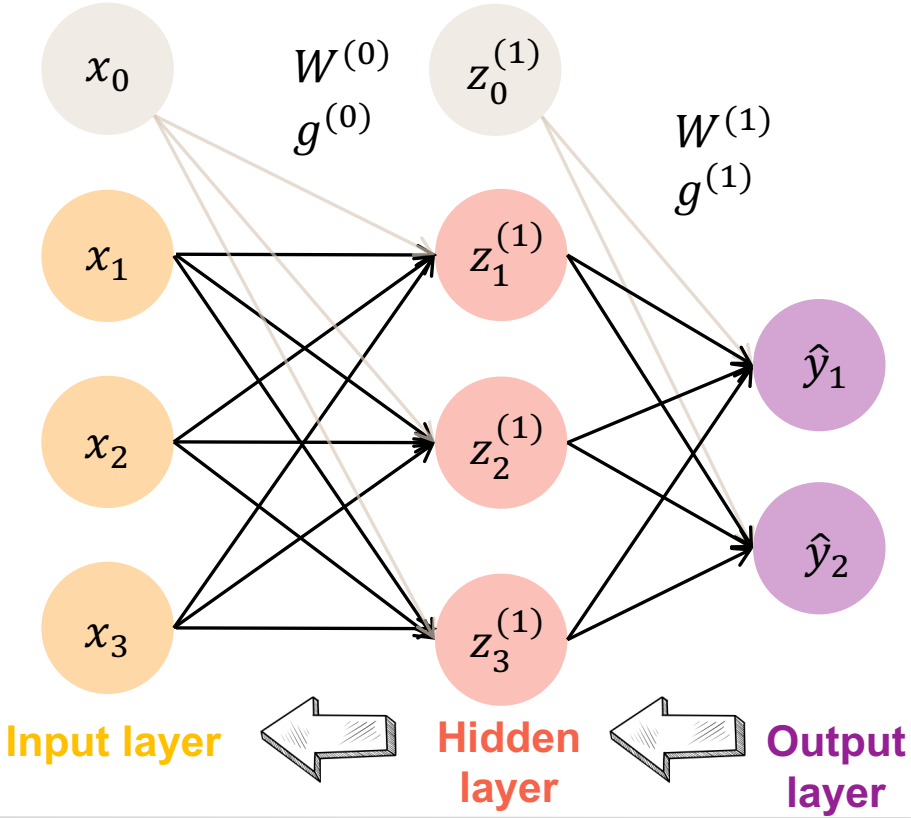
$$\hat{y} = g^{(1)}(W^{(1)} g^{(0)}(W^{(0)} X)) = g^{(1)}(W^{(1)} Z^{(1)})$$

$$\frac{\partial J(W)}{\partial \hat{y}} = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial W^{(1)}} = g'^{(1)}(W^{(1)} Z^{(1)}) Z^{(1)}$$

Backpropagation

Single layer neural network



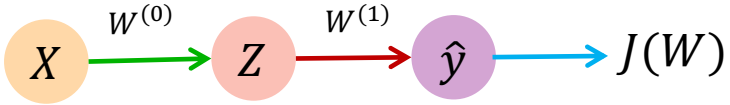
We want to find the network weights that achieve the lowest loss

$$\boldsymbol{W}^* = argmin_{\mathrm{w}} J(\boldsymbol{W});\ \boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}\}$$

With gradient descent

$$W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$$

Calculating gradients with the chain rule



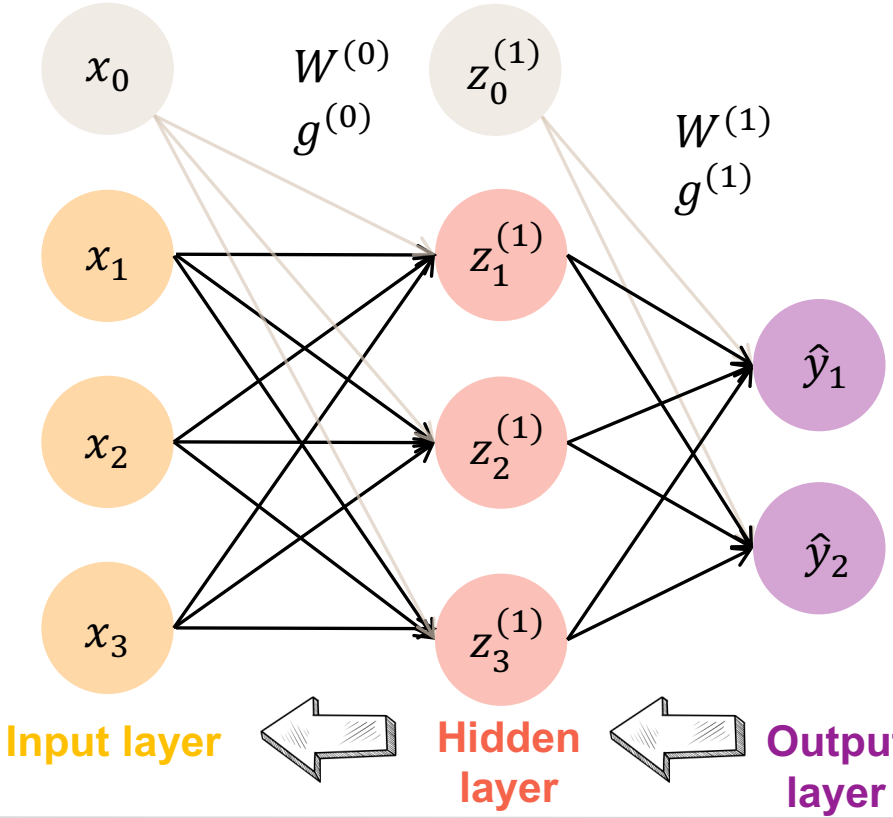
$$\frac{\partial J(W)}{\partial W^{(0)}} = \frac{\partial J(W)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z^{(1)}} \frac{\partial Z^{(1)}}{\partial W^{(0)}}$$

$$Z^{(1)} = g^{(0)}(W^{(0)}X)$$

$\hat{y} - y$ $\quad g'^{(1)}(W^{(1)}Z^{(1)})Z^{(1)}$ $\quad g'^{(0)}(W^{(0)}X)X$

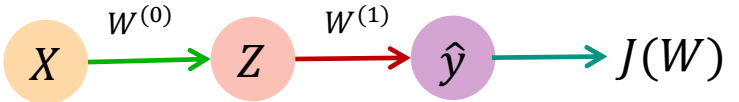
Backpropagation

Single layer neural network



We want to find the network weights that achieve the lowest loss

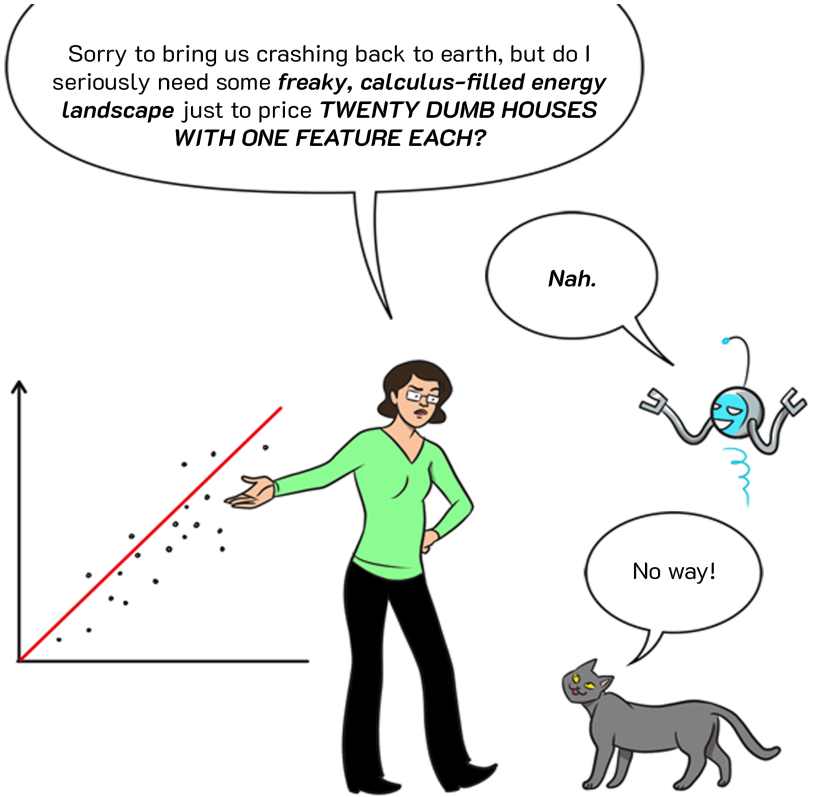
$$\boldsymbol{W}^* = argmin_{\mathrm{w}} J(\boldsymbol{W}); \;\; \boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}\}$$

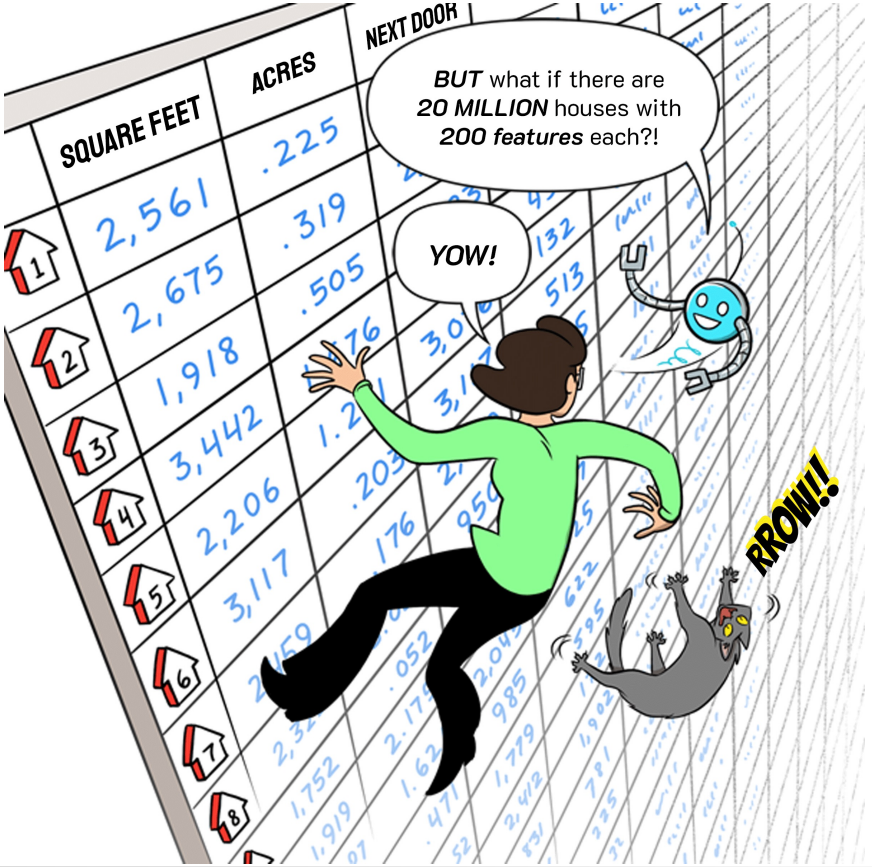


$$\frac{\partial J(W)}{\partial W^{(1)}} = \frac{\partial J(W)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W^{(1)}} \qquad \frac{\partial J(W)}{\partial W^{(0)}} = \frac{\partial J(W)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z^{(1)}} \frac{\partial Z^{(1)}}{\partial W^{(0)}}$$

- Efficient computation of gradients (by parts).
- **Automatic differentiation (autodiff)**:
  - Efficient $O(n)$ and numerically stable way of calculating gradients.
  - Decomposes the algorithm into primitive operations
  - Explicitly constructs a computation graph.
- **Autograd** = Python autodiff library, wraps numpy functions

NNs: computationally efficient for big data!





Comic source: https://cloud.google.com/products/ai/ml-comic-1

Thank you for your attention!

What questions do you have?

Resources:

https://ml-cheatsheet.readthedocs.io/

https://notesonai.com/

https://buildmedia.readthedocs.org/media/pdf/ml-cheatsheet/latest/ml-cheatsheet.pdf

http://introtodeeplearning.com/

https://www.offconvex.org/

All icons from this talk from TheNounProject