ÖAW ÖSTERREICHISCHE AKADEMIE DER WISSENSCHAFTEN

AI WinterSchool

January 20 - 24, 2025

Today’s Program

Part I: Introduction lecture

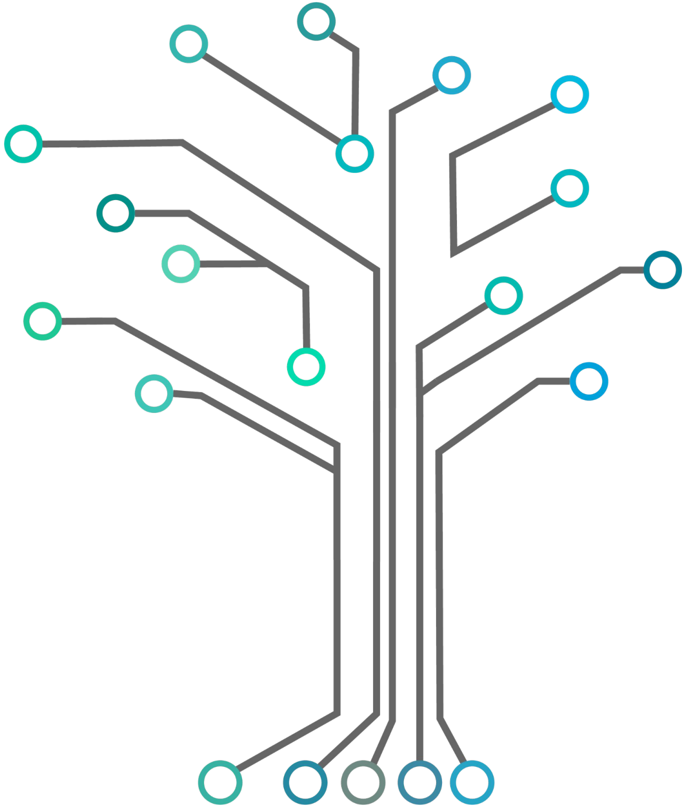
14:15 - 15:45

- Overview
- Theoretical Basics
- Data
- Training
- Evaluation
- Design and Techniques

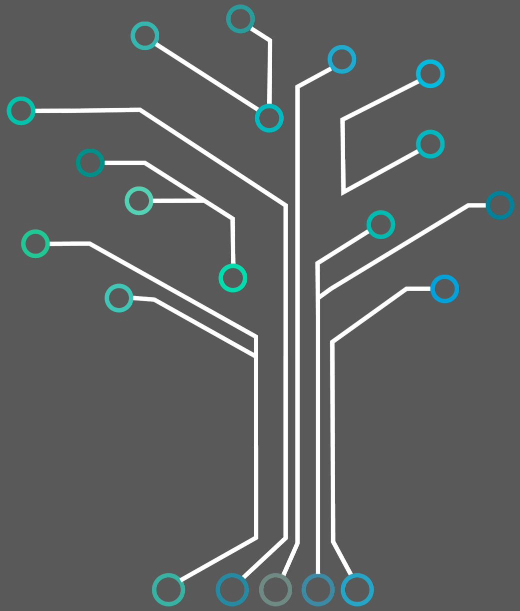
Part II: Hands-on

16:15 - 18:00

- Questions
- Setup
- Some coding



Introduction to
Deep Learning



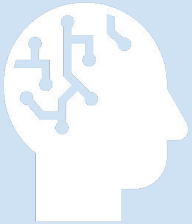
Jan 20, 2025

Overview

Machine Learning as Artificial Intelligence

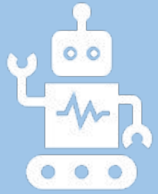
Artificial Intelligence

Any technique that enables computers to mimic human behaviour



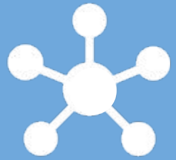
Machine Learning

Learn to perform tasks from data without being explicitly programmed



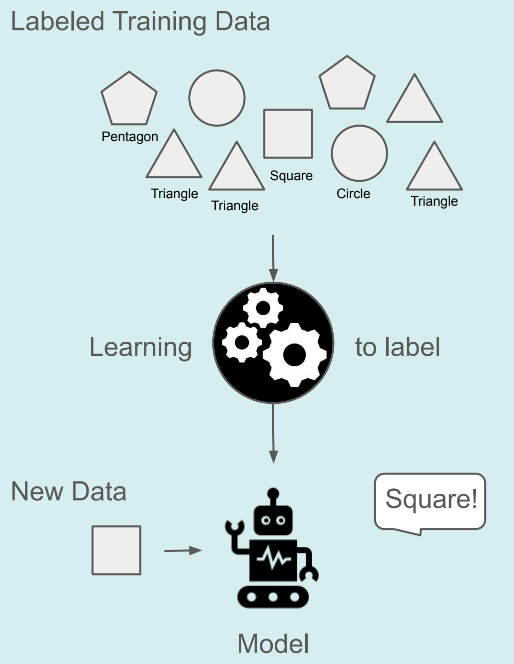
Deep Learning

Extract patterns from data using deep neural networks



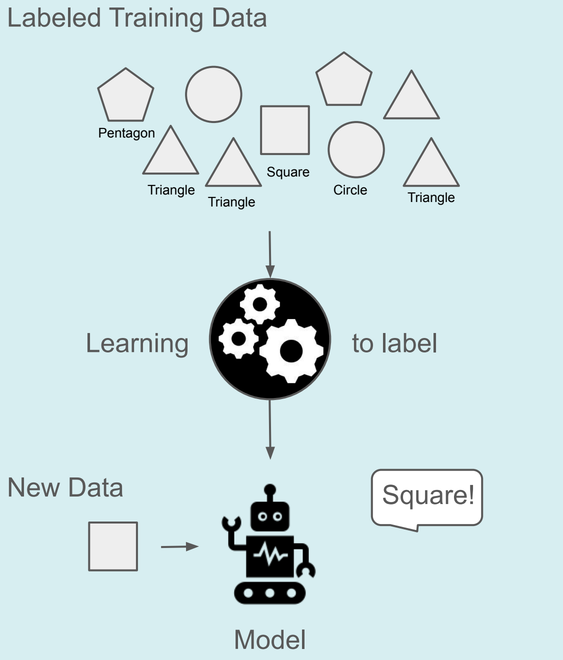
Disciplines of Machine Learning

Supervised Learning

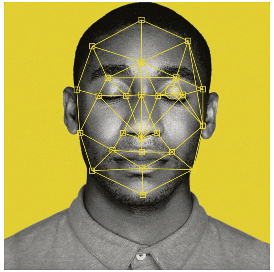


Disciplines of Machine Learning

Supervised Learning

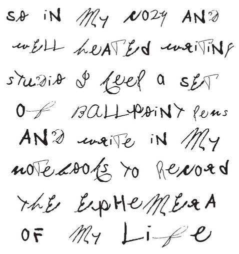


Face recognition



https://www.theguardian.com/technology/2019/jul/29/what-is-facial-recognition-and-how-sinister-is-it

Handwritten transcription



https://www.behance.net/gallery/71324093/The-Handwritten-A

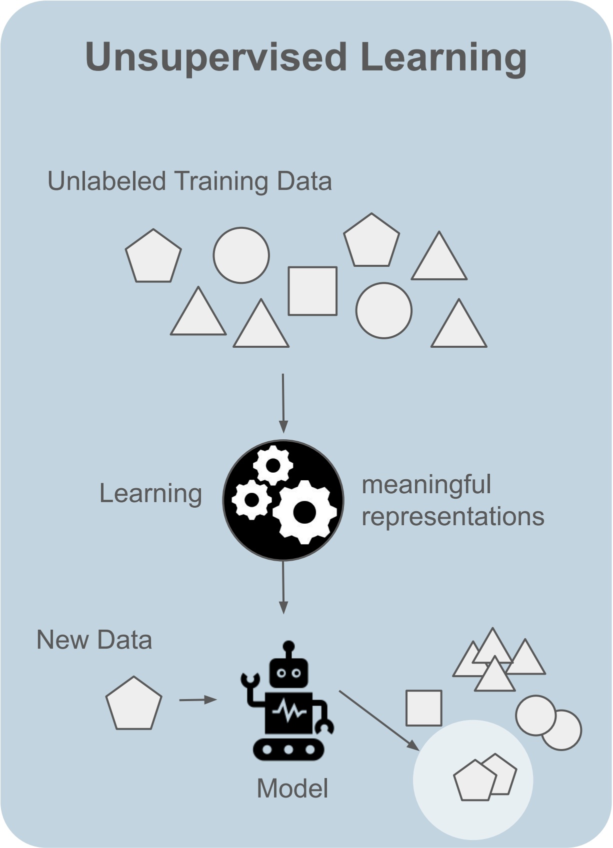
Speech recognition

https://support.apple.com/de-de/HT208336

Medical diagnosis

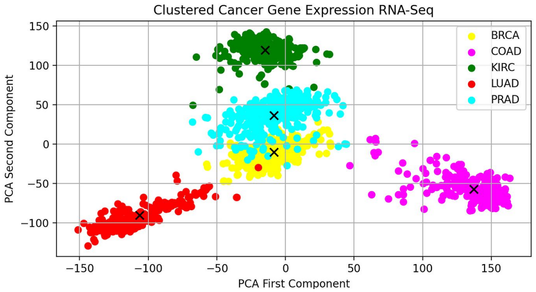
https://www.wired.com/story/fmri-ai-suicide-ideation/

Disciplines of Machine Learning



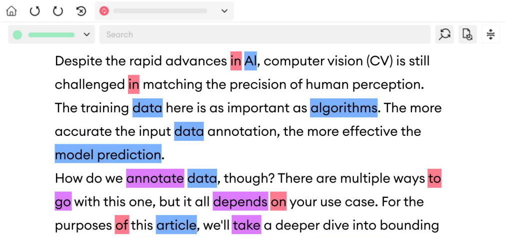
Disciplines of Machine Learning

Gene clustering



https://ernest-bonat.medium.com/building-machine-learning-clustering-models-for-gene-expression-rna-seq-data-d0e5af10416d

Language processing



https://www.superannotate.com/blog/what-is-natural-language-processing

Unsupervised Learning

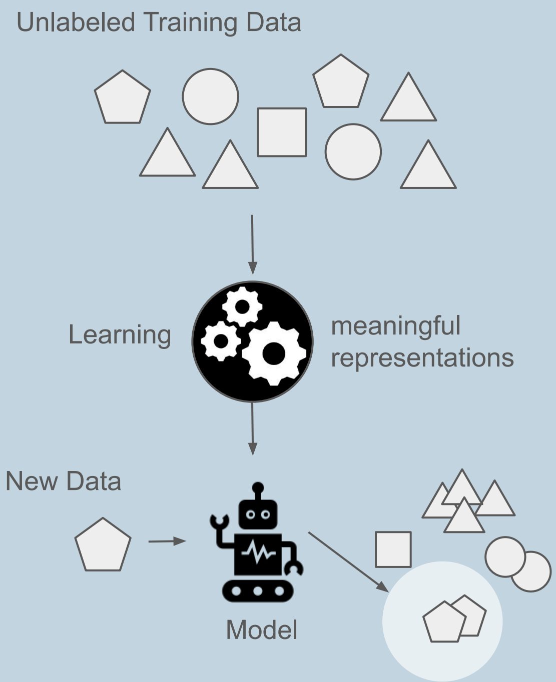


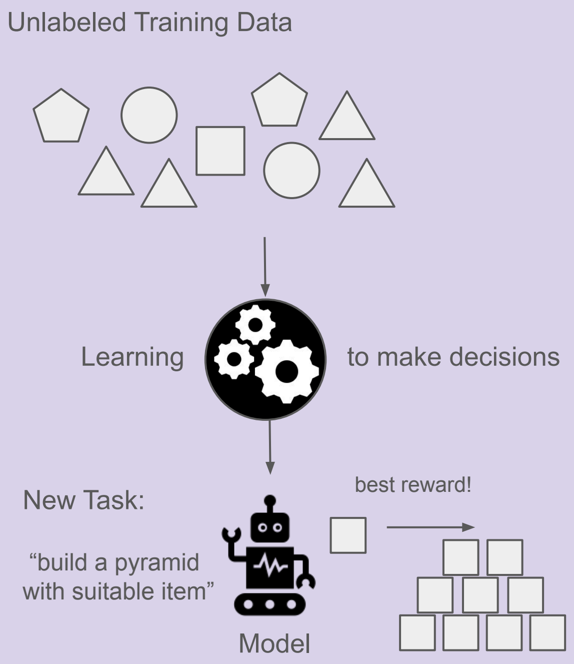
Image clustering

https://neurohive.io/en/state-of-the-art/deep-clustering-approach/

Generation tasks

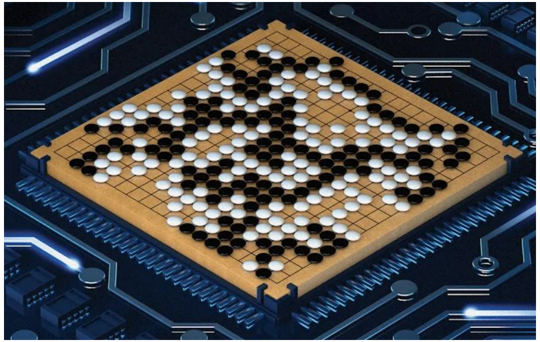
Disciplines of Machine Learning

Reinforcement Learning



Disciplines of Machine Learning

Game playing



https://deepmind.google/research/breakthroughs/alphago/

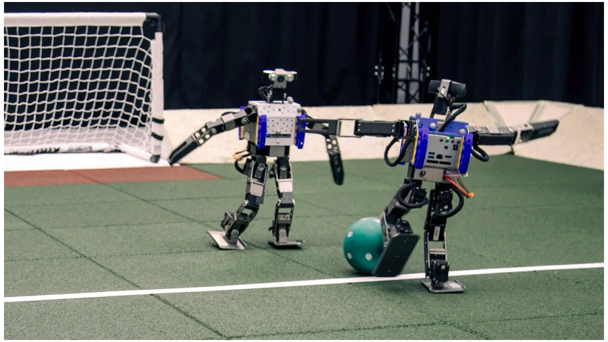
Algorithmic trading

https://www.mathworks.com/videos/reinforcement-learning-in-finance-1578033119150.html

Goal-oriented chatbots

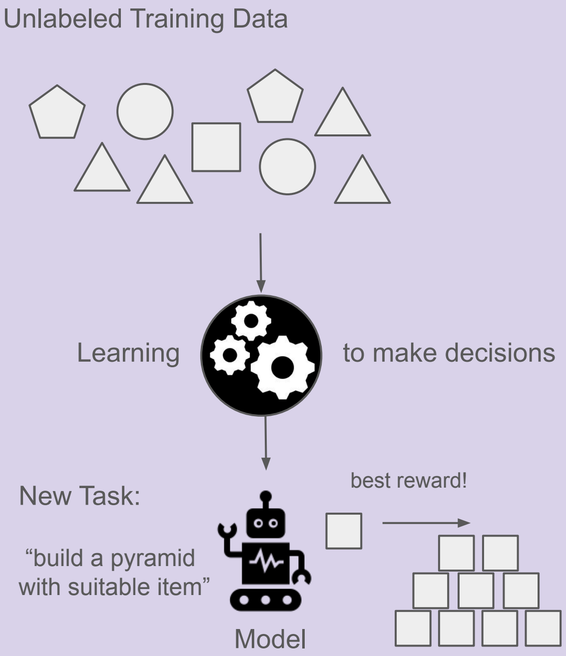
https://towardsdatascience.com/training-a-goal-oriented-chatbot-with-deep-reinforcement-learning-part-i-introduction-and-dce3af21d383

Robotics

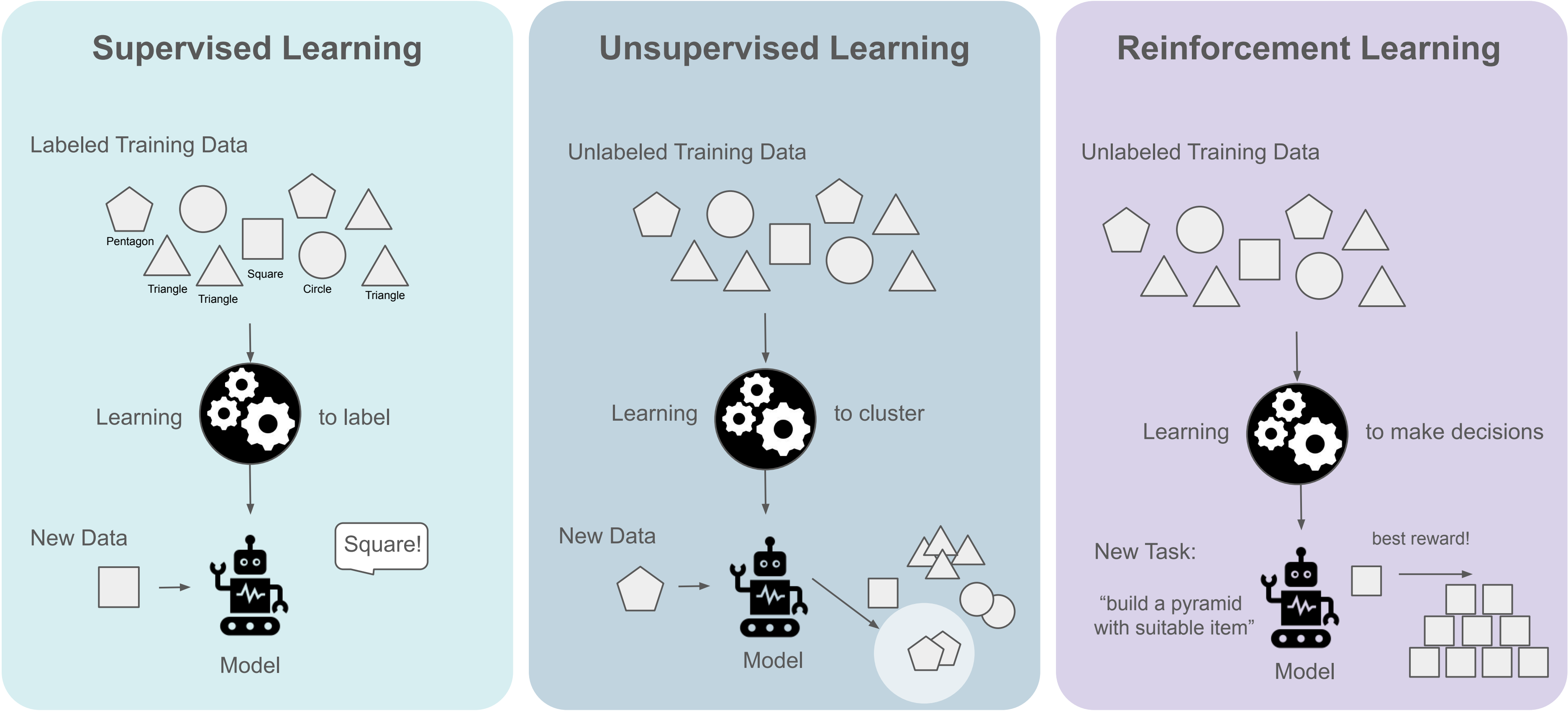


https://www.sciencenews.org/article/reinforcement-learn-ai-humanoid-robots

Reinforcement Learning



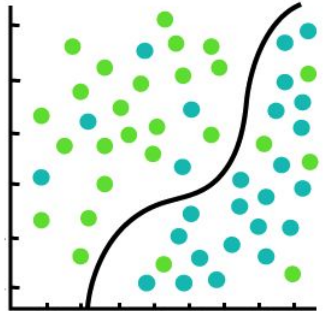
Disciplines of Machine Learning



Supervised Learning Tasks

Classification

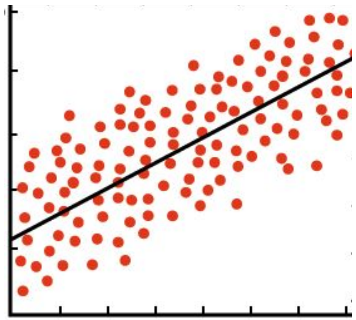
Training: learn to predict a label out of a discrete set



Testing: accuracy as # of correctly predicted

Regression

Training: predict a label as a continuous value directly

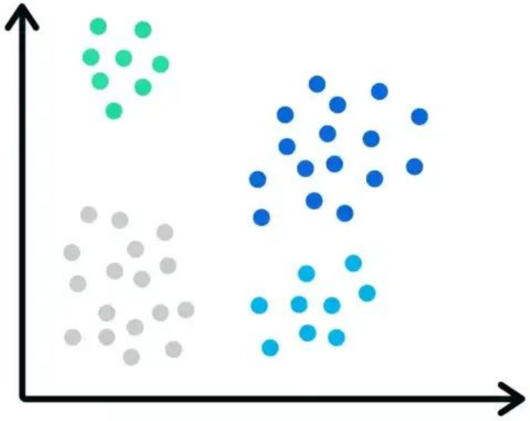


Testing: distance/similarity to actual outcomes

Unsupervised Learning Tasks

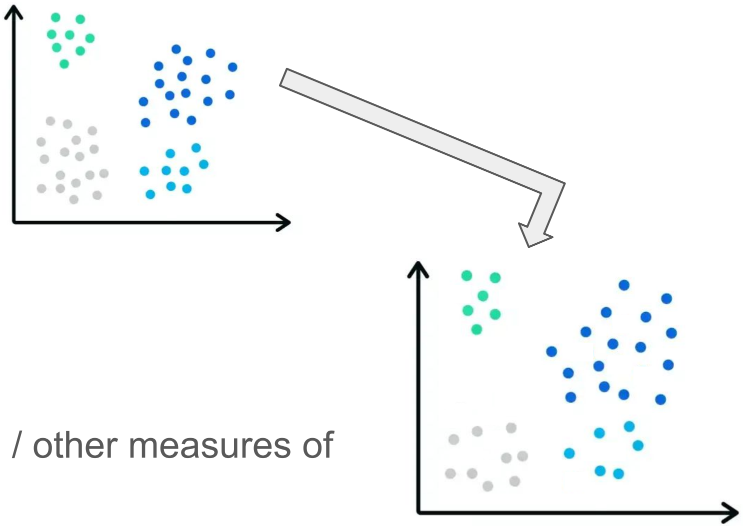
Clustering

Training: learn to identify groups



Generation

Training: create representations to sample realistic outputs

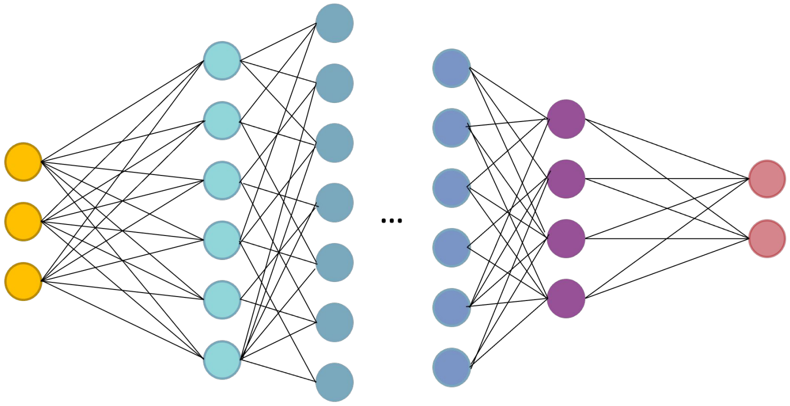


Testing? Depends on the availability of ground truth data / other measures of performance…

Figure modified from: https://towardsdatascience.com/training-a-goal-oriented-chatbot-with-deep-reinforcement-learning-part-i-introduction-and-dce3af21d383

Deep Learning

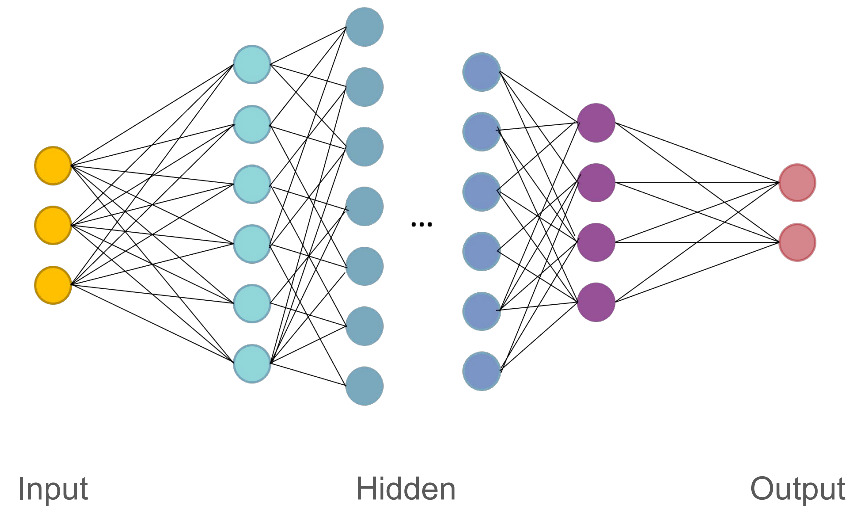
Deep Neural Networks



Input Hidden Output

Deep Learning

Deep Neural Networks

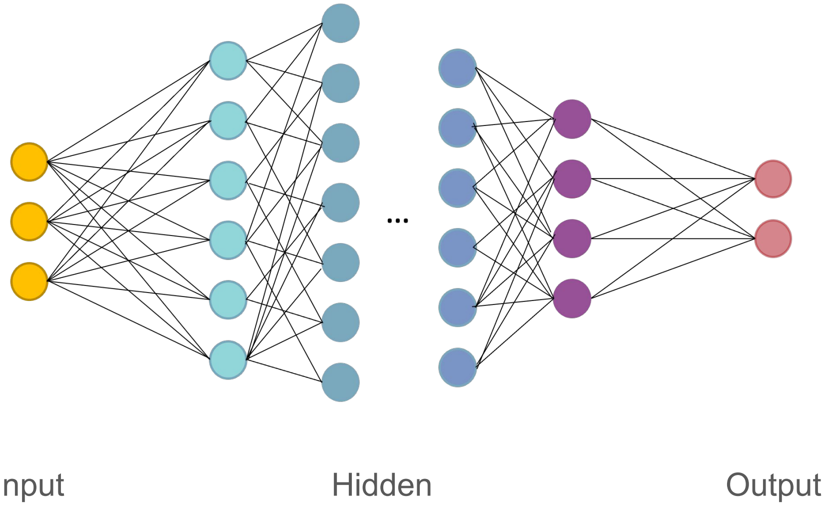


Why this?

- Hierarchical processing: several levels
- All-in-one model: human out of the loop (?!)
- Extremely expressive: can learn “anything”

Deep Learning

Deep Neural Networks



Why this?

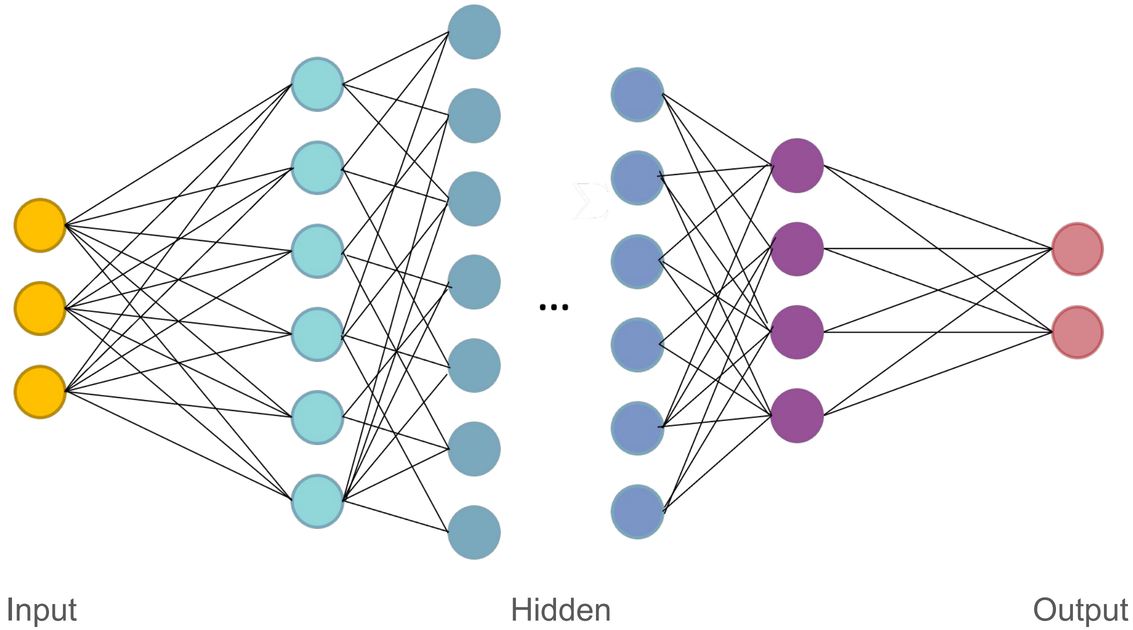
- Hierarchical processing: several levels
- All-in-one model: human out of the loop (?!)
- Extremely expressive: can learn “anything”

Why now?

- Unprecedented amount of available data
- Parallelization of computations by GPUs
- Many available toolkits

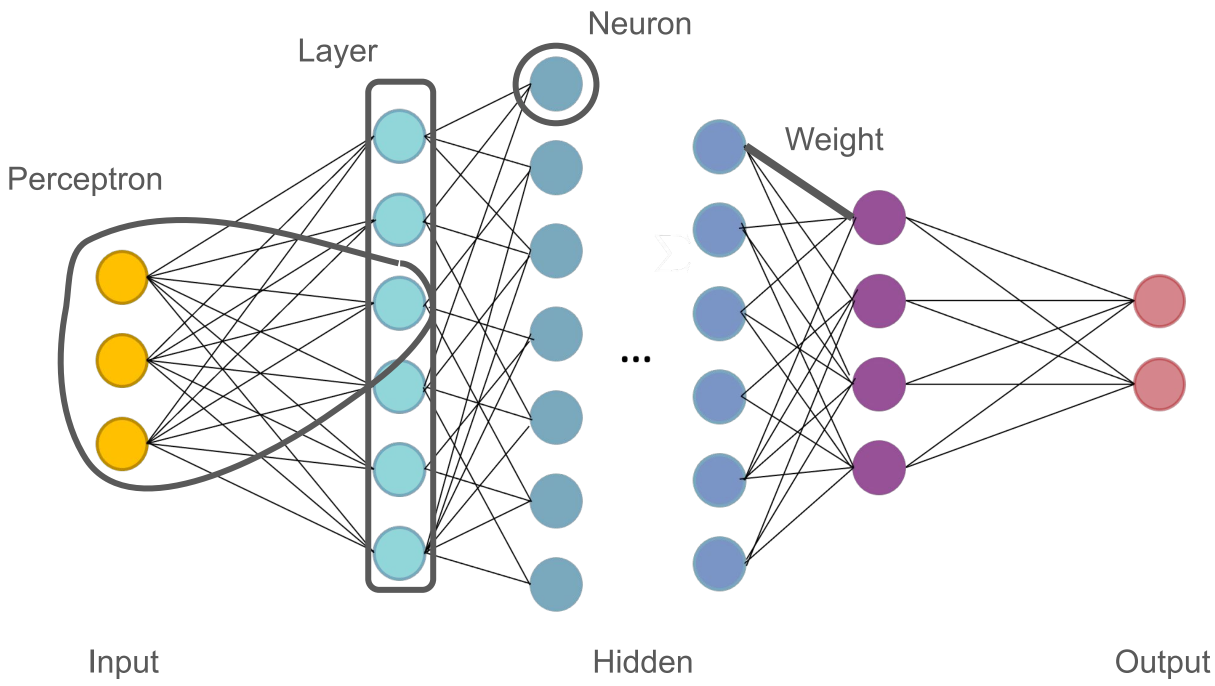
Theoretical Basics

A Neural Network



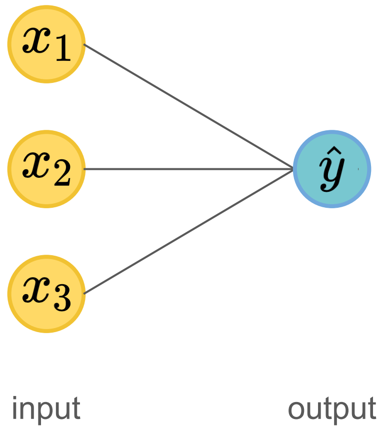
A Neural Network

"Multi-layer Perceptron"



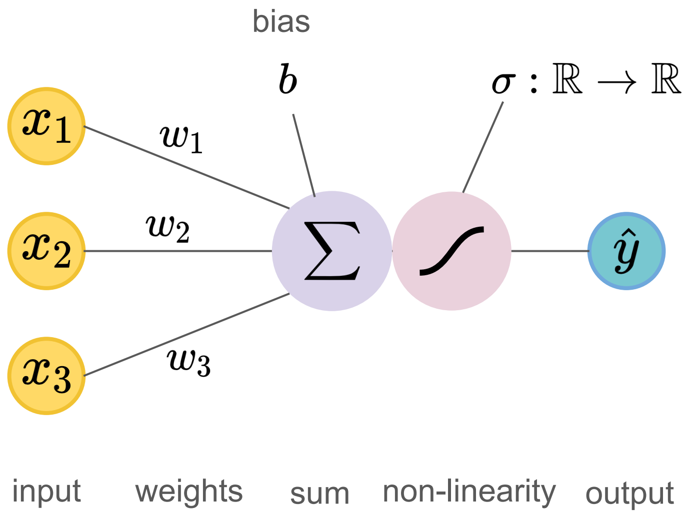
A Neural Network

Perceptron



A Neural Network

Perceptron

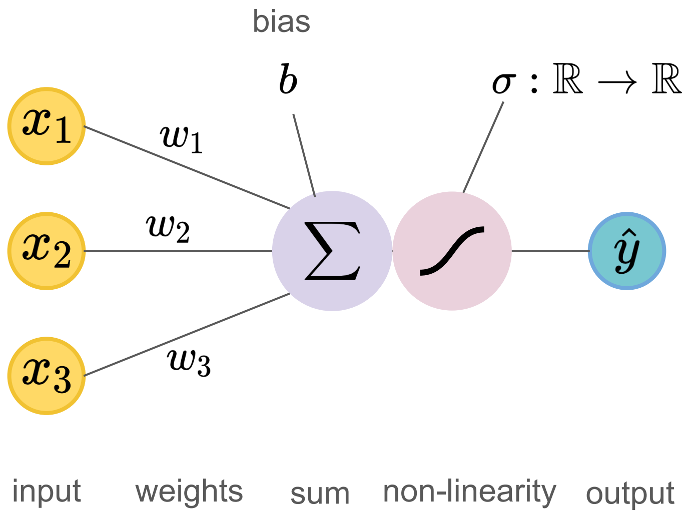


The math

$$\hat{y} = \sigma\left(\sum_{i=1}^{3} w_i \cdot x_i + b\right)$$

A Neural Network

Perceptron



The math…

$$[w_1 \quad w_2 \quad w_3] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = w^T x$$

linear combination of input

activation

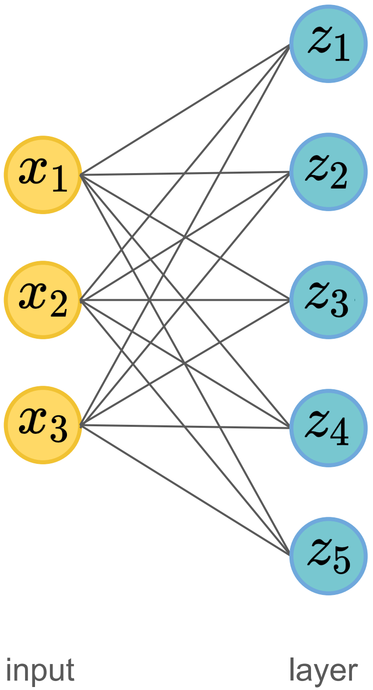
bias

$$\hat{y} = \sigma\left(\sum_{i=1}^{3} w_i \cdot x_i + b\right)$$

$$\hat{y} = \sigma\left(w^T x + b\right)$$

A Neural Network

Single Layer Network

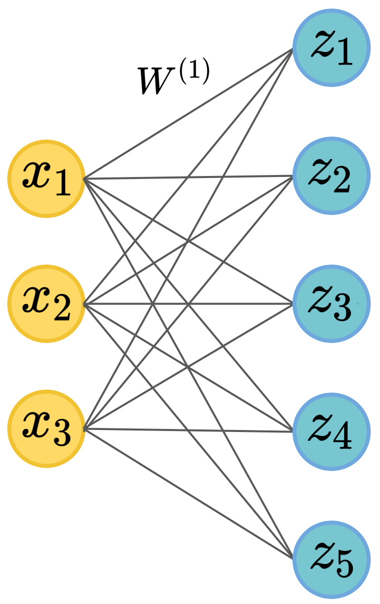


The math…

$$z_j = \sigma\left(\sum_{i=1}^{3} w_{j,i}^{(1)} \cdot x_i + b_j^{(1)}\right), \; j = 1, \ldots, 5$$

A Neural Network

Single Layer Network



The math…

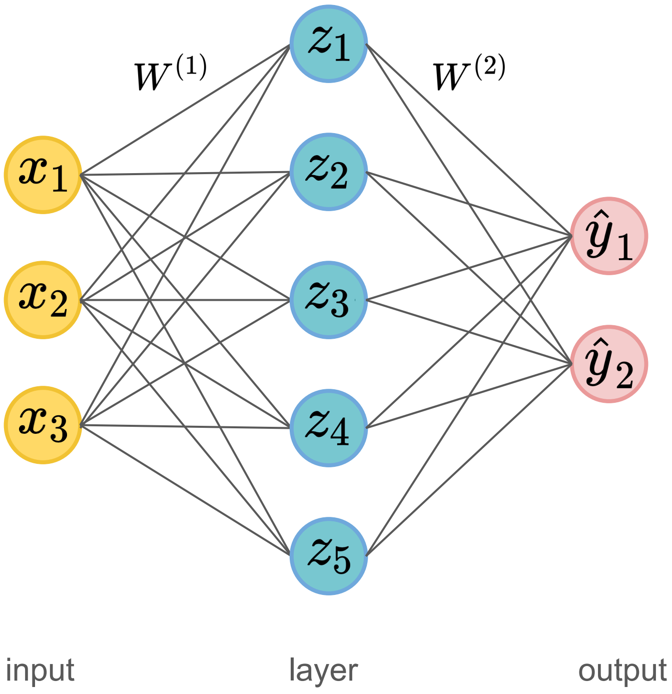
$$\begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \\ w_{4,1}^{(1)} & w_{4,2}^{(1)} & w_{4,3}^{(1)} \\ w_{5,1}^{(1)} & w_{5,2}^{(1)} & w_{5,3}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = W^{(1)}x + b^{(1)}$$

$$z_j = \sigma\left(\sum_{i=1}^{3} w_{j,i}^{(1)} \cdot x_i + b_j^{(1)}\right), \; j = 1, \ldots, 5$$

$$z = \sigma\left(W^{(1)}x + b^{(1)}\right)$$

A Neural Network

Single Layer Network



The math…

$$\begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \\ w_{4,1}^{(1)} & w_{4,2}^{(1)} & w_{4,3}^{(1)} \\ w_{5,1}^{(1)} & w_{5,2}^{(1)} & w_{5,3}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = W^{(1)}x + b^{(1)}$$

$$z_j = \sigma\left(\sum_{i=1}^{3} w_{j,i}^{(1)} \cdot x_i + b_j^{(1)}\right), \ j = 1, \ldots, 5$$

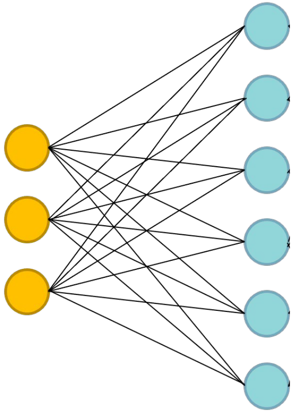
$$z = \sigma\left(W^{(1)}x + b^{(1)}\right)$$

$$\hat{y}_j = \sigma\left(\sum_{i=1}^{5} w_{j,i}^{(2)} \cdot z_i + b_j^{(2)}\right), \ j = 1, 2$$

$$\hat{y} = \sigma\left(W^{(2)}z + b^{(2)}\right)$$

A Neural Network

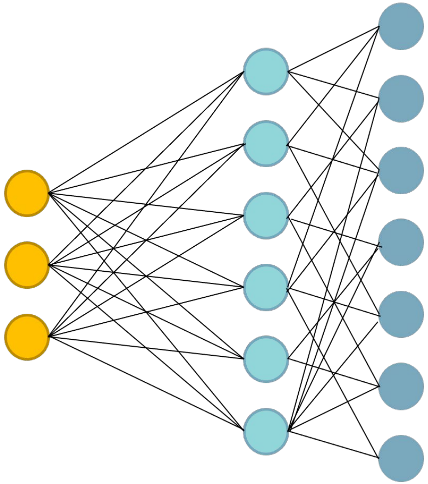
Multi-layer Network



$$\hat{y} = \sigma\left(W^{(1)}x + b^{(1)}\right)$$

A Neural Network

Multi-layer Network

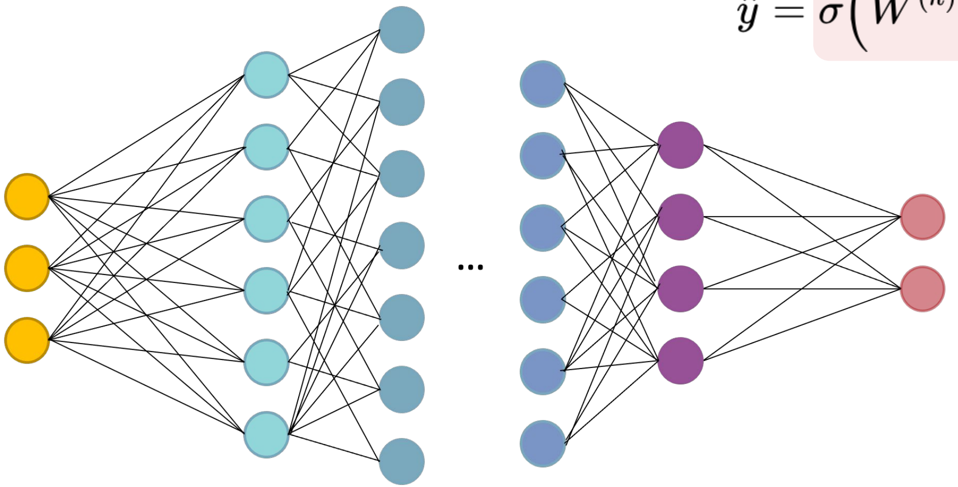


$$\hat{y} = \sigma\left(W^{(2)}\sigma\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right)$$

A Neural Network

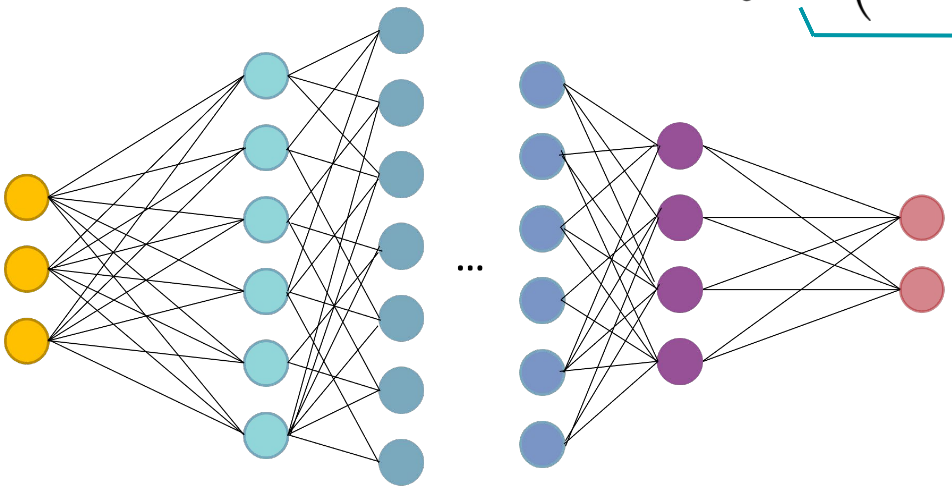
Multi-layer Network

$$\hat{y} = \sigma\Big(W^{(k)} \ldots \sigma\Big(W^{(2)}\sigma\Big(W^{(1)}x + b^{(1)}\Big) + b^{(2)}\Big) \ldots + b^{(k)}\Big)$$



A Neural Network

Multi-layer Network



network output ("prediction")

$$\hat{y} = \sigma\Big(W^{(k)} \dots \sigma\Big(W^{(2)}\sigma\Big(W^{(1)}x + b^{(1)}\Big) + b^{(2)}\Big) \dots + b^{(k)}\Big)$$

$$=: \Phi(x; \theta)$$

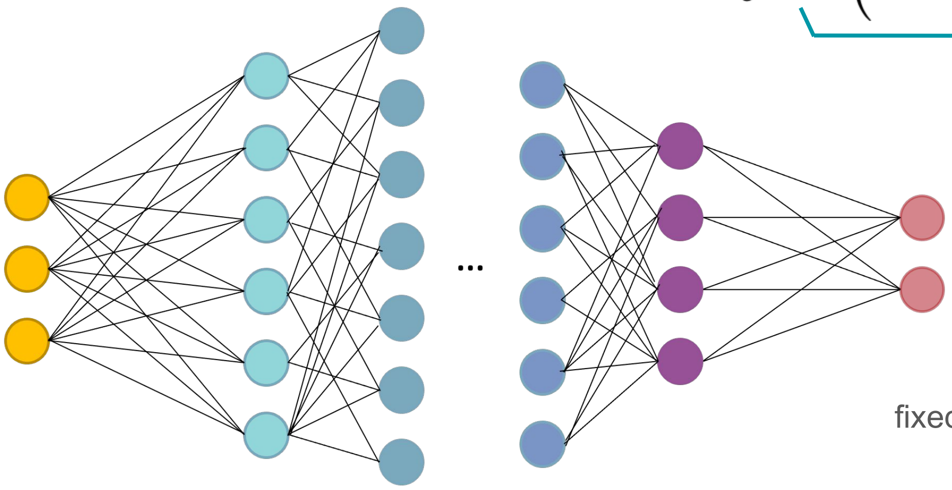
input

network parameters
= weights

$$\theta = \left\{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \dots, W^{(k)}, b^{(k)}\right\}$$

A Neural Network

Multi-layer Network



network output ("prediction")

$$\hat{y} = \sigma\Big(W^{(k)} \dots \sigma\Big(W^{(2)}\sigma\Big(W^{(1)}x + b^{(1)}\Big) + b^{(2)}\Big) \dots + b^{(k)}\Big)$$

$$=: \Phi(x; \theta)$$

input

network parameters
= weights

$$\theta = \left\{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \dots, W^{(k)}, b^{(k)}\right\}$$

Φ has two faces

fixed θ

Evaluation function

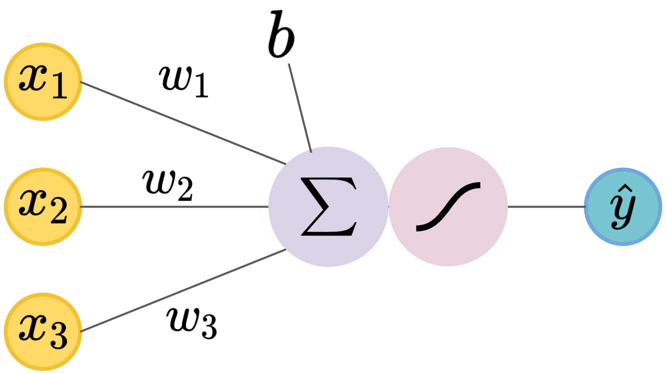
$$\Phi_x : \mathbb{R}^d \to \mathbb{R}^n$$

fixed x

Training function

$$\Phi_\theta : \mathbb{R}^{|\theta|} \to \mathbb{R}^n$$

Non-Linearities: Activation Functions



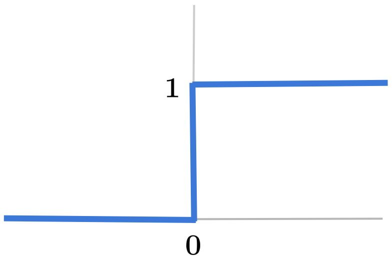
Biological motivation:
activate neuron if threshold *b* is exceeded

$$\sum_{i=1}^{3} w_i \cdot x_i > b \longrightarrow 1$$

activate!

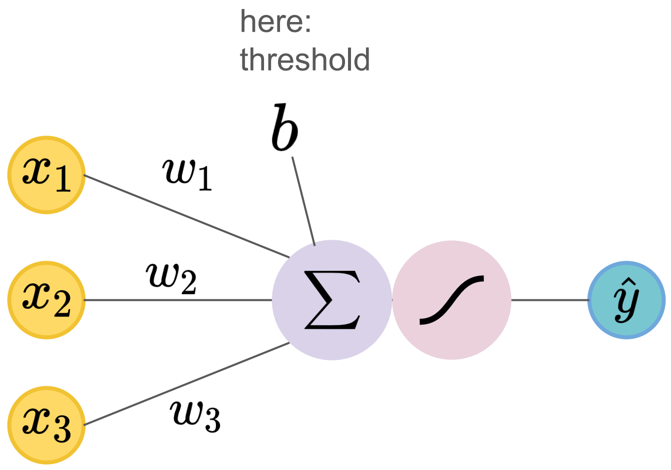
$$\sum_{i=1}^{3} w_i \cdot x_i \leq b \longrightarrow 0$$

discard!



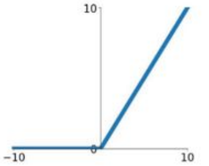
Heaviside (step) function

Non-Linearities: Activation Functions



ReLU

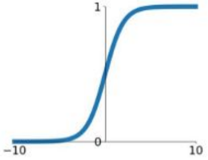
$\max(0, x)$



the “default”

Sigmoid

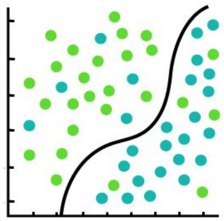
$\sigma(x) = \frac{1}{1+e^{-x}}$

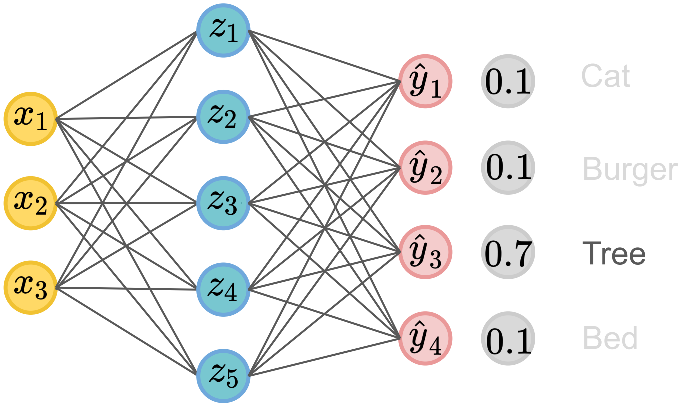


output within [0,1]

Supervised Learning Tasks

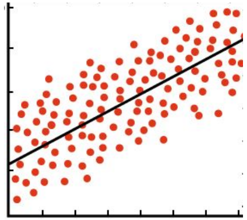
Classification

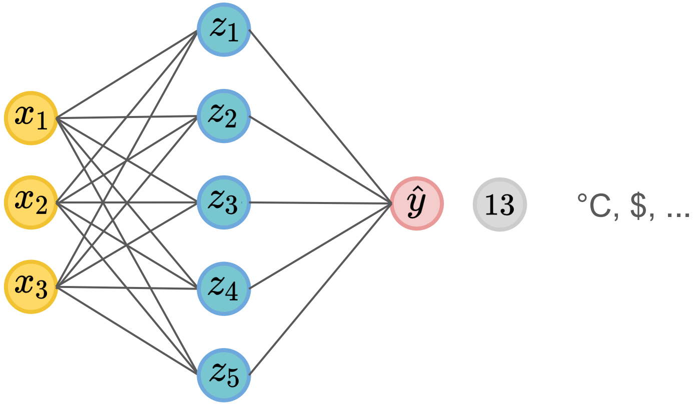




$\hat{y}_i \in [0, 1]$ $\rightarrow$ probability distribution (soft-max)

Regression





predict the value directly

“Expressive Power”

What can a neural network learn?

“Expressive Power”

What can a neural network learn?

anything

“Expressive Power”

What can a neural network learn?

“anything”

Universal Approximation Theorem

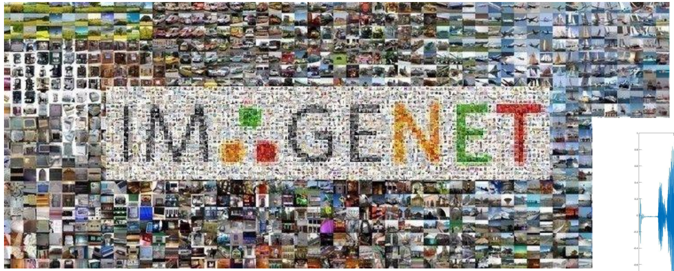
“Neural networks with a non-polynomial activation function can approximate any continuous function arbitrary well”

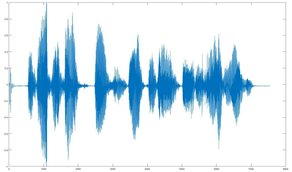
For any continuous f there is $\theta^{\#}$ st. $\left\|\Phi\left(x;\theta^{\#}\right)-f(x)\right\|<\varepsilon$

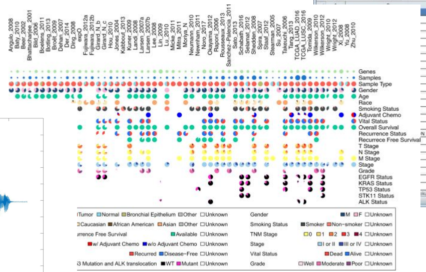
Data

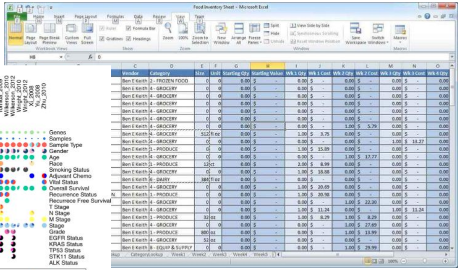
What is a dataset?

- An <u>organized</u> collection of data
  - One “unit” of data = an instance / data point
  - Information about a data point = *Features*
  - Labels or other annotations often included
    → Required for supervised tasks but not (necessarily) for unsupervised
  - Normalize it: $[0,1], [-1,1], [0,256], (x_{(k)} - \mu)/\sigma$



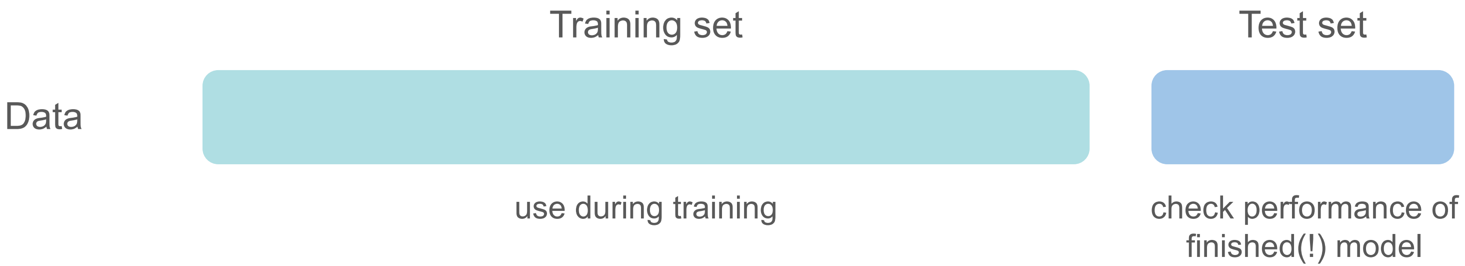






Properties of a (good) dataset

- What about dataset size…?
  - Defined entirely by the task (from dozens/hundreds to millions)
  - Only certainty is that “the more the merrier”, but also “the more representative the merrier”
- <u>Do not forget:</u> the **data split** (~80/20%)

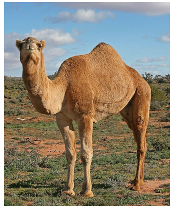


Training

Training

Supervised learning:

Given samples of training data with corresponding labels (x, y)



camel

cat

Pikachu

input x (matrix with values)

label y (binary vector with a 1 at the correct class)

- 0 camel
- 0 cat
- 1 Pikachu

Goal: Optimize the weights such that $\Phi($  $)$ = cat for all of the samples in the training data.

but also $\Phi($  $)$ = cat for samples outside!

Training

How to achieve this goal?

Loss function (*error, cost*) $\mathcal{L}\left(\hat{y}, y\right)$: how good is prediction $\hat{y}$ compared to the true label y

- Zero-one loss: $\mathcal{L}_{0-1}\left(\hat{y}, y\right) = 1 \text{ if } \hat{y} = y \text{ and } 0 \text{ else}$ *- Is it exactly the same or not?*
- Square loss (L2): $\mathcal{L}_{sq}(\hat{y}, y) = \|\hat{y} - y\|^2$ *- Euclidean distance*
- Cross entropy loss: $\mathcal{L}_{ce}(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$ *- maximize likelihood*

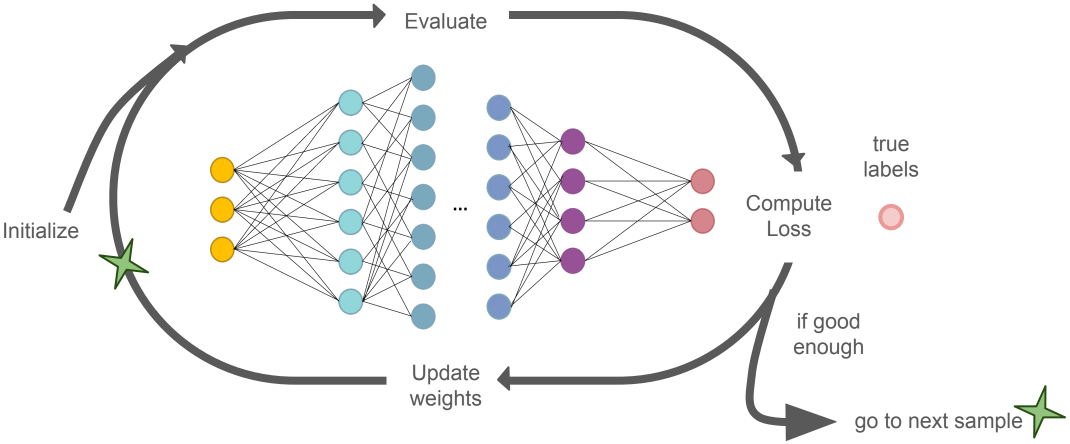
→ **minimizing the loss function will improve the prediction!**

Training

Idea: Start with random weights

1) Take a sample  and measure good bad the prediction is: $\Phi($  $)$ = Pikachu
2) Update the weights to improve the prediction (i.e., loss decreases): $\Phi($  $)$ = cat

Repeat the process for every sample in the training data set.

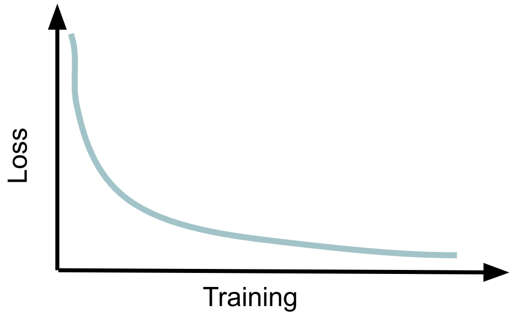


Training

GOAL: find a weight update rule that produces a sequence that gradually decreases the loss.

$$\left(\theta^{(0)}, \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(k)}\right) \text{ such that } \mathcal{L}\left(\Phi\left(x;\theta^{(0)}\right), y\right) \geq \mathcal{L}\left(\Phi\left(x;\theta^{(1)}\right), y\right) \geq \mathcal{L}\left(\Phi\left(x;\theta^{(2)}\right), y\right) \geq \ldots \geq \mathcal{L}\left(\Phi\left(x;\theta^{(k)}\right), y\right)$$

As training progresses, later weights should result in smaller losses.



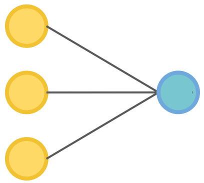
And do it over the whole training set:

$$\theta^{\#} := \arg\min_{\theta} \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}\left(\Phi\left(x_{(k)};\theta\right), y_{(k)}\right)$$

Find the weights which result in minimal loss over the whole training set.

→ **non-linear, non-convex optimization problem!**

Special Case: Linear Perceptron

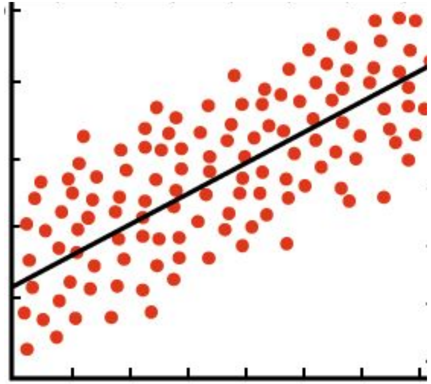


$$\hat{y} = \Phi(x) = w^T x = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Loss $\mathcal{L}_{sq}(\hat{y}, y) = \|\hat{y} - y\|^2$

Least squares problem!

→ **Linear Regression!**



Weight Updates: A simple optimization technique

Gradient Descent

Gradient of the loss: “how does the loss change, if a weight changes?”

$$\nabla L\left(w_1, \ldots, w_n\right) = \begin{bmatrix} \frac{\partial L}{w_1} \\ \vdots \\ \frac{\partial L}{w_n} \end{bmatrix}$$

→ points to **steepest ascent** (i.e., the direction to change the weights, so that there is maximal change in the loss)

Weight Updates: A simple optimization technique

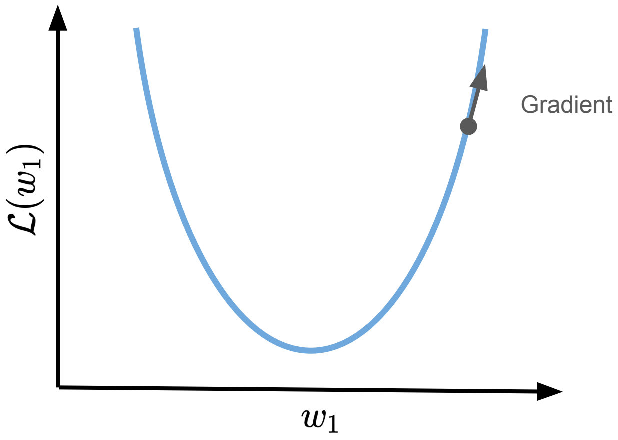
Gradient Descent

Gradient of the loss:

$$\nabla L\left(w_1, \ldots, w_n\right) = \begin{bmatrix} \frac{\partial L}{w_1} \\ \vdots \\ \frac{\partial L}{w_n} \end{bmatrix}$$

→ points to **steepest ascent**

$n = 1$



Weight Updates: A simple optimization technique

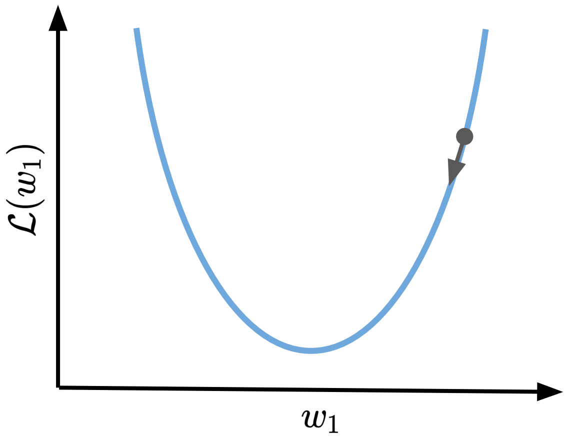
Gradient Descent

Gradient of the loss:

$$\nabla L\left(w_1, \ldots, w_n\right) = \begin{bmatrix} \frac{\partial L}{w_1} \\ \vdots \\ \frac{\partial L}{w_n} \end{bmatrix}$$

→ points to **steepest ascent**

$n = 1$



go opposite direction of steepest ascent

Weight Updates: A simple optimization technique

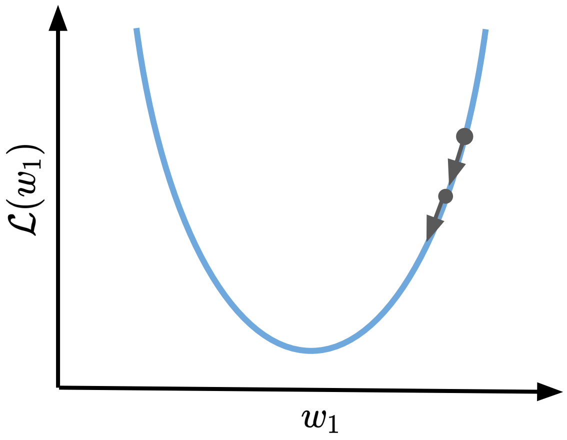
Gradient Descent

Gradient of the loss:

$$\nabla L\left(w_1, \ldots, w_n\right) = \begin{bmatrix} \frac{\partial L}{w_1} \\ \vdots \\ \frac{\partial L}{w_n} \end{bmatrix}$$

→ points to **steepest ascent**

$n = 1$



go opposite direction of steepest ascent

Weight Updates: A simple optimization technique

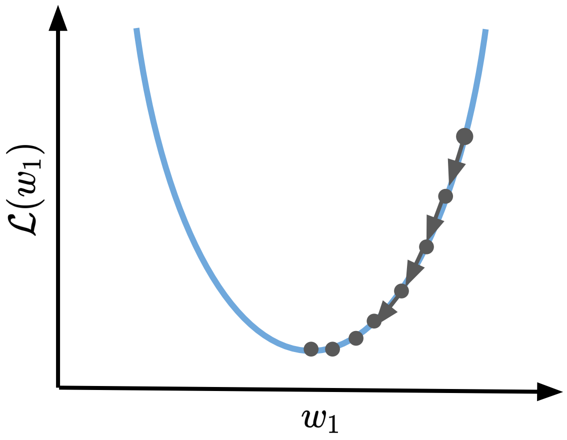
Gradient Descent

Gradient of the loss:

$$\nabla L\left(w_1, \ldots, w_n\right) = \begin{bmatrix} \frac{\partial L}{w_1} \\ \vdots \\ \frac{\partial L}{w_n} \end{bmatrix}$$

→ points to **steepest ascent**

$n = 1$



go opposite direction of steepest ascent

$$w_1^{new} \leftarrow w_1^{old} - \eta \cdot \nabla \mathcal{L}\left(w_1^{old}\right)$$

Weight Updates: A simple optimization technique

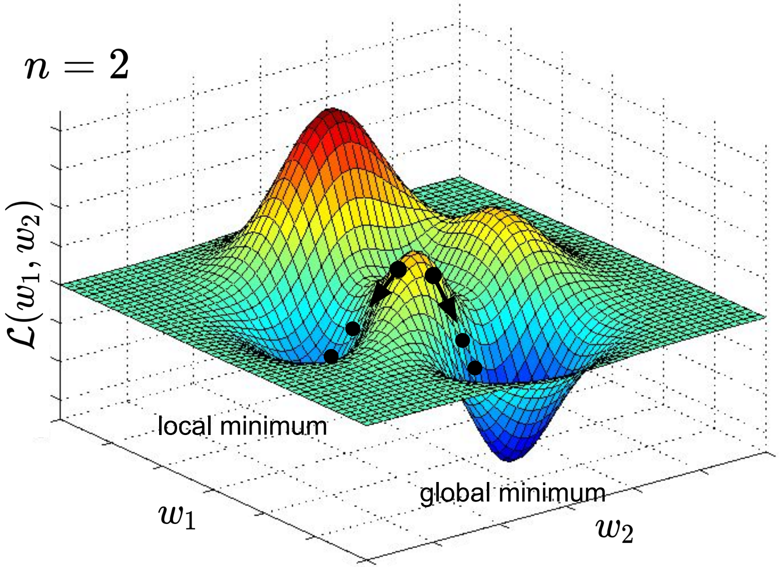
Gradient Descent

Gradient of the loss:

$$\nabla L\left(w_1, \ldots, w_n\right) = \begin{bmatrix} \frac{\partial L}{w_1} \\ \vdots \\ \frac{\partial L}{w_n} \end{bmatrix}$$

→ points to **steepest ascent**

$n = 2$



go opposite direction of steepest ascent

$$w_1^{new} \leftarrow w_1^{old} - \eta \cdot \nabla \mathcal{L}\left(w_1^{old}, w_2^{old}\right)$$
$$w_2^{new} \leftarrow w_2^{old} - \eta \cdot \nabla \mathcal{L}\left(w_1^{old}, w_2^{old}\right)$$

Weight Updates: A simple optimization technique

Gradient Descent

Gradient of the loss:

$$\nabla L\left(w_1, \ldots, w_n\right) = \begin{bmatrix} \frac{\partial L}{w_1} \\ \vdots \\ \frac{\partial L}{w_n} \end{bmatrix}$$

→ points to **steepest ascent**

Algorithm

Initialize $\theta^{(0)}, \eta > 0$

Until convergence:

Compute gradient $\nabla \mathcal{L}\left(\theta^{(n)}\right)$

Update weights $\theta^{n+1} \leftarrow \theta^{n} - \eta \cdot \nabla \mathcal{L}\left(\theta^{(n)}\right)$

"learning rate"

Return weights

Training on Batches

Gradient descent is very expensive…

Example: A single step of gradient descent for AlexNet (neural network ~160M parameters) on ImageNet (dataset ~1.2M images) requires ~2*10^14 flops!

Training on Batches

Gradient descent is very expensive…

Example: A single step of gradient descent for AlexNet (neural network ~160M parameters) on ImageNet (dataset ~1.2M images) requires ~2*10^14 flops!

Train on small batches of the dataset!

“Training with large minibatches is bad for your health. More importantly, it’s bad for your test error. Friends don’t let friends use minibatches larger that 32.”

-Yann LeCun

Evaluation

Training-Test

Training set

Test set

Data

during training: monitor loss/error

after training:
check performance

Training-Test

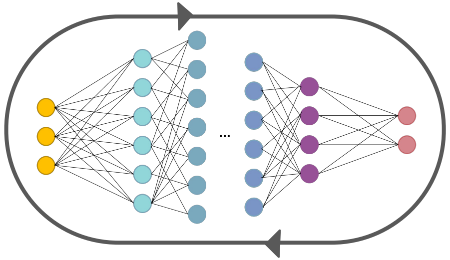
Training set

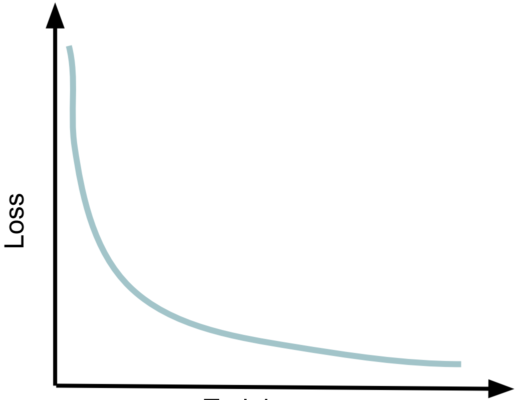
Test set

Data

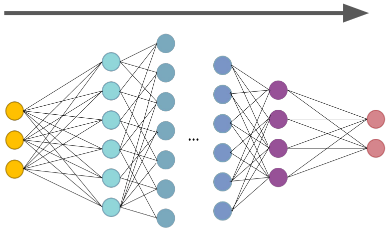
during training: monitor loss/error

after training:
check performance





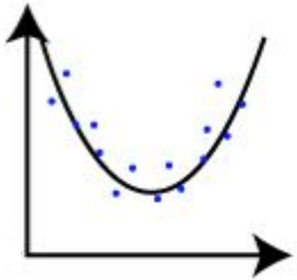
Evaluate on unseen data



Bias-Variance Tradeoff

Over- and underfitting

Example:
Learn a second-degree polynomial from noisy observations

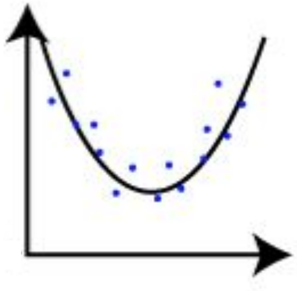


ground truth
deg = 2

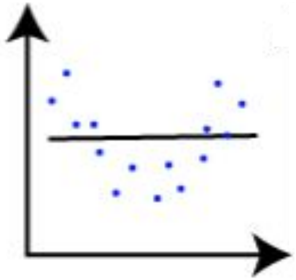
Bias-Variance Tradeoff

Over- and underfitting

Example:
Learn a second-degree polynomial from noisy observations



ground truth
deg = 2



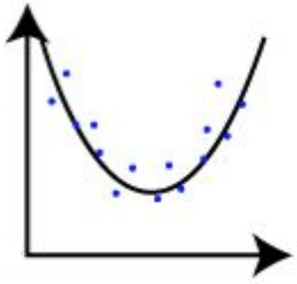
underfitting
deg too low

Simple model:
high bias,
good capturing of essentials,
bad fit

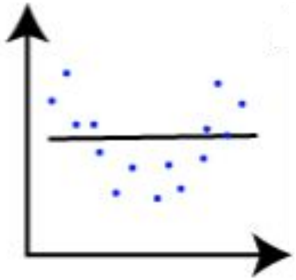
Bias-Variance Tradeoff

Over- and underfitting

Example:
Learn a second-degree polynomial from noisy observations



ground truth
deg = 2



underfitting
deg too low

overfitting
deg too high

Simple model:
high bias,
good capturing of essentials,
bad fit

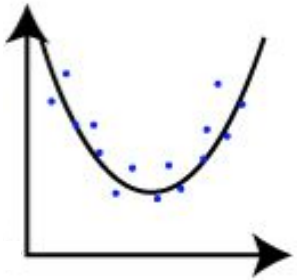
Complex model:
high variance,
good fit to data,
too specific

https://shapeofdata.wordpress.com

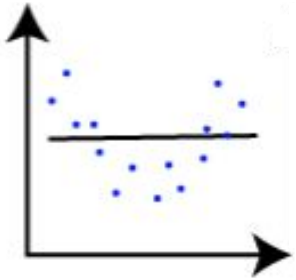
Bias-Variance Tradeoff

Over- and underfitting

Example:
Learn a second-degree polynomial from noisy observations



ground truth
deg = 2



underfitting
deg too low

overfitting
deg too high

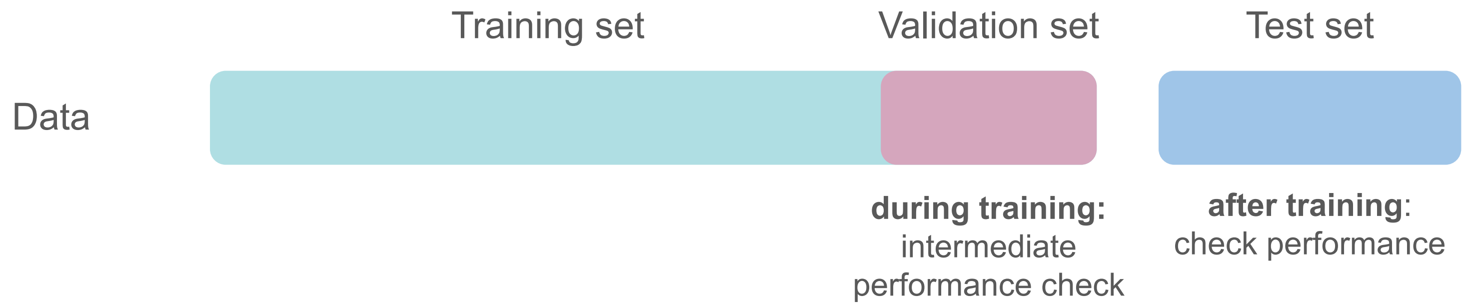
Simple model:
high bias,
good capturing of essentials,
bad fit

Complex model:
high variance,
good fit to data,
too exact

Trade-off between
model assumptions (bias) and
model complexity (variance)

https://shapeofdata.wordpress.com

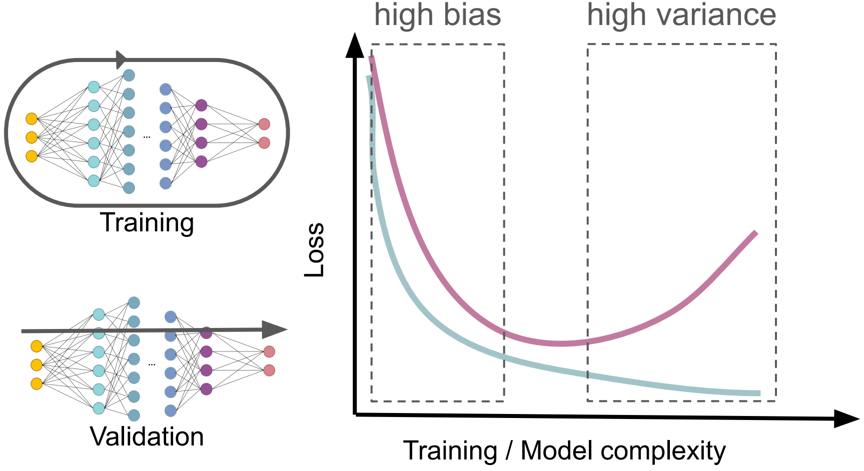
Training-Validation-Test



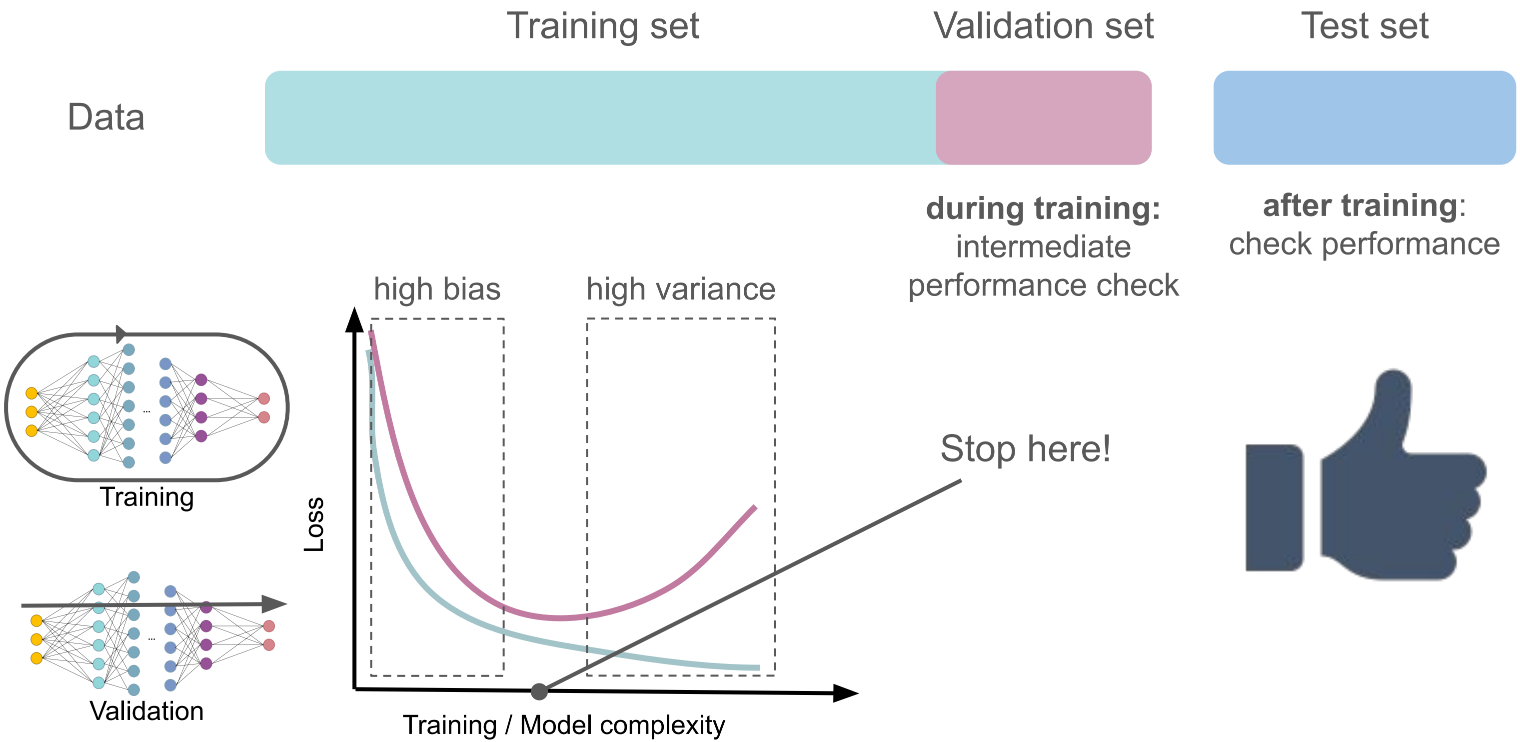
Training-Validation-Test

during training: intermediate performance check

after training: check performance



Training-Validation-Test

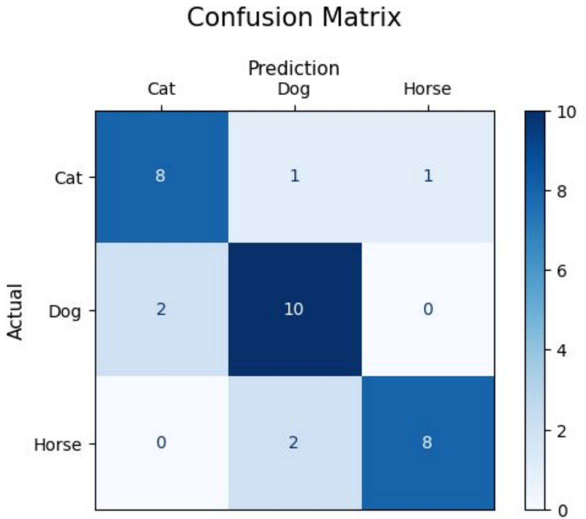


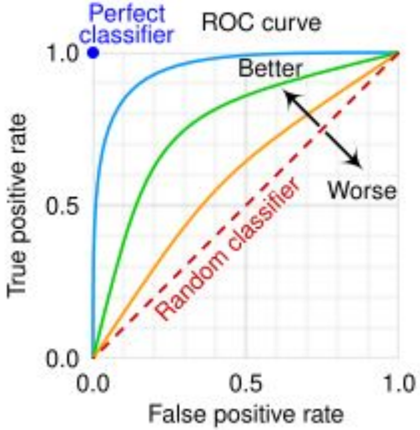
Metrics of performance

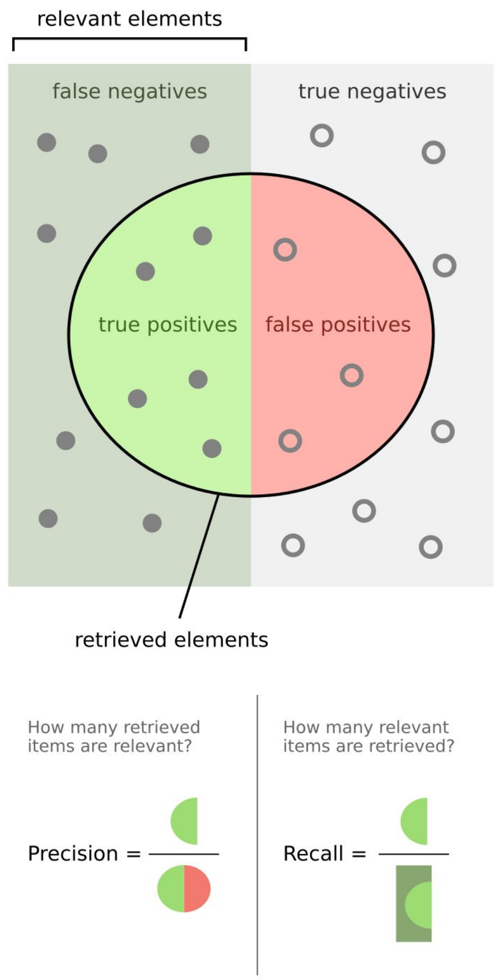
- Defined by the task: MSE, accuracy, mAP, etc…

$$accuracy = \frac{number\, of\, correct\, predictions}{total\, number\, of\, predictions}$$

- In case of **classification**:

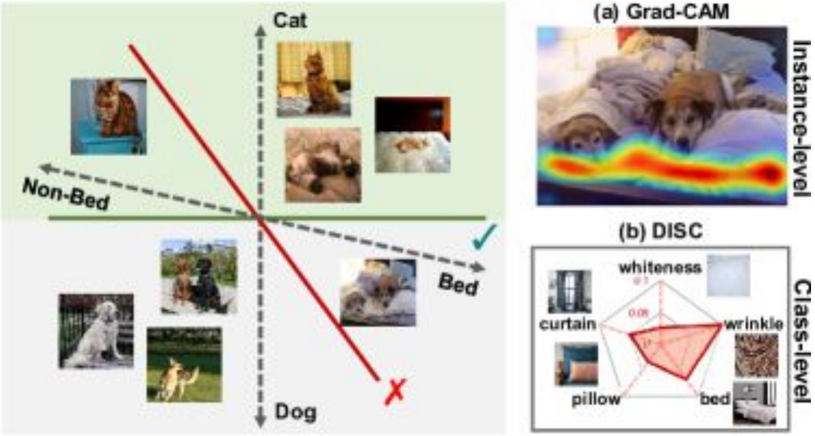






Interpretability

- XAI: steering away from the black box
- Crucial in high-responsibility decision making, e.g. medicine
- **TOOLS:** explainable architecture, post-hoc analysis, etc.



Wu et al., 2023: Discover and Cure - Concept-aware Mitigation of Spurious Correlation

Bias

- Mitigating bias
  - Especially important in decision making with a social effect (e.g., granting parole [1])
- **TOOLS:** metrics to assess group fairness (demographic parity, equalized odds, etc.), transparency about biases in the data collection process…

	WHITE	AFRICAN AMERICAN
Labeled Higher Risk, But Didn't Re-Offend	23.5%	44.9%
Labeled Lower Risk, Yet Did Re-Offend	47.7%	28.0%

[1]: Angwin et al., 2016: Machine Bias

Design and Techniques

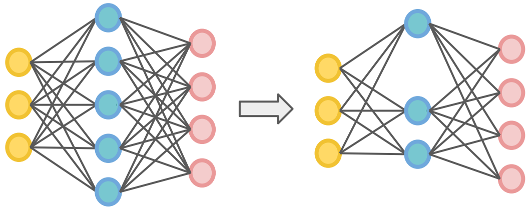
Common Techniques

Regularizing: $$\min_{\theta} \mathcal{L}\left(\theta\right) + \lambda \cdot r\left(\theta\right)$$

regularization term of the network weights

...often $r(\theta) = \|\theta\|^p$

Dropout: set weights to zero at random



Stochastic Gradient Descent (SGD): use the gradient of a randomly selected subset

Batch normalization: normalize the samples w.r.t. to the other samples in the batch

Popular architectures

Convolutional neural networks: apply “filters” to extract spatial features, textures, patterns, etc.

- Popular choice in image processing.
- Examples: VGG-16, VGG-19, AlexNet, etc.

Popular architectures

Convolutional neural networks: apply “filters” to extract spatial features, textures, patterns, etc.

- Popular choice in image processing.
- Examples: VGG-16, VGG-19, AlexNet, etc.

Autoencoders: learn a compact statistical representation of the data and sample from it.

- Useful in dimensionality reduction, data generation, denoising, etc.
- Example: Variational Autoencoders (VAE)

Popular architectures

Convolutional neural networks: apply “filters” to extract spatial features, textures, patterns, etc.

- Popular choice in image processing.
- Examples: VGG-16, VGG-19, AlexNet, etc.

Autoencoders: learn a compact statistical representation of the data and sample from it.

- Useful in dimensionality reduction, data generation, denoising, etc.
- Example: Variational Autoencoders (VAE)

Residual neural networks: use shortcut connections to skip layers (helps with vanishing gradients).

- Useful in applications requiring large networks: image segmentation, object detection, etc.
- Example: ResNet

Popular architectures

Convolutional neural networks: apply “filters” to extract spatial features, textures, patterns, etc.
- Popular choice in image processing.
- Examples: VGG-16, VGG-19, AlexNet, etc.

Autoencoders: learn a compact statistical representation of the data and sample from it.
- Useful in dimensionality reduction, data generation, denoising, etc.
- Example: Variational Autoencoders (VAE)

Residual neural networks: use shortcut connections to skip layers (helps with vanishing gradients).
- Useful in applications requiring large networks: image segmentation, object detection, etc.
- Example: ResNet

Transformers: capture relationships in sequential data by considering the whole context.
- Useful in applications with sequential data (e.g., text), but also otherwise (vision transformers).
- Examples: GPTs, BERT, ViT, DINOv2

Today’s Program

Part I: Introduction lecture

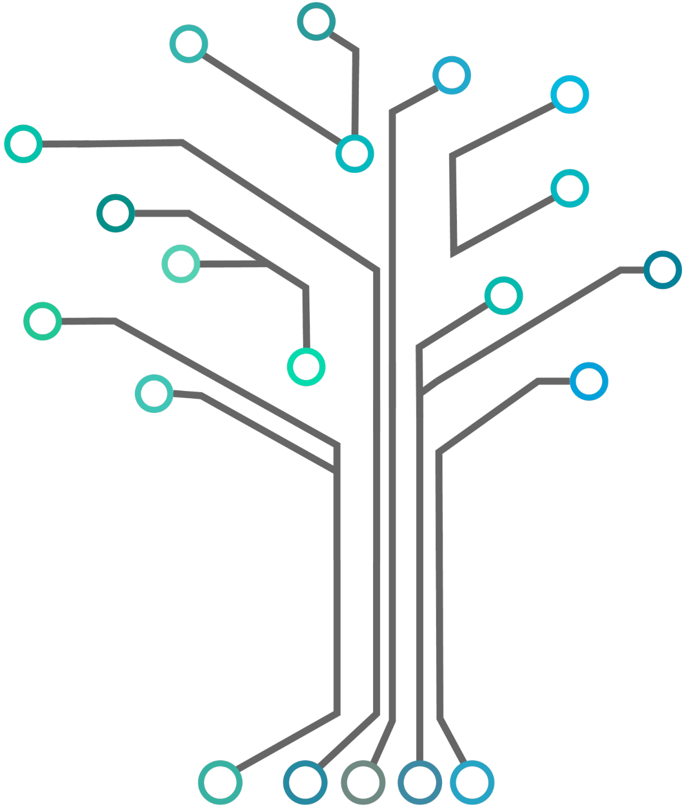
14:15 - 15:45

- Overview
- Theoretical Basics
- Data
- Training
- Evaluation
- Design and Techniques

Part II: Hands-on

16:15 - 18:00

- Questions
- Setup
- Some coding



ÖAW ÖSTERREICHISCHE AKADEMIE DER WISSENSCHAFTEN

AI WinterSchool

January 20 - 24, 2025

Exercises

Exercises

Using Google Colab and PyTorch.

Open the notebook **Intro_WS_2025.ipynb**.

Follow the instructions in the notebook.