Artificial Intelligence Zusammenfassung

Artificial Intelligence

↳ building intelligent machines that think and act rational

Intelligence

↳ The ability to acquire and apply knowledge and skills

↳ The ability to apply knowledge to manipulate one's environment

⇒ Intelligence measures an agents ability to achieve goals in a wide range of environments

Birth of AI

1956 → idea of a calculator that is able to solve intellectual problems as well as or better than humans

Strong vs Weak AI

Strong AI

↳ self concious (sentient)

⇒ Artificial General Intelligence

Weak AI

↳ appears to be Intelligent

⇒ Narrow AI

Turing Test

↳ Human Interrogator communicates with a human or machine over a text-only channel and tries to figure out what it is.

⇒ If the Interrogator thinks a machine is a human, he just gave it a level of Intelligence

Chinese Room Experiment

↳ An individual is placed in a closed room with a set of instructions. He gets an input and has to process it with the given rules and returns an output.

⇒ The individual does not know what he is doing, but from the outside, the individual seams intelligent.

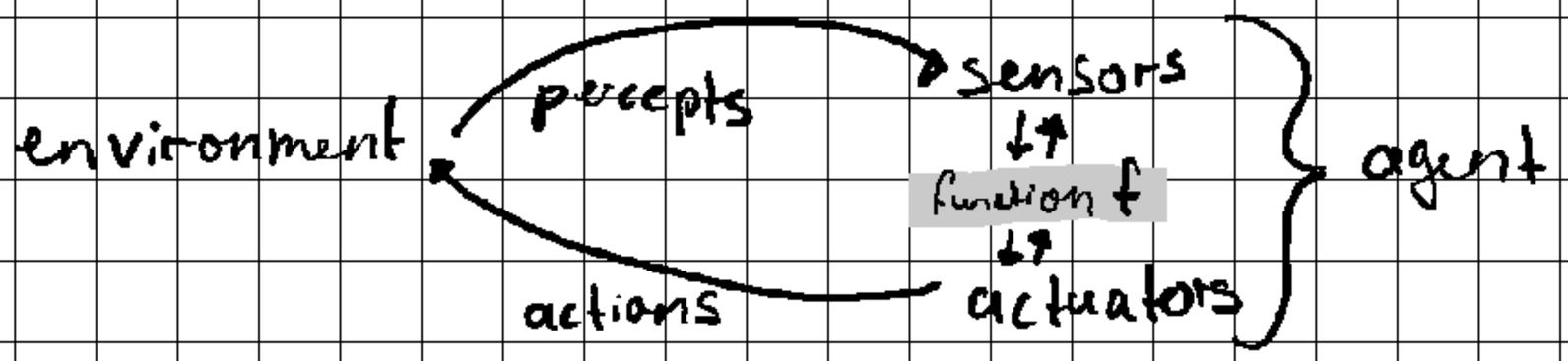
Agents

↳ can be anything that interacts with the environment

→ an agent function f maps from percept histories to actions

$$f : P^* \rightarrow A$$

(Function f = brain/neural network)



Rational Agent

↳ selects an action that is expected to maximize its performance measure given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

⇒ tries to do "the right thing"

Task Description & Environment

↳ Description of the Environment for the deployment of the rational agent

- Task Environment

↳ PEAS

↳ P → Performance Measure

E → Environment

A → Actuators

S → Sensors

- Properties of Environment

- Fully - vs. Partially observable → sensors cover every possible environment state

- Deterministic vs. Stochastic → is the environment always computable

- Episodic vs Sequential → is the agents experience atomic or is it required to solve a task

- Static vs Dynamic → does the environment change while the agent is "working"

- Discrete vs Continuous → is the number of actions that can be taken countable

- Single- vs Multi-Agent → are there any other agents do they compete, cooperate, communicate

Agent Design

↳ How to design an agent to work best for the given environment

→ several degrees of complexity:

Table < Rules < Search < Learning

Table: naive
Rules: adaptive
Search: path finding
Learning: learns with changing conditions

Types of Agents

- Simple Reflex-based → Table
- Model Reflex-based → Rules
- Goal-based → Search
- Utility-based → Search (quantitative comparisons)
- Learning-based → Learning

Problem Representation

↳ before an agent can start searching for a solution, a goal must be identified

Five components of a problem representation

- an initial state
- a set of possible actions
- a transition model describing the results of those actions (graph)
- a goal test function
- a path cost function

⇒ The set of all states reachable from the initial state by any sequence of actions, forms a directed graph

⇒ Search Algorithms are used to find a path (solution) through the graph

↳ they can be evaluated on the basis of completeness, optimal time, complexity, space complexity

Uniform Search

↳ blind / brute force search

→ only able to distinguish goal and non goal state

Algorithms: Bfs & Dfs

Breadth - first Search

↳ traverses a graph using a **queue**

↳ writes neighbor nodes to a queue and and moves to the node that has been in the queue the longest

↳ **First in First out**

⇒ guaranteed to find the shallowest path to the goal

⇛ Time Complexity: Exponential ($O(b^g)$)

Depth - first Search

↳ traverses a graph using a **stack**

↳ pushes neighbor nodes to a stack and moves to the top node

↳ **First in last out**

⇛ Better memory complexity than Bfs ($O(bm)$)

⇛ can't complete if the graph has an infinite depth

⇛ Time Complexity: Exponential ($O(b^m)$)

Informed (Heuristic) Search

↳ uses an evaluation function f(n) for each node to estimate utility

⇒ different Heuristics can be used as evaluation functions

⇒ Node with lowest cost is expanded first

↳ Greedy Search

↳ A* Search

Greedy Search

↳ traverses graph by evaluating the neighboring nodes and moving to the optimal one (ex: shortest distance to goal)

⇒ can get stuck in loops

⇒ Time complexity: Exponential, but improves with a good Heuristic

A* Search

↳ traverses graph by calculating the cost to this point and estimating the additional cost until the goal is reached

Evaluation Function: $f(n) = g(n) + h(n)$

- ↳ $g(n)$ → cost so far
- ↳ $h(n)$ → estimated cost to goal
- ↳ $f(n)$ → estimated total cost

⇒ completes unless there are infinitely many nodes

⇒ Time Complexity: Exponential, but can moves towards Linear, the better the Heuristic

↳ Time Complexity is affected by the relative error of the Heuristic

↳ Relative Error: $\frac{h(n) - h^*(n)}{h^*(n)}$

$h(n)$ → estimate

$h^*(n)$ → real cost

Why do we need machine Learning

↳ newly created data gets bigger and bigger and humans are not able to handle the load

⇛ ML turns data into information

↳ it enables us to query the data

Queries

- query by text → ex: search engine
  ↳ there is alway a difference what we think about and what we are able to write down (semantic gap)
- query by example → ex: image similarity search
- ubiquitous - no explicit queries → ex: camera face detection

Things we can look for

- object → what do we see where in an image
- concept → what is the context, where is the image
- event → what is happening in the image

Entities to recognize

- objects (static → images)
- scenes (static → images)
- activities (dynamic → videos)
- events (dynamic → videos)

⇒ These are classes in ML. Each can be identified using a label

Types of Machine Learning

- **Supervised Learning**
  - ↳ learns relationship between data and a desired output
    - ↳ tries to learn based on labeled content
      - ⇛ Learning known Patterns

- **Unsupervised Learning**
  - ↳ Identification of unknown distributions, patterns and dependencies
    - ↳ labels are unknown
      - ⇛ learning unknown Patterns

- **Semi-Supervised Learning**
  - ↳ learning structures with only a few labels

- **Self-Supervised Learning**
  - ↳ learning representation of data by controlled pseudo-labels

Pseudo labels

↳ the model generates labels and tries to minimize the difference between prediction and actual data

(ex: masking part of an image and try to predict the missing part)

Benchmarking

↳ evaluate the quality of ML results

Criteria

- accuracy → does it work
- Speed → time complexity
- storage requirements → space complexity

Classification performance

↳ compare ML results to a ground truth set (labeled set)

↳ error Rate = $\frac{\#\text{errors}}{n}$

accuracy = 1 - error Rate

∇ when labeling a set, watch out for the interpretation (an image can be labeld differently while maintaining the same meaning, but for the model those would be two diffrent classes)

⇛ with multiple label one can add an exepted error by comparing a number of the top results of the ML with the ground truth.

Retrival Benchmark

X = number of retrieved documents

R = number of relevant documents

→ precision: $p = |X \cap R| \ / \ |X|$

→ recall : $r = |X \cap R| \ / \ |R|$

different Errors

- false positives (FP) → ex: detected face, but its not a face
- true positives (TP) → ex: detects face correctly
- false negative (FN) → ex: doesn't detect a face but it is one
- true negative (TP) → ex: correctly ignores a battle

- error Rate $= \dfrac{FP + FN}{TP + TN + FP + FN}$
- true Positive Rate (TPR) $= \dfrac{TP}{TP + FN}$
- true Negative Rate (TNR) $= \dfrac{TN}{TN + FP}$
- false Positive Rate (FPR) $= \dfrac{FP}{FP + TN}$
- false Negative Rate (FNR) $= \dfrac{FN}{FN + TP}$

Confidence-based Benchmark

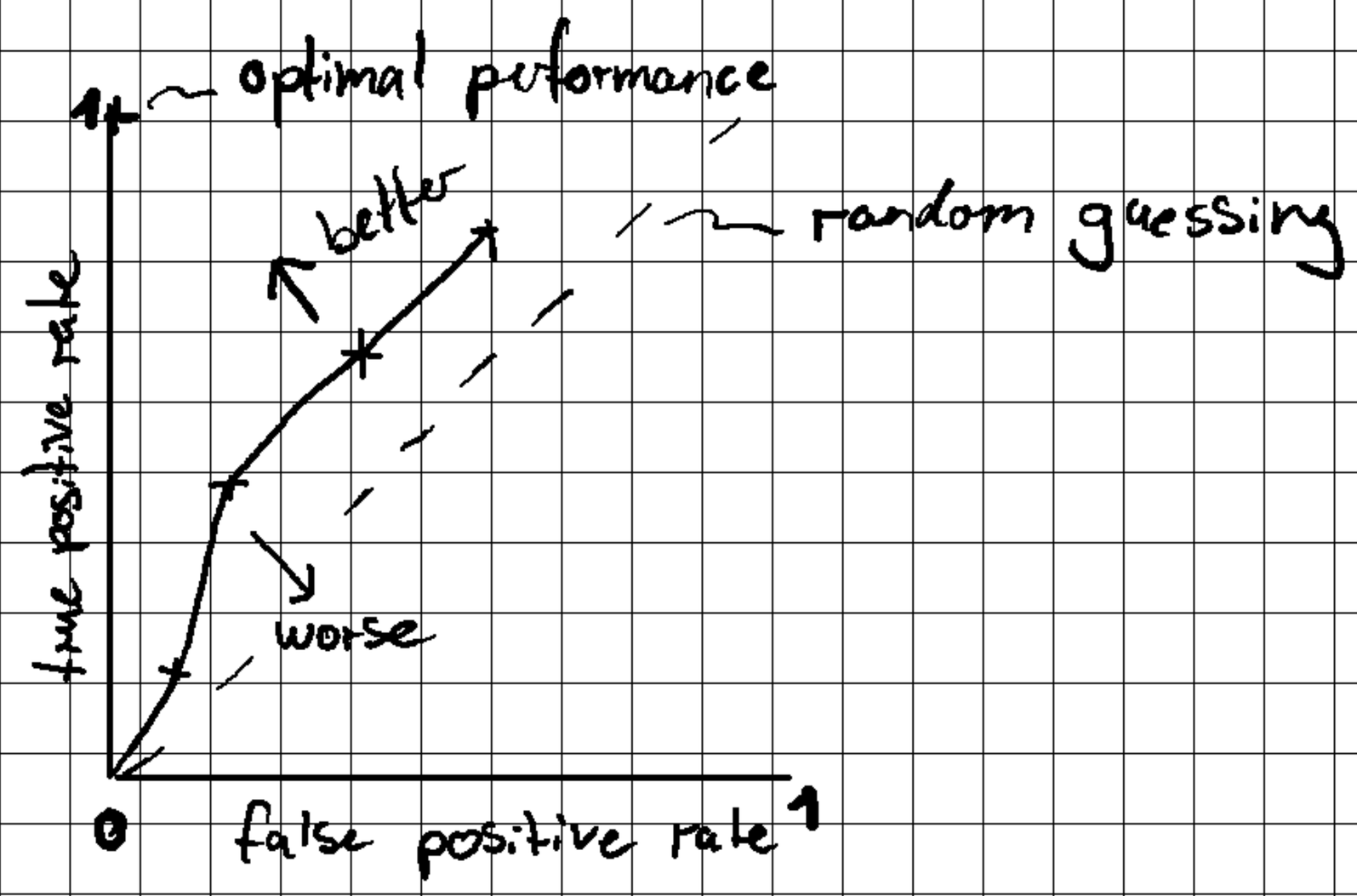
↳ ML model returns a probability of certainty

⇒ enables us to rank the results

⇒ we can define a threshold, when is a result acceptable

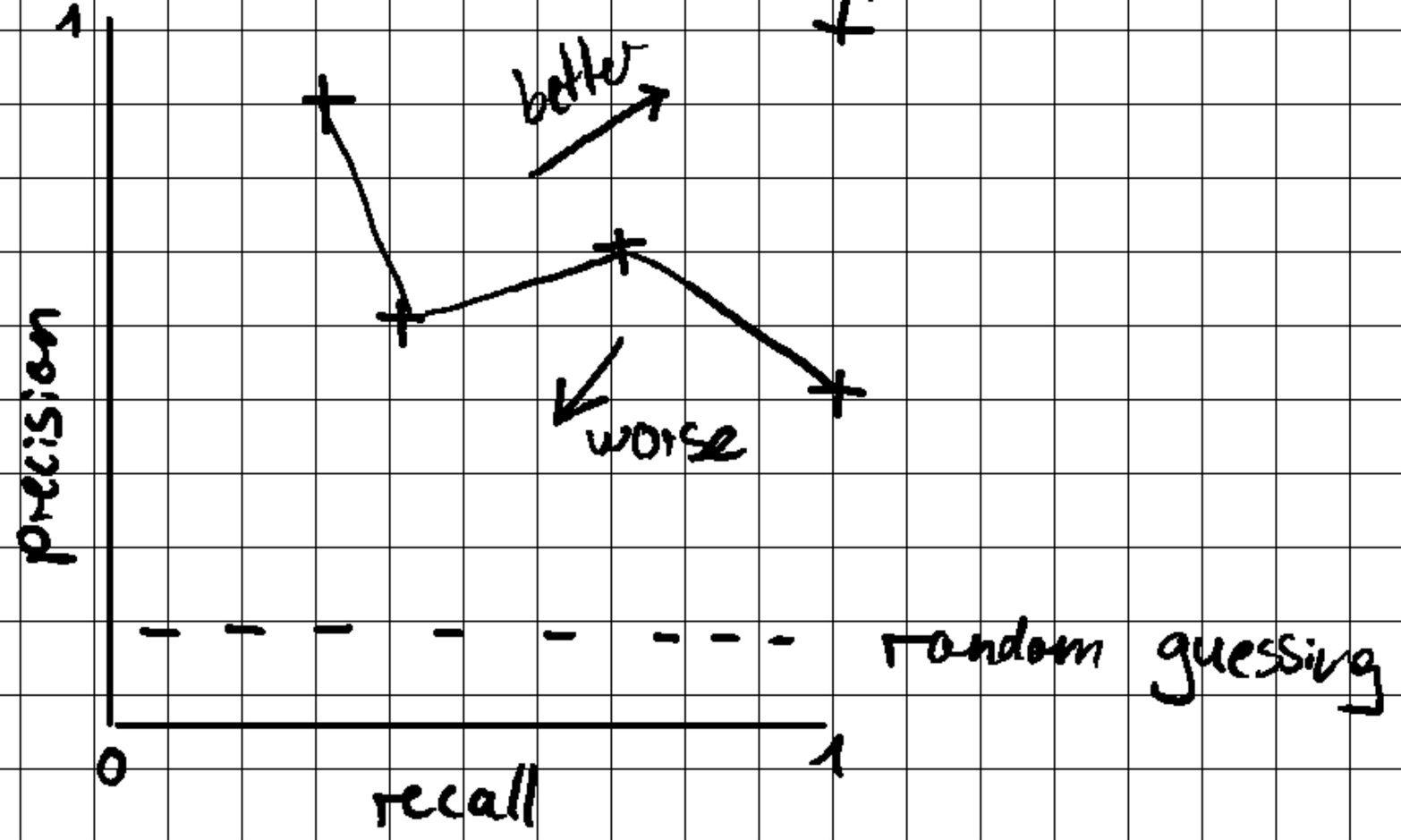
Receiver operating characteristic (ROC) curve

↳ graphical illustration of model performance based on classification threshold



Recall Precision Curve

↳ graphical illustration of the performance based on accuracy of the returned results



↳ can be used to calculate an average precision

Average Precision = area under RP curve

$$AP = \frac{1}{|R|} \cdot \sum \text{prec@r}$$

Precision at rank

rank = number of returned results

Benchmark Entire Systems

Limitations:

- different kind of relevance (topical, user ...)
- semantic gap
- diversity & UX

Supervised Learning

↳ learning with labeled data

Goal → Map objects to pre-defined classes

→ to train one needs seperate test & train data set

Training Data

→ Sample pairs (x_i, y_i)...

x_i: ex. image, y_i: label

↳ try to extract features → mathematical representation

(ex: image to rgb graph)

⇒ Sample x_i is represented by vector $\vec{x_i}$

∇ to many features can cause problems

(curse of high dimensionality)

⇒ we try to find a classificator that correctly identifies the training data by minimizing the error.

We try to find a decision boundry that is non-linear but regularized.

Classifiers

↳ decide to which class a data point belongs

⇒ minimize the probability of error

$$y^* = \arg\max_{y} P(y|x)$$

($P(y|x)$: probability of a feature x belonging to a class y)

y^*: best class

y: all possible classes

⇒ tries to maximize the probability

Types of Classifiers

- Generative Classifiers
  ↳ learn the distribution and tries to generate synthetic obervations
  ex: Naive Bayes

- Discriminative Classifiers
  ↳ doesn't care about distribution, just tries to find a decision boundary (approximation)
  ex: logistic Regression, Support Vector Machine

Naive Bayes

label: y

concept: c

$$\Rightarrow P(c|x)$$

$$P(c|x) = \underbrace{\frac{P(c,x)}{P(x)} = \frac{P(c)\cdot P(x|c)}{P(x)}}_{\text{Bayes Rule}}$$

⇒ Naive Bayes now calculats both probabilities $P(c|x)$ and $P(\neg c|x)$ to classify.

⇒ we can use this to calculate the distribution density with a density formula that matches our data

↳ we use the normal distribution as dictated by the central limit theorem.

$$P(c|x) = \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(x-\mu)^2}{2\sigma^2}$$

Central Limit Theorem

↳ when independent random variables are added, their sum tends toward a normal distribution

Overfitting

↳ ML Model learns data too well

⇛ good training data results, bad new data results

causes:

- few trainig samples
- high dimensions
- many system parameters
- complex models

Nearest Neighbor

↳ assign class to a data point based on the classes of the nearest neighbors in the training set

Set of training samples: $\{x_i, y_i\}$

x_i: feature representation

y_i: label

unknown sample : x

1. calculate Distance $D(x, x_i)$ of x to every training sample x_i
2. select the k closest instances $x_{i1} \dots x_{ik}$ and their class labels
3. classify x according to the majority class of its k neighbors

Calculation:

$$P(y|x) = \frac{1}{k} \sum_{j=1}^{k} \sigma(y_{ij}, y), \quad \sigma = \begin{cases} 1 & \text{if } y_{ij} = y \\ 0 & \text{if } y_{ij} \neq y \end{cases}$$

Distance Measures

- Euclidian Distance
- Manhatten Distance
- Hamming Distance

Euclidian Distance

↳ used in the context of continuous variables

$$D(x, x_i) = \sqrt{\sum_{i=1}^{n} (x - x_i)^2}$$

Manhatten Distance

↳ used in the context of binary or encoded variables

$$D(x, x_i) = \sum_{i=1}^{n} ||x - x_i||$$

Hamming Distance

↳ used in the context of categorical variables

$$D(x, x_i) = \sum_{i=1}^{n} \delta(x, x_i) \quad , \quad \delta = \begin{cases} 0 & \text{if } x_i = x \\ 1 & \text{if } x_i \neq x \end{cases}$$

Logistic Regression

↳ approximate a decision boundry by changing learnable parameters in a function $f_\theta(x)$ until an optimal result is found

⇛ function cannot be linear as a class may have outliers.

⇛ we use a **Sigmoid function** σ

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

⇛ $f_\theta(x) = \sigma(w \cdot x + b)$

$$= \frac{1}{1+e^{-(wx+b)}}$$

$w \rightarrow$ weight vector

$b \rightarrow$ bias

⇛ w has to be maximized

$$w^* = \arg\max_w \prod_{i:c_i=1} \sigma(w \cdot x_i) \cdot \prod_{i:c_i=0} (1 - \sigma(w \cdot x_i))$$

Training Algorithm

1. initialize weight w^0
2. until Loss function L converges
   ↳ update weight according to Gradient Descent learning
   ↳ increase $k = k+1$

Gradient Descent Learning

$$w^{k+1} = w^k - \lambda \nabla \mathcal{L}_n$$

$$\nabla \mathcal{L}_n = \sum_{i=1}^{n} \left(c_i - \sigma(w^k x_i) \right) x_i$$

$\Rightarrow$ goal is to minimize error $\mathcal{L}$

Support Vector Machines (SVMs)

↳ classify data by finding the best boundary (hyperplane) that seperates data points of different classes

⇒ extract features from all data points

↳ data point is now represented by multiple features

↳ place all datapoints into an x-dimensional plane based on its feature representation

↳ find hyperplane that best approximates an optimal boundary

Training samples : $x_1 \dots x_n \in R^d$

labels : $y_1 \dots y_n \in \{-1, 1\}$

↳ label belongs/doesn't belong to class

find Hyperplane w that seperates classes

$$f(x) = \underbrace{\langle w, x \rangle}_{\text{dot product}} + b$$

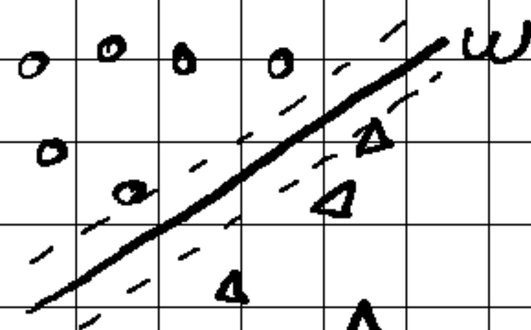
$$f(x) = \langle w, x \rangle \quad (x \to [x, 1])$$

↳ is part of the class if $x > 0$

⇒ finding hyperplane, using maximum margin principle

Margin Maximization Principle

↳ find the Hyperplane with as much margin between data points and Hyperplane, to keep room for future samples to make the classification more robust



⇒ depending on the label the data point is on one side plane w

$$\Rightarrow \quad y_i \cdot \langle w, x_i \rangle \geq 1 \quad \text{for all } i$$

Types of samples

- Safe samples → far away from boundary → $\langle w, x_i \rangle > y_i$
- Support vectors → samples on the margin → $\langle w, x_i \rangle = y_i$

⇒ relationship between y and w

- size of margin y is $\frac{1}{\|w\|^2}$
- maximizing the margin is equivalent to minimizing $\|w\|^2$

⇒ $w^* = \arg\min_{w \in \mathbb{R}^d} \|w\|^2$

⇒ SVM gives us a globally optimal solution

Problems when using SVMs

- Non Seperability
  ↳ data set is not linear separable

  Solutions

- Slack Variables → allow for some etor

$$w^* = \min_{w, \xi_1 \dots \xi_n \in \mathbb{R}} \|w\|^2 + C \underbrace{\sum_i \xi_i}$$

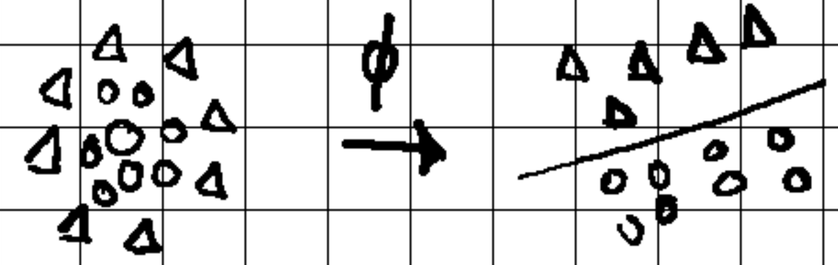
Sum of all slack variables

C → Hyper-parameter

↳ the smaller C, the larger the margin but with more incorrectly qualified samples

- Kernel function → transform the data to make it seperable

→ find transformation ϕ



⇒ we perform classification on $\phi(x_i)$ instead of x_i.

finding ϕ can be difficult. Instead we will use similarity functions $k(x_i, x_j)$ that compare two samples x_i, x_j → this approach is called

kernel trick

Kernel Trick

$$w = \sum_{j=1}^{n} \alpha_j \cdot \phi(x_j)$$

α_j = weight of the sample

↳ "heavier" towards the boundary → support vectors

↳ learnable parameter

In the SVM Equation

$$w^* = \min_{\alpha_i, \xi_i \in \mathbb{R}^n} \sum_{i,j=1}^{n} \alpha_i \alpha_j \cdot \underbrace{k(x_i, x_j)}_{\text{kernel function}} + C \sum_i \xi_i$$

Combination of distinct classifiers

- early fusion
  - ↳ concatenate features
    - ↳ take all features and put them through a single classifier to get to a decision
- late fusion
  - ↳ combine classification results
    - ↳ take each feature and put it through a specified classifier. Afterwards take those results and combine them to get to a decision

Unsupervised Learning

↳ unknown Labels → learning based on reconstruction

Clustering

↳ partition Dataset into coherent parts (clusters)

Types of clustering

- centroid clustering
  ↳ top down → start with the whole set and divide
    ⇛ Typical Approaches
      ↳ k-Means
      ↳ Expectation Maximization

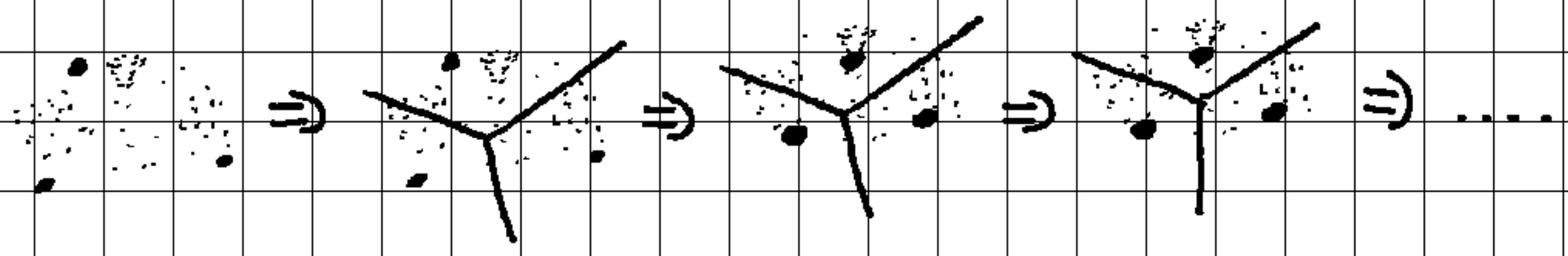
- Hierarchical Clustering
  ↳ bottom up → start with single sample clusters and merge
    ⇛ Typical Approaches
      ↳ Agglomerative Clustering

k-Means Clustering

- Define k Means at random points
- calculate the distance of each datapoint to each mean
- calculate borders based on the nearest mean of each datapoint
- recalculate means based on all "member" datapoints
- recalculate distances and new bordes
- repeat until means no longer move

Distance: $k(i) = \underset{k}{\operatorname{argmin}} \|x_i - \mu_k\|^2$ (ex: $\|\overbrace{(5,2)}^{\text{point}} - \overbrace{(3,1)}^{\text{centroid}}\|^2 = \|(2,1)\|^2 = 2^2+1^2$)

new Mean: $\mu_k := \{x_i : k(i) = k\}$ (points: $\mu_1 \{(2,3),(5,4)\}$, Mean: $\left(\frac{2+5}{2}, \frac{3+4}{2}\right) = \left(\frac{7}{2}, \frac{7}{2}\right)$)



⇒ ends when no member switches cluster

⇒ restart when cluster is empty

⇒ choose k (number of means)

↳ high k → increases model complexity

↳ low k → increases fitting error

Optimal k:

increases fitting error | increases complexity

$$k^* = \underset{k}{\operatorname{arg\,min}} \; -2 \ln P(x_1 \dots x_n \mid k) + f(k)\frac{n}{\log n}$$

Expectation Maximization

↳ more general scheme than k-means

Goal:

find Parameter Θ that maximizes the likelihood of the observed data by adjusting μ and Σ so that the probability of the observed data being generated by the model is as high as possible

$\mu \rightarrow$ the means of the cluster

$\Sigma \rightarrow$ variances & covariances of the cluster

$$\Theta^* = \arg\max_{\Theta} P(D|\Theta) = \arg\max_{\Theta} \sum_{U} P(\Theta, U|D)$$

$D \rightarrow$ samples

$U \rightarrow$ cluster memberships of the data points

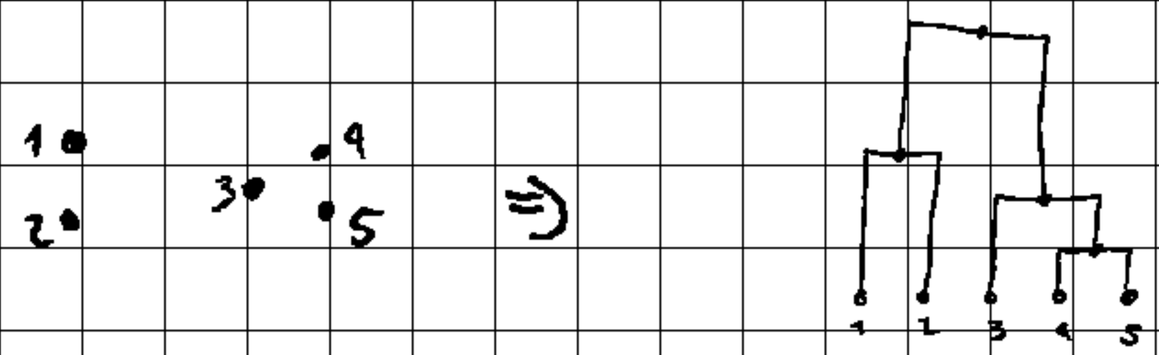
Agglomerative Clustering

↳ Hierarchical Clustering that starts with singelton clusters

- start with singelton clusters
- pick two clusters with minimal distance
- fuse the two clusters into one

⇒ results can be displayed as a Dendogram

↳ Hight shows distance (time until merge



Linkage Strategies

- Single Linkage: $dist(X,Y) = \min_{x \in X, y \in Y} \|x-y\|_2$
- Complete Linkage: $dist(X,Y) = \max_{x \in X, y \in Y} \|x-y\|_2$
- Average Linkage: $dist(X,Y) = \frac{1}{|X| \cdot |Y|} \sum_{x \in X, y \in Y} \|x-y\|_2$

$$\left(\underbrace{\|x-y\|_2}_{\text{Euclidian Distance}} = \sqrt{(x_1-y_1)^2 + (x_2-y_2)^2 + \dots + (x_n-y_n)^2} \right)$$

Dimensionality Reduction

↳ project features to lower dimensional space, while most variation is preserved

Types:

- Principle Component Analysis (PCA)
- Topic Models
- Fisher's LDA
- ⋮

Principle Component Analysis

↳ compute k principle components

↳ those are components that capture the most variance

given sample $x_1 \dots x_n$

1. shift the data to mean 0

$$x_i = x_i - \frac{1}{n}\sum_i x_i$$

2. compute the covariance Matrix

$$\Sigma = \frac{1}{n}\sum_i x_i^T \cdot x_i$$

3. compute the eigenvalues of the covariance Matrix

solve for λ: $\det(A - \lambda I) = 0$

(A: covariance Matrix; I: Identity Matrix)

4. compute the eigenvectors

solve for $\vec{v}$: $(A - \lambda I) \cdot \vec{v} = 0$

and normalize → $\frac{\vec{v}}{\|\vec{v}\|}$ $\left(\|\vec{v}\| = \sqrt{x^2+y^2} \text{ for } \vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}\right)$

5. the first k eigenvectors of Σ correspont to the k principal components $w_1 \dots w_k$

6. project the data

$$x_i^{PCA} = \begin{pmatrix} w_1 \\ \vdots \\ w_k \end{pmatrix} \cdot x_i^T$$

Neural Networks

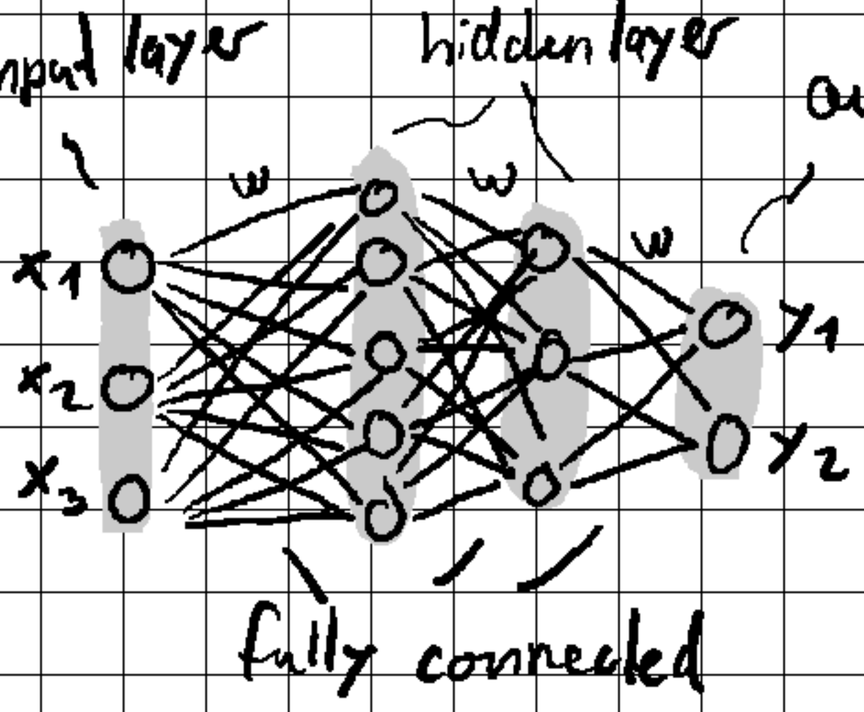
→ a cascade of multiple layers of non-linear processing units for feature extraction and transformation.
Each successive layer uses the output from the previous layer as input.

Types of Architectures

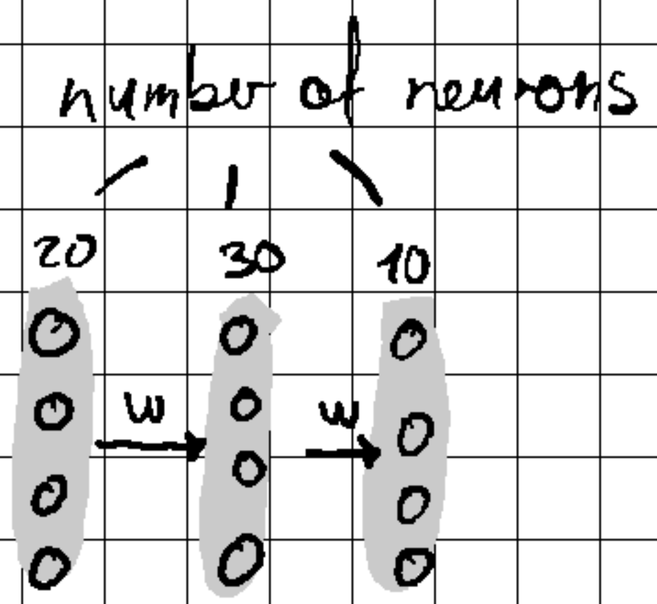
- Multilayer Perceptrons (MLP)
- Recurrent Neural Networks (RNN)
- Deep Autoencoder
- Convolutional Neural Network (CNN)
- Generative Adversarial Networks (GAN)
- Attention & Transformer

⇒ A Neural Network is a weighted, directed graph of connected neurons

⇒ we organize them in layers:



simplification ⇒



Feedforeward Neural Network (FNN)

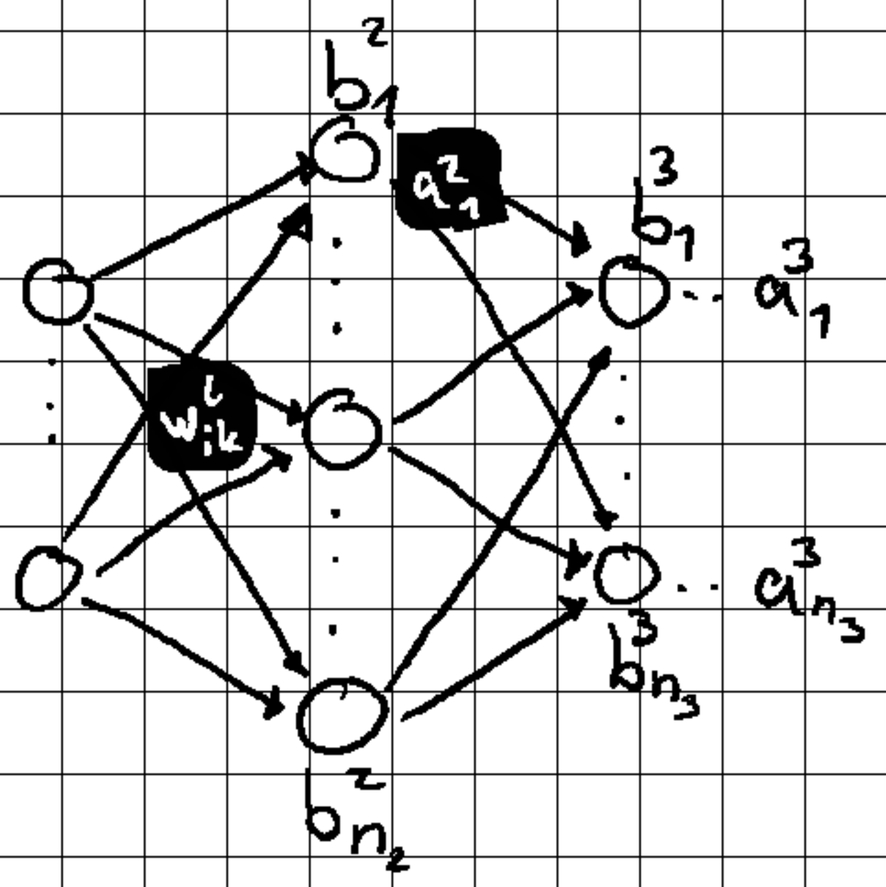
↳ free from cycles → every neuron is connected only with neurons from the previous or from the next layer

Exception:

→ skip connections or residual connections

⇒ two layers are called fully connected, if there exist a pairwise connection between all neurons

Graph Notation



a_i^L → activation/output of neuron i in layer L

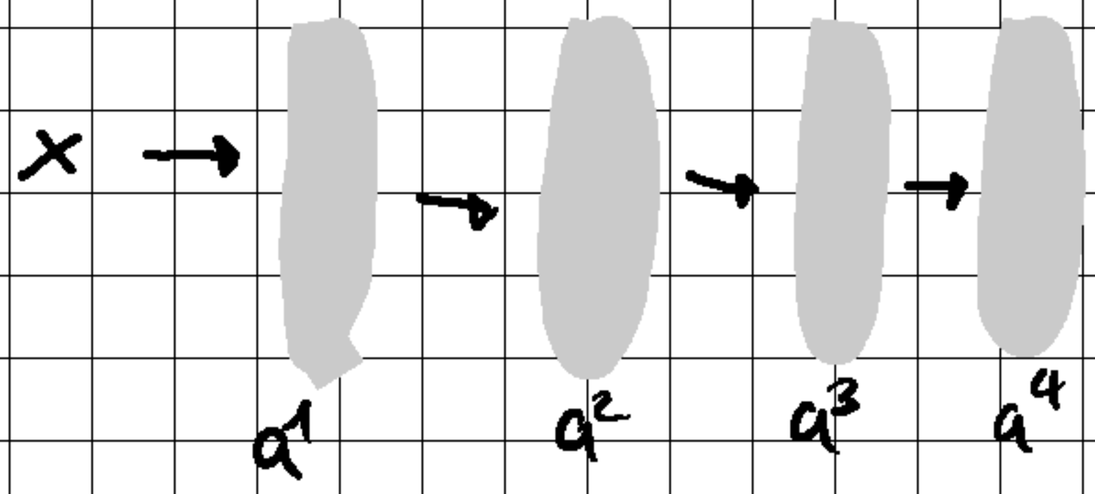
b_i^L → bias of neuron i in layer L

w_{ik}^L → weight of the connection between neuron i in layer $(L-1)$ and neuron k in layer L

Signal transformation

↳ Neural Networks are nested Functions

→ Signal follows the path from the "inside" to the "outside"



$$f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(x))))$$

$$a^4 := \sigma(w^4\,\sigma(w^3\,\sigma(w^2\,a^1 + b^2) + b^3) + b^4)$$

W → weight

a → activation

b → bias

End-to-End Learning

input x → function $f(x)$ → output target y

function $f(x)$

↳ cascade of non-linear transformations

↳ training of features & classification

Artificial Neuron

↳ mimics the behaviour of a biological neuron, where it receives inputs, processes them and produces an output

Limitation:

→ can only solve problems that are linearly seperable.

↳ It cannot handle more complex cases like the XOR problem

Rosenblatt Perceptron

↳ linear classifier that tries to separate data points into two classes using a linear boundary

→ introduces weights

Algorithm

1. initiate weights randomly
2. take a sample x_i and predict $\hat{y}_i$
3. for error predictions update w
   - ↳ increase weights if $\hat{y} = 0$ and $y = 1$
   - ↳ decrease weights if $\hat{y} = 1$ and $y = 0$

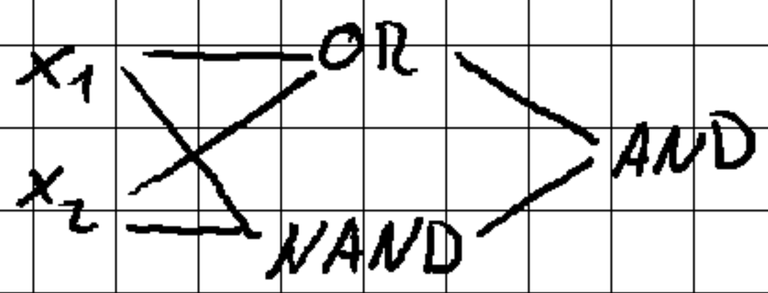
XOR Problem

↳ non-linearly separable data

⇒ we need a way to get an XOR output without an XOR gate

Solution → connecting multiple neurons

⇒



Universal Approximation Theorem

↳ By connecting multiple neurons with sigmoid activations, we can approximate any continuous function with infinitesimal error

Challenges:

- How many neurons do we need?
- How do we find the model weights?

Learning Setup

- Data → samples with or without labels $\{x_i, y_i\}^N$
- Model → Function mapping from inputs to outputs
  ↳ $\hat{y} = f_\theta(x_i)$
- Objective / Loss → Defines an optimum quantified by how well the model performs
  ↳ $\theta^* = \min_\theta J(\theta) = \sum \mathcal{L}(\hat{y}_i, y_i)$
- Estimator / Optimizer → Algorithm to adjust parameters iteratively to optimize the objective function
  ↳ $\theta^{(t+1)} = \theta^t - \eta \nabla J(\theta)$

⇛ The data stays constant, while the others change based on the approach (ex: Naive Bayes, SVM, etc...)

Linear Regression vs Logistic Regression

Linear Regression → used for predicting continuous values by fitting a linear relationship between inputs & outputs

Logistic Regression → used for binary classification tasks by modeling the probability that a given input belongs to a class

Training Neural Networks

→ The optimization Problem belongs to the NP complete problems

↳ This makes it impossible to find an optimal solution efficiently in polynomial time

⇒ to proof this, we can reduce our problem to the set splitting problem, which is proven to be NP-hard

Set splitting problem

↳ Given a set and a collection of subsets, determine if the set can be partitioned such that every subset has elements in both partitions.

Supervised Learning

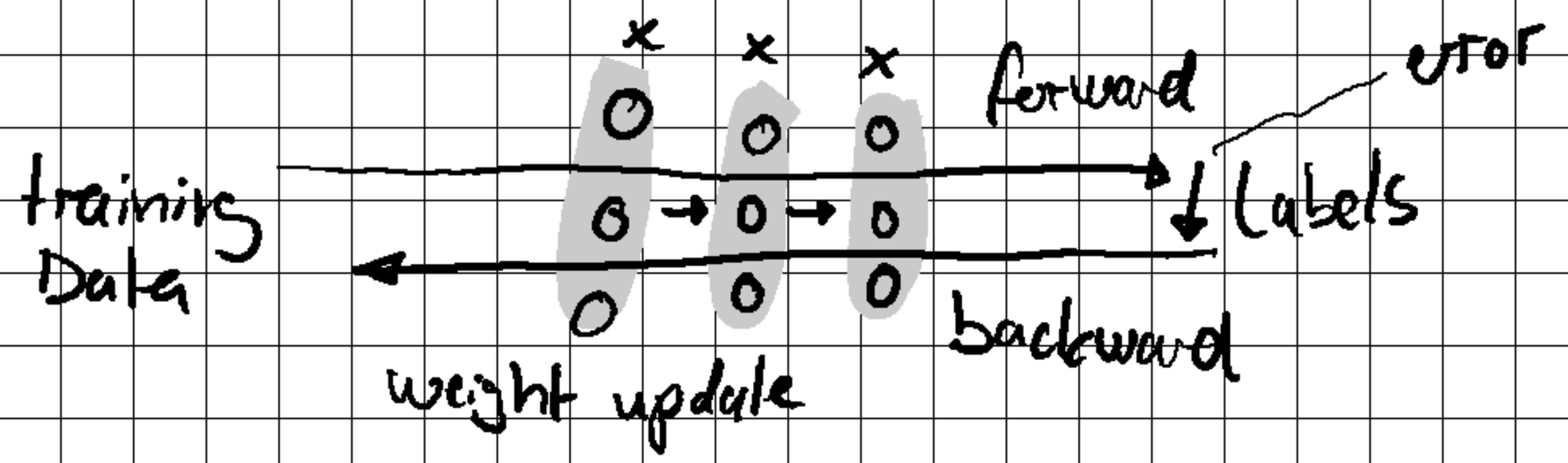
Training

- given labeld training data $\{x_i, y_i\}^N$
- goal: find optimal model parameter θ^*

Algorithm

- initialize θ
- generate $\hat{y}$ in forward pass
- calculate objective $J(\theta)$
- update θ in backward pass

} repeat until convergence



Single layer Training - Regression

- Sigmoid activation function $\sigma(z)$
- prediction $\hat{y}$ is the activation a which is the sigmoid function applied to z

$$\hat{y} = a = \sigma(z)$$

- the input x is linearly combined using weight w and bias b to compute z

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

- now the goal is to get $\hat{y} \approx y$

$$\Rightarrow \quad \sigma(w^T x_i + b) \approx y_i$$

Loss function

↳ measures error between predicted value $\hat{y}$ and actual value y

ex: Mean Squared Error:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y}_i - y_i)^2$$

Objective (Cost) Function

↳ average loss over all training data

$$J(w,b) := \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2}(\hat{y}_i - y_i)$$

Single Layer Training - Classification

↳ sigmoid activation function changes, as we want a binary classification

⇒ 1 → class presence

0 → no class presence

Loss Function

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

Objective (cost) Function

$$J(w,b) = -\frac{1}{N} \sum_{i=1}^{N} \left(y \log \hat{y} + (1-y) \log (1-\hat{y}) \right)$$

Multi Class

↳ what about the presence of multiple classes

Loss Function

$$\mathcal{L}(\hat{y}, y) = -\sum_{m=1}^{M} y_m \log \hat{y}_m$$

Objective (Cost) Function

$$J(w,b) = -\sum_{i=1}^{N} \sum_{m=1}^{M} y_{i,m} \log \hat{y}_{i,m}$$

Stochastic Gradient Descent (SGD)

1. initializ w & b randomly
2. generate prediction forward pass
3. calculate loss
4. update weight in bakward pass

} repeat until convegence

lets say we have the loss function $\mathcal{L} = \frac{1}{2}(y-\hat{y})^2$

and we know $\hat{y} = wx+b$

$$\Rightarrow \mathcal{L} = \frac{1}{2}(y-(wx+b))^2$$

$\Rightarrow$ The gradients of the loss function with respect to w and b are as follows:

$$\frac{\partial \mathcal{L}}{\partial w} = \left[\frac{1}{2}(y-(wx+b))^2\right]' = x(\hat{y}-y)$$

$$\frac{\partial \mathcal{L}}{\partial b} = \left[\frac{1}{2}(y-(wx+b))^2\right]' = \hat{y}-y$$

$\Rightarrow$ with this we can calculate the new a and b using a learning rate α

$$w := w - \alpha(x(\hat{y}-y))$$

$$b := b - \alpha(\hat{y}-y)$$

Backpropagation

↳ update w and b layerwise

1. Forward pass → output predicted $\hat{y}$
2. Compute Loss → some loss function (ex $\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$
3. Backward pass
   1. calculate the derivative of the loss function with respect to the predicted $\hat{y}$
   2. propagate the gradient backwards
      ↳ compute gradient of the loss with respect to the weights and biases in the output layer
      ↳ use chain rule:

$$\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial \mathcal{L}}{\partial a^l} \cdot \frac{\partial a^l}{\partial z^l} \cdot \frac{\partial z^l}{\partial w^l}$$

w^l → weights in layer l

a^l → activations in layer l

$z^l = w^l a^l + b^l$ → weighted sum of inputs for layer l

$\frac{\partial a^l}{\partial z^l}$ = derivative of the activation function

3. update weights and biases

$$w^l := w^l - \alpha \frac{\partial \mathcal{L}}{\partial w^l}$$
$$b^l := b^l - \alpha \frac{\partial \mathcal{L}}{\partial b^l}$$

(α → learning rate)

Vanishing Gradient Problem

↳ In deep networks (many layers) gradients become very small as they "travel" back during backpropagation, making early layers learn very slowly.
This is bad, as the early layers are critical for capturing fundamental features.

Perceptive Field

↳ reduce connections in a fully connected graph

↳ perceptive Field = mask over output of a layer that is able to detect a match
(ex: a face-mask → fires on detected face)

→ saves steps through elimination of redundant calculations

Convolution

↳ Data compression

→ slide "window" over signal and calculate the folding of the data

(⇒ mathematically they are just weighted sums)

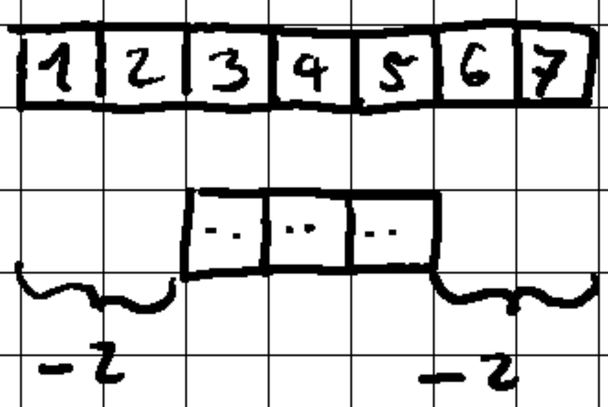
Convolution Layer

Example:

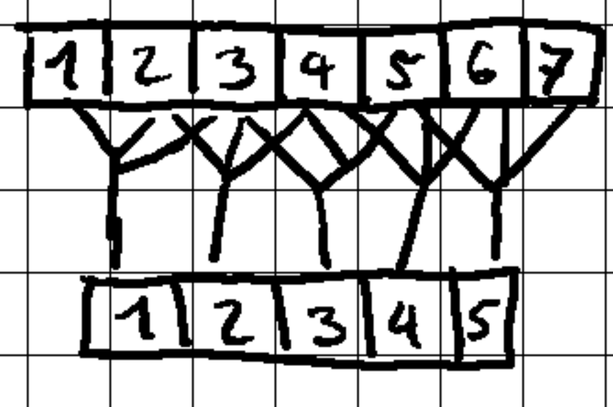
Image → RGB Layer = $32 \cdot 32 \cdot 3$

Filter / Kernel = $5 \cdot 5 \cdot 3$

Dot Product: $32 \cdot 32 \cdot 3 \times 5 \cdot 5 \cdot 3$ = $28 \cdot 28$ (compressed Image)

⇒ Image: 

kernel:

⇒ 

Dot Product of set of 3 with size 3 kernel

⇒ Nr of kernel Layers define output Layers:

Input 3 Layer } 3 output Layer
kernel 1 Layer }

Input 3 Layer } 1 output Layer
kernel 3 Layer }

Pooling

↳ Data reduction through omission

Max Pooling

↳ take only highest value of filters

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→

6	8
3	4

stride → how much the filter moves

5

max pool with 2x2 filter, stride 2

Convolutional Neural Networks (CNN)

↳ Models designed for processing grid-like data, particularly images

⇛ They use a combination of Convolution-, Pooling- and fully connected Layers.

- Input Layer → takes data (ex: image)
- Convolution Layer → applies kernels to produce feature maps
- Activation Layer → applies ReLU functions to add non-linearity
  - ↳ Rectified Linear Unit → for learning more complex patterns
- Pooling Layer → reduce size, while keeping most important infos
- Fully connected Layer → combine features & make prediction

Data Augmentation

↳ Augment training data by sub-sampling

ex: Image → crop centr, corners, flip image, rotate image

⇒ adds more variation to training data

↳ increases amount of training data

Modelling Sequences

↳ use context & memory to help predict the next value

↳ ex: Text → the next word depends on the previous

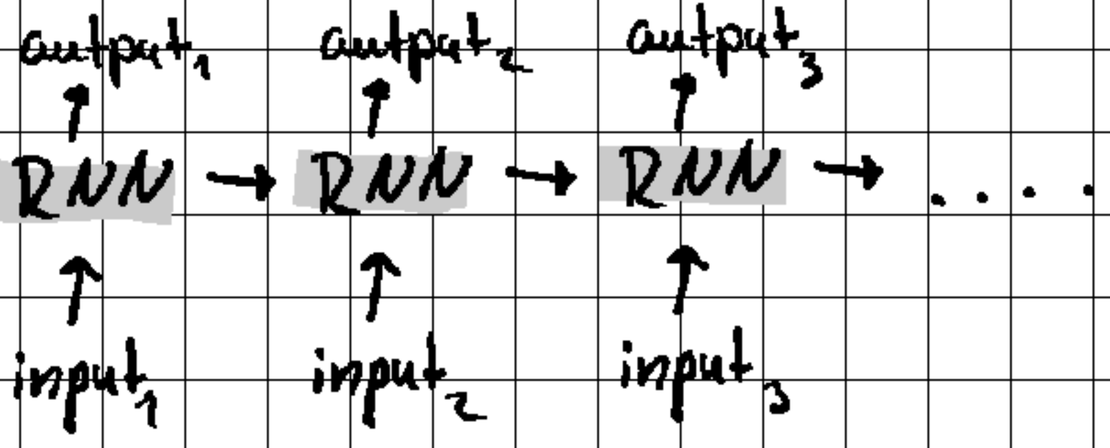
Video → the next frame depends on the previous

Weather → ...

⋮

We can Model this with **Recurrent Neural Network (RNN)**

Recurrent Neural Network (RNN)



⇒ each copy passes some information to its successor

= hidden state

update hidden state:

$$h_t = f_\theta(h_{t-1}, x_t)$$

RNN function

input vector at time t

generate output

$$y_t = g_\phi(h_t)$$

Comparison to "Standard" Neural Network

→ RNN has shared weights, parameters are shared

→ Multi-Layer Neural Network has sperate parameter for each Layer

⇒ we can use output layer of CNN to build our hidden state in a RNN to generate e.g. image captions

RNN Example

Sentiment Analysis

↳ detect if a message is positive or negative

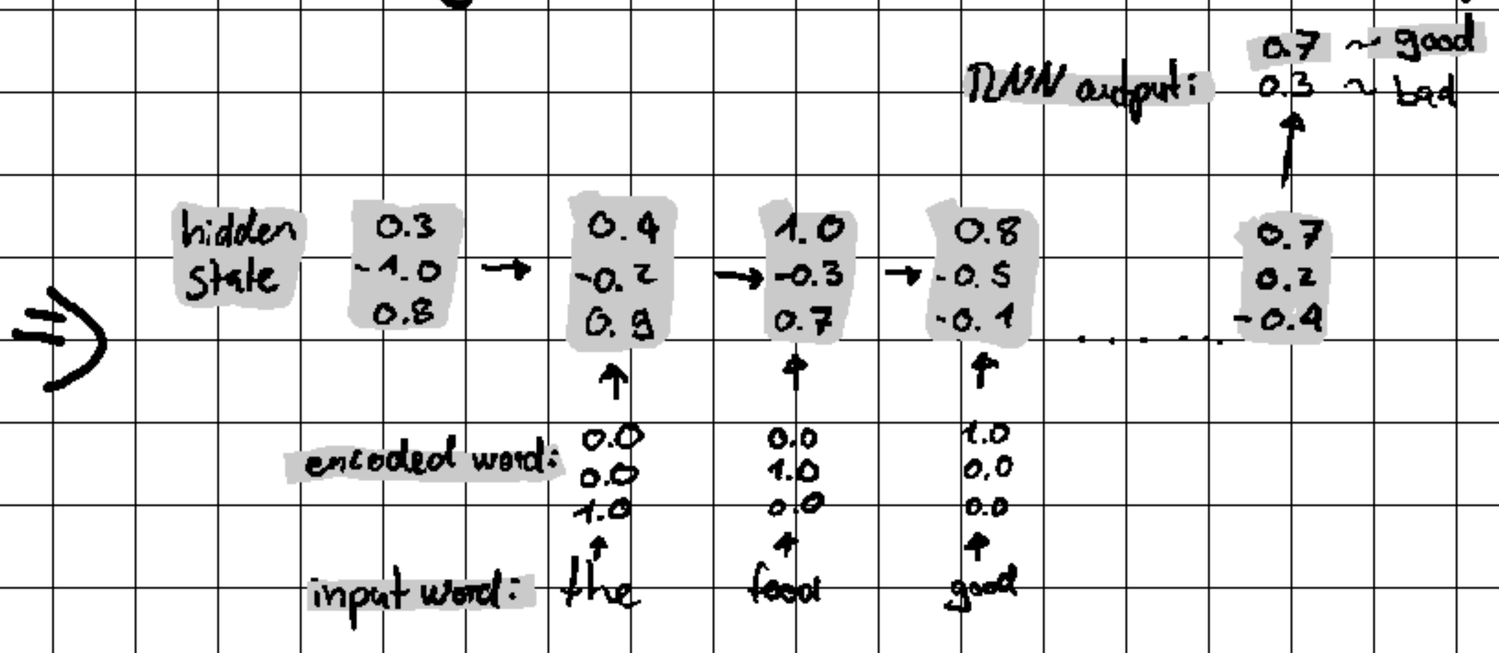
universe of words : good, food, the, ⋮

encode to matrix where each word is a component

good → 0.0
food → 0.0
the → 0.0
⋮ → ⋮

⇉ if word is used the component is set to 1

1. we start with a hidden state
2. for each word we use a function to calculate a new hiddenstate using the encoded word and the old hidden state
3. at the end we calculate an output from the hidden state and get a encoded sentiment, the higher value is more likely



Backpropagation through time (BTT)

↳ Backpropagation of RNNs

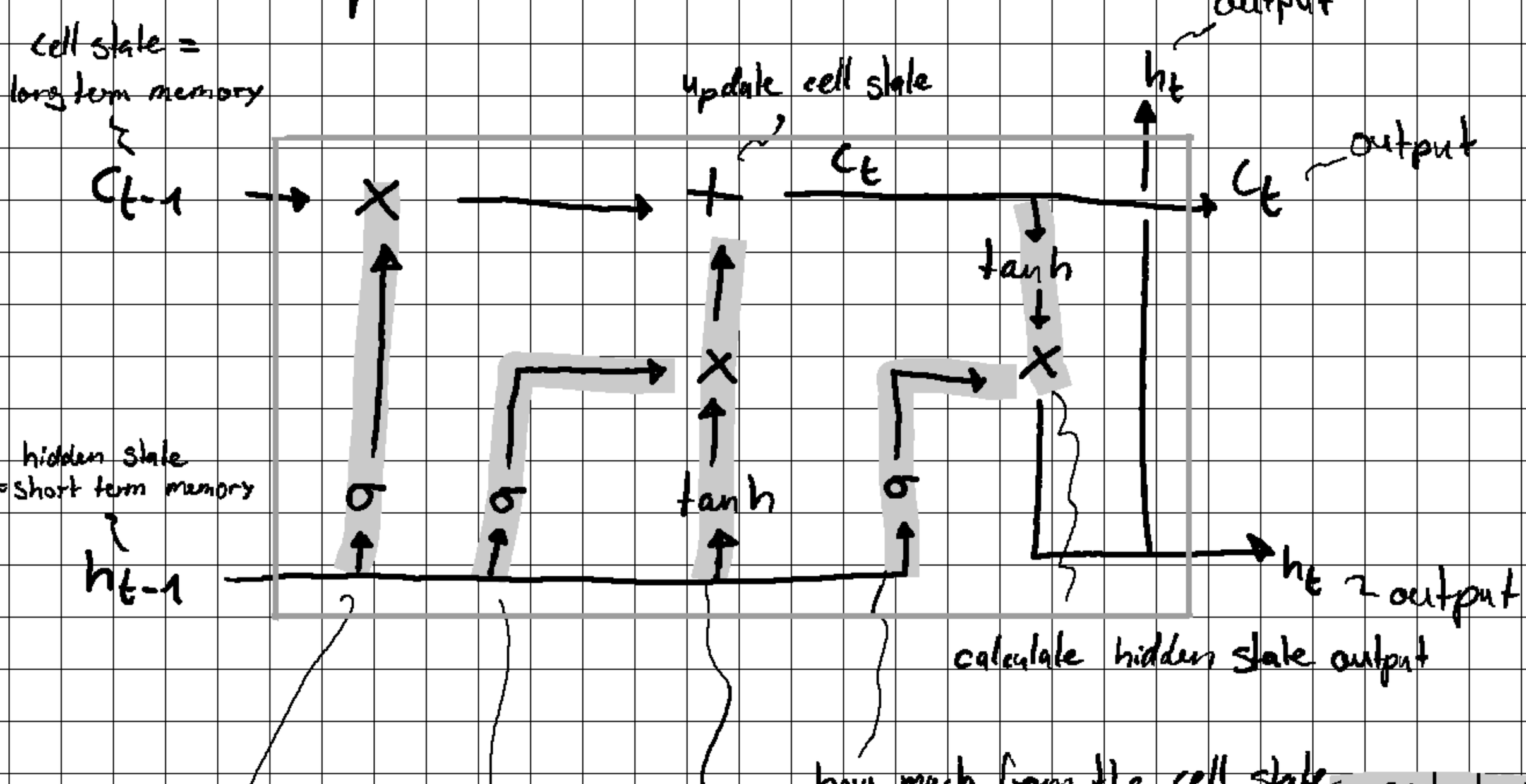
- if they are Many-to-One, then the loss function is calculated at the end
- if they are Many-to-Many then each step has its own error. The whole error is simply the sum

⇒ on Backpropagation, the same weight matrix gets updated for each step.

⇒ this can lead to exploding Gradients or Vanishing Gradients.

Long - Short Term Memory (LSTM)

↳ each cell stores and forgets information based on learnable parameters



Trust in AI

classification problems

↳ can we trust an AI to make a correct decision?

↳ ex: overlap an image with the noise of a second image

↳ humans see image 1, machines image 2

How can we trust something? is it real or generated?

↳ ex: video of a politician

↳ can we some how add a watermark?

Regulations / Principles proposed by Pentagon

→ Responsible → human guided development

→ Equitable → no harm throug unintentional bias

→ Traceable → algorithms should be transparent

→ Reliable → make sure, model doesn't oversteb out of scope

→ Governable → humans should be able to step in and control

Copyright

↳ change one parameter of the model, the whole model changes → no way of knowing if stolen or self-created

Backdoor

↳ there is no way of knowing if a built in backdoor exists.