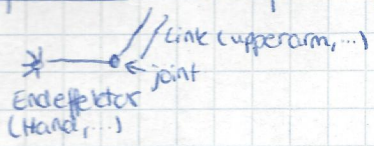
Mathematische Grundlagen MG

Kinematics = analyzes robot/machine motion based on geometry information
essential concept: position

statics = forces and moments applied on the mechanism at rest
ess. conc.: stiffness

dynamics = analyzes forces and moments resulting from motion and acceleration



Kinematic chain = set of links connected by joints
↳ as graph: joints = vertices, links = edges

Degrees of Freedom = number of independent parameters needed to specify the position of a mechanism/object compeletely

SO(3) = represents rotations 3×3 matrices (real), $R^TR=1$, $\det(R)=1$

SE(3) = represents rigid body motions (p,R) $p\in\mathbb{R}^3$, $R\in SO(3)$

Endomorphism = linear transformations that map Euclidean space onto itself
$\Phi : \mathbb{R}^3 \to \mathbb{R}^3$
$\Phi(a) = A\cdot a$ $A\in\mathbb{R}^{3\times3}$

change of basis: $A = (e_x', e_y', e_z')\,(e_x, e_y, e_z)^{-1}$ (new, old)

Isomorphism = bijective endomorphism
(1) preserve angles, lengths, handedness => rotation group

Rotations

$$R_{x,\alpha} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \qquad R_{y,\alpha} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

$$R_{z,\alpha} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$R_{x,\theta}^{-1} = R_{x,-\theta} = R_{x,\theta}^T$

concatenation $\Phi_{z,\theta_3}(\Phi_{y,\theta_2}(\Phi_{x,\theta_1}(a))) = R_{z,\theta_3}\cdot R_{y,\theta_2}\cdot R_{x,\theta_1}\cdot a$

right-to-left: rotations around local axes
l-to-r: rotations around global axes

Euler angles

rotation = three rotations around three coordinate axes

$\rightsquigarrow R_{z,\alpha} \cdot R_{x',\beta} \cdot R_{z'',\gamma}$ z, x', z'' convention

Roll, Pitch, Yaw convention z, y', x''

⊕ more compact, descriptive than matrices
⊖ not unique gimbal lock
not continuous

Affine transformations

homogeneous coordinates $a = (a_x, a_y, a_z, h)^T$ $h \in \{0, 1\}$

$b = Ax + t$

Denavit-Hartenberg

Θ_i = x um z
d_i = x entlang z
a_i = z entlang x
α_i = z um x

Quaternionen

$q = (a, b, c, d) = a + ib + jc + kd$

$i^2 = j^2 = k^2 = ijk = -1$

$ij = k$ $ji = -k$
$jk = i$ $kj = -i$
$ik = j$ $ki = -j$

! NOT commutative

$\langle q, r \rangle = ab + \langle v, u \rangle = ab + v_1 u_1 + v_2 u_2 + v_3 u_3$

$q^* = (a, -u)$ $|q| = \sqrt{qq^*} = \sqrt{a^2 + u_1^2 + u_2^2 + u_3^2}$ $q^{-1} = \frac{q^*}{|q|^2}$

Rotation axis a, angle Φ $q = (\cos\frac{\Phi}{2}, a \sin\frac{\Phi}{2})$

rotation v by q: $v' = qvq^{-1} = qvq^*$

⊕ compact, no gimbal lock, continuous repr.
⊖ no translations

Algorithmic Fundamentals AF

Task space and configuration space ARMAR

- hand moves in cartesian space → workspace
- robot motions require motions of the joints → configuration space

↝ related by forward and inverse kinematics

Forward kinematics
determine pose of the end-effector from the joint angles

Inverse kinematics
determine the joint angles for a given end-effector pose

→ Jacobi-Matrix is link

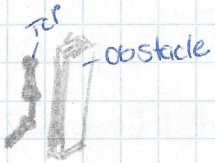
$x = f(\theta)$

$\dot{x} = J(\theta) \cdot \dot{\theta}$

$\dot{\theta} = J^{-1}(\theta) \cdot \dot{x}$

$J(\theta) = \frac{\partial f(\theta)}{\partial \theta} \in \mathbb{R}^{m \times n}$

Workspace
$W = \mathbb{R}^6$ Cartesian space
TCP = Tool center Point



C-Space Robot's configuration space
- n-dimensional
- C_{free} = All collision-free configurations
- C_{obs} = All configurations that result in a collision in workspace
- $C = C_{free} \cup C_{obs}$
- C_{free}, C_{obs} change during task execution

Motion planning for Grasping Tasks

Given:
- Start configuration c_{start}
- target config. c_{goal}
- set of obstacles in workspace
- kinematic chain of the robot

Task: compute collision-free motion from c_{start} to c_{goal}

Problem:
- high-dimensional c-space
- c_{obs} not available → generating is time consuming

Solution: sampling-based methods
- → Probabilistic Roadmaps
- → RRT

~~Graphs~~ Se

Graph search: A* best-first algorithm

Finds optimal path from v_s to v_{goal}
→ optimal in terms of path costs

- 2 lists of nodes
  - > Open set O = nodes to explore
  - > closed set C = visited nodes
- update all visited nodes v_n:
  - > pred(v_n)
  - > $g(v_n)$ accumulated costs to reach node
  - > $h(v_n)$ heuristic that estimates costs to reach v_g

Init $O = \{v_s\}$
$C = \emptyset$
$g(v_i) = \infty$, $1 \leq i \leq k$
$g(v_s) = 0$

Loop while $O \neq \emptyset$:
- → expand $v_i \in O$ with minim. $f(v_i) = g(v_i) + h(v_i)$
- → if $v_i = v_g$
  - ⇝ solution: travers pred.(v_i) until v_s
- → O. remove(v_i)
- → C. add(v_i)
- → update all successors v_j of v_i:
  - $v_j \in C$? continue
  - $v_j \notin O$? O. add(v_j)
  - $g(v_i) + cost(v_i, v_j) < g(v_j)$?
    - $g(v_j) = g(v_i) + cost(v_i, v_j)$
    - $h(v_j) = heuristic(v_i, v_g)$
    - $pred(v_j) = v_i$

> heuristic must not overestimate minimal cost of reaching v_g

Rapidly-Exploring Random Trees (RRT)

- Single query motion planner → suitable for changing scenes
- Randomized Methods
  - no guarantee that a solution is found within time horizon
  - probabilistically complete: if solution exists RRT will find it given time is infinit
- Performance: efficient for large set of planning problems
- Cobs known since RRT works in C-space

Init empty tree T, add c_{start} to T

Loop
1. Generate random samples cs
2. search nearest neighbor c_{nn} in T
3. add intermediate samples on path to T
   - step size d
   - check subpath for collisions: stop on collision detection
4. Goto 1

Every k-th loop: check if c_{goal} can be connected to T

Voronoi-Regions:
The probability that an RRT node is selected for expansion is proportional to the size of the corresponding Voronoi region

Bidirectional RRTs
Build two trees: T_1 init. with c_{start}, T_2 with c_{goal}
each sample extends both trees

Postprocessing for smooth trajectories
perform k times:
- select 2 points of the trajectory
- if direct connection is collision free: replace subpath

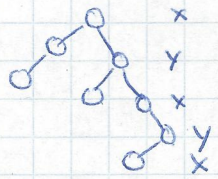
Collision detection
C_{obs} not available?
each $c \in C$ results in a workspace pose of the robot → check for collisions in workspace

path collision? sample the path segment
→ check segments for collisions

Nearest Neighbor search different approaches

(1) Linear search $O(n)$
calculate all distances $d_i(Q, P_i)$

(2) kd-Trees Each level has a cutting dimension
- cycle through dimensions as walking down the tree
- each node contains a point $P=(x,y)$
- insert point (x', y'):
  move down the tree
  - at each node: compare coordinate of the cutting dimension to decide whether to descent left or right
  - eg: $x' < x$?
    yes → left, no → right



bd = best distance

→ Find Nearest Neighbor(Q):
(1) descend in the tree as if the point was inserted
(2) reached leaf N: update: current_NN = N, current_bd = d(Q, N)
(3) start ascending again
(4) during ascent: calculate for each N: 1. $d(Q,N) <$ current_bd? yes: aktualisieren
2. check if other subtree needs to be checked:
- compute distance in corresponding cutting dimension, if less than bd:
  → descent this tree similar to 1 until leaf
  → check for improvement
  → goto (3)

(5) break when root is reached and no further descent possible

- worst case $O(n)$
- average case $O(2^d + \log(n))$

(3) Octrees trees with branching factor 8
doesn't require rebalancing like kd-Trees

Probabilistic Roadmaps (PRM)

- sampling based motion planning algo for multi-query tasks
- Buildup and query phase
- may be incomplete: no guarantee that PRM finds solution if it exists

Buildup

Initialize empty graph G (roadmap)
loop: create random samples c_S
add c_S to G
try to connect c_S to G
stop after k loops
→ may result in disconnected subgraphs

Query

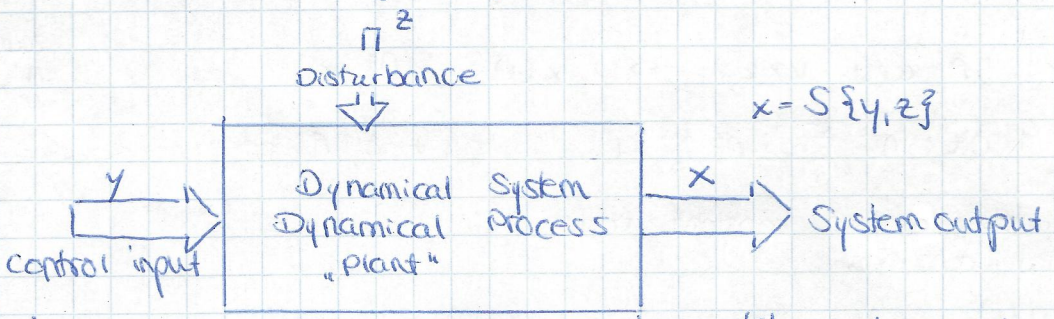
connect c_{start} and c_{goal} with the roadmap
search path, eg with A*

Dynamics of Linear Systems = DoLS

Control Theory (= Regelungstechnik)
Theory of automatic, goal-orientated modification of dynamical processes at run-time

Fundamental question: design of a system for automatic, targeted modification of a process with incomplete knowledge of the underlying process and in the presence of disturbances

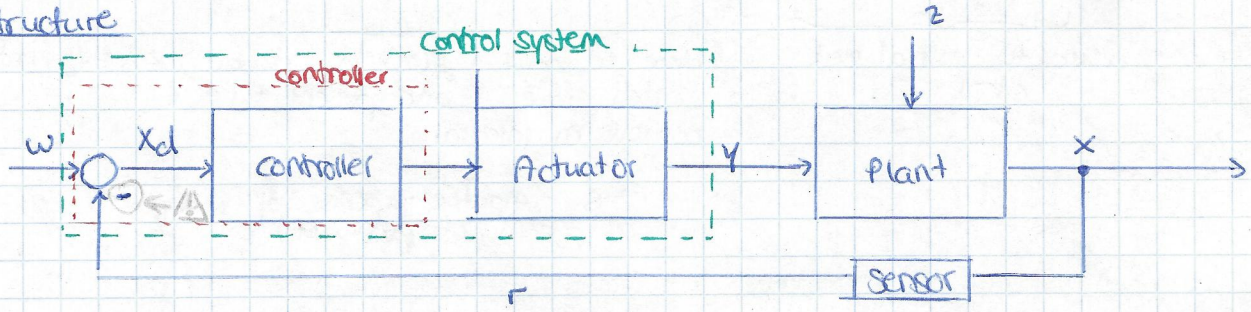
~> methods of control theory are universally applicable, independent of the specific nature of the given system



The system output is to be influenced by the control input to achieve the desired output despite unknown/partially known disturbances

Closed-loop control system
fulfills principle of operation: plant monitored continuously, obtained info used to modify input to achieve desired output despite disturbances

Structure



w = reference
x = system output
x_d = control error
r = feedback
y = control input
z = disturbance

Desired value of x : x_s
Measured value of x : $r = k_j \cdot x$ $\quad k_j > 0$, const.
Reference value : $w = k_j x_s$

$$\Rightarrow \quad x_d = w - r = k_j (x_s - x)$$

Control system definition
A control system is an arrangement that continuously monitors the plant's output, computes the deviation from a reference value and uses this error to adjust the system output to match the reference.
This is achieved with limited knowledge about the plant and the disturbance

Differential equations

linear systems: $\dot{x} = A(t)x + B(t)u$ — x = system state vector, u = input / control vector

↝ develop general formula for a vector-matrix differential equation in terms of the state-transition matrix (stm)

state-transition matrix: describes how state $x(t)$ at time t evolves to state $x(\tau)$ at different time τ

- Time-invariant systems: stm = matrix exponential function
- Time-varying systems: no easy expression for stm

Solutions

- $\boxed{\dot{x} = Ax}$ A = const., $k \times k$ $\Rightarrow$ $x(t) = e^{At} \cdot c$

zum prüfen der Lsg: $\frac{d}{dt}x$ berechnen, einsetzen

$$e^{At} = 1 + At + A^2\frac{t^2}{2!} + A^3\frac{t^3}{3!} + \dots = 1 + \sum_{i=1}^{\infty} A^i \frac{t^i}{i!}$$

c = suitably chosen constant vector
1 = initial condition

- $e^{A(t_1+t_2)} = e^{At_1} \cdot e^{At_2}$ $\Rightarrow$ $(e^{tA\tau})^{-1} = e^{-A\tau}$

$$\Rightarrow x(t) = e^{A(t-\tau)}x(\tau)$$

- $\boxed{\dot{x} = Ax + Bu}$

$$x(t) = \underbrace{e^{A(t-\tau)}x(\tau)}_{\text{due to initial state } x(\tau)} + \underbrace{\int_\tau^t e^{A(t-\lambda)}Bu(\lambda)\,d\lambda}_{\text{due to input } u(\tau) \text{ in interval } \tau \le \lambda \le t \text{ between initial time } \tau \text{ and present time } t}$$

- convolution integral (= Faltung)

↝ $e^{At}B$ = impulse response of the system with input $u(t)$, output $x(t)$

output y:

$$y(t) = C(t)e^{A(t-\tau)}x(\tau) + \int_\tau^t C(t)e^{A(t-\lambda)}B(\lambda)u(\lambda)\,d\lambda$$

⇒ both $x(t), y(t)$ do not hold when A is time-varying

↝ in linear systems: dependence is linear ↝ solution to $\dot{x} = A(t)x$ is $x(t) = \Phi(t,\tau)x(\tau)$

$\Phi(t,\tau)$ is stm

$\Phi(t,\tau) = e^{A(t-\tau)}$ for time-invariant systems

State transition matrix (stm)

- time-invariant systems: stm is function of $t-\tau$
  - τ = initial time
  - t = present time
  - no loss in generality by setting $\tau=0$ and computing $\Phi(t)=e^{At}$
  - → replace t with $t-\tau$ in subsequent calculations
- time-varying systems: $\Phi(t,\tau)$ cannot be deducted from $\Phi(t,0)$

Properties

- stm is expression of the solution to the homogeneous equation
  $\dot{x}(t) = A(t)x(t) \rightsquigarrow x(t) = \Phi(t,\tau)\,x(\tau)$
- time derivative of $x(t)$ must satisfy the equation for any t and $x(t)$
- $x(\tau)$ = initial data, not a time function
  $\Rightarrow \dfrac{dx(t)}{dt} = \dfrac{\partial\Phi(t,\tau)}{\partial\tau}\,x(\tau)$
- stm is function of two arguments t,τ
- $\Phi(t,\tau) = e^{A(t-\tau)} \Rightarrow \dfrac{\partial\Phi(t,\tau)}{\partial t}\,x(\tau) = A(t)\,\Phi(t,\tau)\,x(\tau)$

  $\Rightarrow \dfrac{\partial\Phi(t,\tau)}{\partial t} = A(t)\,\Phi(t,\tau)$ $\quad\boxed{\dot{\Phi} = A\Phi}$

  $\rightsquigarrow \Phi$ satisfies same differ. eq. as state x
- $x(t) = \Phi(t,t)\,x(t) \quad \forall x(t)$

  $\Rightarrow \Phi(t,t) = I \quad \forall t$
- A = constant matrix $\Rightarrow \Phi$ exists, $\Phi(t) = e^{At}$ is valid
- $\Phi(t_3,t_1) = \Phi(t_3,t_2)\,\Phi(t_2,t_1)$ (semi-group property)
- $\Phi(t,\tau) = \Phi(\tau,t)^{-1} \Rightarrow \Phi$ is never singular

Delta function

$$\delta(t) = \begin{cases} \infty, & t=0 \\ 0, & t\neq 0 \end{cases}$$

unit step "jump" function

$$\sigma(t) = \begin{cases} 0, & t<0 \\ 1, & t\geq 0 \end{cases}$$

Laplace Transforms

F = laplace transform of f

Laplace transform of signal f(t)

$$\mathcal{L}[f(t)] = F(s) = \int_0^\infty f(t)\, e^{-st}\, dt$$

s = complex frequency $s = \delta + j\omega$

- $\mathcal{L}[\dot{f}(t)] = s\int_0^\infty e^{-st} f(t)\,dt - f(0) = \underline{s \cdot F(s) - f(0)}$
- $\mathcal{L}\left[\int_0^t f(t)dt\right] = \underline{\frac{1}{s} F(s)}$
- $\mathcal{L}[\delta(t)] = \int_0^\infty \delta(t) e^{-st}\,dt = \underline{1}$
- $\mathcal{L}[\sigma(t)] = \frac{1}{s}$
- $e^{-\alpha t} = \begin{cases} 0 & , \alpha > 0 \\ 1 & , \alpha = 0 \\ -\infty & , \alpha < 0 \end{cases}$ Schwingung klingt ab / Dauerschwingung / Schwingung klingt auf

$\mathcal{L}[e^{-\alpha t}] = \frac{1}{s+\alpha}$

Rules

- $\mathcal{L}[f(t)+g(t)] = \mathcal{L}[f(t)] + \mathcal{L}[g(t)]$
- $\mathcal{L}[af(t)] = a \cdot \mathcal{L}[f(t)]$, $\mathcal{L}[A \cdot \dot{x}(t)] = A \cdot X(s)$
- $\mathcal{L}\begin{bmatrix} f_1(t) \\ f_2(t) \\ \vdots \\ f_n(t) \end{bmatrix} = \begin{bmatrix} \mathcal{L}[f_1(t)] \\ \vdots \\ \mathcal{L}[f_n(t)] \end{bmatrix} = F(s)$
- $\mathcal{L}[e^{at}] = \frac{1}{s-a}$ $\rightsquigarrow$ $\mathcal{L}[e^{At}] = (s \cdot I - A)^{-1}$
- $\mathcal{L}\left[\int_0^t f(t-\lambda) g(\lambda)\,d\lambda\right] = F(s) \cdot G(s)$

State transition matrix

$\Phi(s) = (s \cdot I - A)^{-1}$ = resolven of A = characteristic frequency matrix
$\rightsquigarrow$ characterizes dynamic behaviour of the system

$\rightsquigarrow$ calculate stm:
(1) $(s \cdot I - A)$ calculate
(2) obtain resolvent by inverting $(^{-1})$
(3) element-wise inverse laplace transform

Transfer functions

input - output relations

input u, output y, initial state $x(0) = 0$ (assumed)

- Laplace-transform of the state $X(s) = (s \cdot I - A)^{-1} \cdot B \cdot U(s)$
- output $y(t) = C \cdot x(t)$

  $\Rightarrow Y(s) = C \cdot X(s)$

  $\Rightarrow Y(s) = C \cdot (s \cdot I - A)^{-1} \cdot B \cdot U(s)$
- Transfer-function matrix: (relates $\mathcal{L}$[output] to $\mathcal{L}$[input])

  $H(s) = C \cdot (s \cdot I - A)^{-1} \cdot B$
- Impuls-response matrix

  $H(t) = \mathcal{L}^{-1}[H(s)] = C \cdot e^{At} \cdot B$
- $y(t)$ convolution of impulse response matrix:

  $y(t) = \int_0^t H(t-\lambda)\, u(\lambda)\, d\lambda = \int_0^t C e^{A(t-\lambda)} B\, u(\lambda)\, d\lambda$

Transformation of state variables

$z = Tx$ with T non-singular $k \times k$ $\Rightarrow x = T^{-1} z$

$\rightsquigarrow \quad \dot{x} = Ax + Bu, \quad y = Cx + Du$

$\rightarrow \quad \dot{z} = \bar{A} z + \bar{B} u, \quad y = \bar{C} z + \bar{D} u$

$\bar{A} = TAT^{-1} \quad \bar{B} = TB \quad \bar{C} = CT^{-1} \quad \bar{D} = D$

Action Representation AR

Assumption: Movement sequences consist of segments → motion primitives

Action Representation:
- Reusability - composed to actions for solving complex tasks
- Flexibility - different conditions possible
- Adaptivity - goal directed execution in dynamic environments
- Portability - same action repr. on different robots

Requirements
- Formalism to describe functions as dynamic systems
- Learnable from demonstration
- movement is generalized
- extendable for online feedback

Dynamic Movement Primitives

(1) · Demonstration by human
→ Position + velocity for each time step y_0, v_0

· Arbitrary start pos. y_0, velocity v_0

· Arbitrary goal position g

· Arbitrary duration (e.g. time T)

(2) Trajectory from y_0 to g in duration T with characteristic shape of demonstration

⇒ Dynamical systems: critically damped spring mass system with perturbation

Damped spring-mass system

$$\ddot{x} = \underbrace{K(g-x)}_{\text{spring term: goal attractor}} - \underbrace{D\dot{x}}_{\text{Damping term: limits acceleration}}$$

x = current pos.
D = damping const.
K = spring const.

↳ converges to g

→ transform to 1st order system: substitution

$$\dot{x} = v = \text{current velocity}$$

$$\rightarrow \quad \dot{v} = K(g-x) - Dv$$

with temporal factor τ:

$$\tau\dot{v} = K(g-x) - D \cdot v$$
$$\tau\dot{x} = v$$

$\tau < 1$: Mass-spring is faster
$\tau > 1$: slower

Transformation system

perturbation force term: shape trajectory

$$\tau \dot{x} = v$$

$$\tau \dot{v} = K(g - x) - D \cdot v + \underbrace{(g - x_0) \cdot f(u) \cdot u}_{\text{scaled perturbation term}}$$

perturbation force

→ $f(u)$ is learned from demonstration

- $f(u) = 0$ → unique attractor point g (converge to g)
- $f(u)$ phasic (active in finite time window) → point attractor (discrete movement)
- $f(u)$ periodic → oscillator (periodic movement)

constants D, K: damped spring-mass system oscillates around the goal

→ solution: critically damped system: damping just as strong to compensate overshooting over goal

Canonical system including time independence

- $f(u)$ depends on time $\rightarrow$

→ use canonical system with phase variable u instead of explicit time dependence

$$\tau \dot{u} = -\alpha u \quad , \quad \alpha = -(\log(u_{min}))\tau$$
$$u_{min} = 0,001$$
$$u(0) = 1$$

Perturbation

perturbation force f determines shape of trajectory

- non-linear function representing demonstrated trajectory
- f depends on canonical system $u(t)$

$$f = \frac{-K(g - x) + Dv + \tau \dot{v}}{(g - x_0)u}$$ (from transf. sys.)

→ Given demo y calculate f for all timestamps with

$$g = y(T) \qquad x, v, \dot{v} = y, \dot{y}, \ddot{y}$$
$$x_0 = y(0) \qquad \tau = T$$

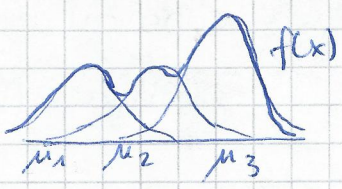
⇒ approximate f with continuous representation

→ Algos:
- RBF
- Splines
- LWPR
- ...

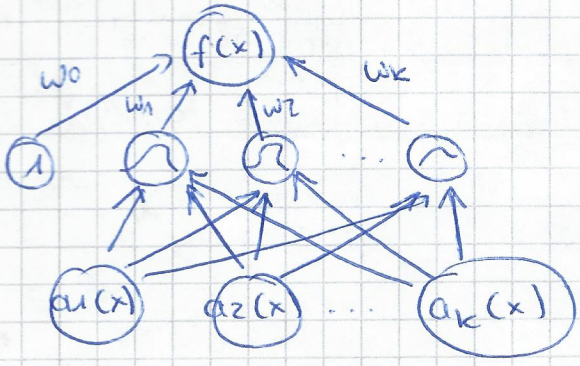
(weighted) Radial Basis Function (W)RBF

RBF

- Function approximation $\hat{f} = w_0 + \sum_{u=1}^{K} w_u \cdot K_u(x)$



- Kernel function $K_u(x) = e^{-\frac{1}{2\sigma_u^2} \|\mu_u - x\|^2}$
- Modelled as neural network: one hidden layer, one output neuron



Training:

(1) determine receptive fields:
- # k of hidden neurons
- Params μ_u, σ_u
  - one neuron / sample or
  - uniformly distributed in X or
  - based on clustering of the training data or
  - Expectation maximization (finds optimal)
  - ...

(2) determine weights w_u
- as in regular NNs
- μ_u, σ_u fixed

WRBF locally weighted regression

$$f_{approx}(u) = \frac{\sum_{i=1}^{N} w_i \cdot \psi_i(u)}{\sum_{i=1}^{N} \psi_i(u)}$$

h_i = width of RBF
c_i = center of RBF
w_i = weight of RBF

RBF: $\psi_i(u) = e^{h_i (u - c_i)^2}$

Finding w_i is 1D optimization problem (with fixed h_i, c_i)

⊕ · can reduce jitter from original demonstration
· can reduce memory consumption

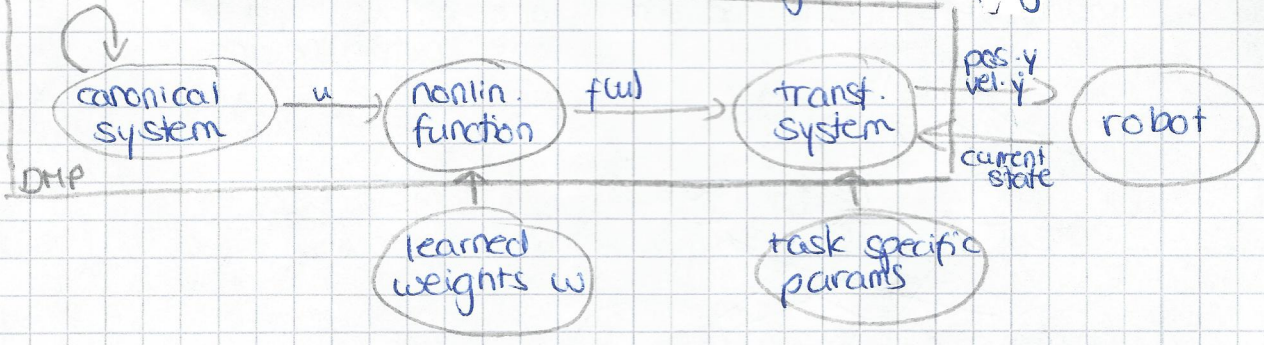
DMP Execution

canonical system $\dot{u} = -\alpha u$

nonlinear function $f(u) = \frac{\sum_i w_i \psi_i(u)}{\sum_i \psi_i(u)}$, $\psi_i(u) = e^{h_i (u - c_i)^2}$

transformation system $\tau \dot{x} = v$

$\tau \dot{v} = K(g - x) - Dv + r(g - x_0) f(u) u$



Online Feedback

External Perturbation force

- phase stopping: phase slows down until ext. pert. force is gone (e.g. person holds arm of robot)
- modifying canonical system for phase stopping

$$\tau \dot{u} = \frac{-\alpha u}{1 + \alpha_{pu} |\tilde{x} - x|}$$

x = desired pos.
$\tilde{x}$ = actual pos.

α_{pu} controls how fast phase slows down

- transf. system change:

$$\tau \dot{x} = v + a_{px} |\tilde{x} - x|$$

changing goal

simple online changing would lead to a jump in acceleration

→ filtering the goal with the ODE $\dot{g} = \lambda (g - g_d)$

g_d = desired goal
λ = goal change rate factor

Periodic DMPs

observation: most periodic motions start with discrete part
↳ transient part
↳ needed for well-defined start point of periodic motion
↳ have to be learned

=) · Canonical system with $\dot{\phi} = \Omega$, Ω = freq. of periodic movement

· $\tau = 1/\Omega$

· $f(\phi) = \frac{\sum_i w_i \Gamma_i(\phi)}{\sum_i \Gamma_i(\phi)} r$ $\qquad \Gamma_i(\phi) = e^{h_i(\cos(\phi - c_i) - 1)}$

· $\dot{x} = \Omega v$
$\dot{v} = \Omega k(g - x) - Dv + (g - x_0) f(\phi) \phi$

Goal: one DMP that can encode periodic movements AND all transients in a single dynamic system

General DMP

two parts:
· canonical system: state of DMP in time drives perturbation to control transf. sys.

· transf. system: generates motion according to state of canonical system

→ use 2D canonical System

i.e. $s(t) = (\varphi(t), r(t)) \in \mathbb{R} \times (0, \infty)$ for φ, r solution of

$$\begin{cases} \dot{\varphi} = \Omega \\ \dot{r} = \eta(\mu^\alpha - r^\alpha) r^\beta \\ \varphi(0) = \varphi_0 \\ r(0) = r_0 \end{cases} \qquad \mu, \eta, \alpha, \beta > 0, \text{ const.} \quad \Omega > 0$$

⇝ $r(t) \to \mu$ $(t \to \infty)$ (monotonously)
α, β, η define speed, shape of convergence
(φ, r) as polar coord.

transf. syst.

$$\begin{cases} \dot{z} = \Omega(\alpha_z(\beta_z(g - y) - z) + f(\varphi, r)) \\ \dot{y} = \Omega z \end{cases}$$

$\alpha_z, \beta_z > 0$ const. $\qquad g \in \mathbb{R}$ anchor point

$f(\varphi, r)$ 2D input

$$f(\varphi, r) = \frac{\sum_{j=1}^{M} \psi_j(\varphi, r) \tilde{w}_j + \sum_{i=1}^{N} \gamma_i(\varphi, r) w_i}{\sum_{j=1}^{M} \psi_j(\varphi, r) + \sum_{i=1}^{N} \gamma_i(\varphi, r)}$$

Machine Learning = ML

Common Tasks:

- Classification: Assign input to one (predefined) category
- Regression: Predict numerical value(s) given input
- Clustering: establish structure of clusters given observed data points
- Image Segmentation: Detect objects + boundaries
- Anomaly Detection: Detect unusual instances in a set or series of observations
- ...

ML Foundations

Linear Algebra Basics

- Tensor: Generalization of vectors and matrices → array of numbers with variable number of axes
- Linear combination of vectors: $\sum_i c_i v^{(i)}$
- Span of vectors: Set of all points obtainable by linear comb. of these vectors
- Linear independence: no vector can be obtained by linear comb. of the others
- Norms f with
  - $f(x) = 0 \Rightarrow x = 0$
  - $f(x+y) \leq f(x) + f(y)$
  - $\forall \alpha \in \mathbb{R}: f(\alpha x) = |\alpha| f(x)$
  - L^1 Norm: $\|x\|_1 = \sum_i |x_i|$
  - L^2 Norm $\|x\|_2 = \|x\| = \sqrt{x_1^2 + \dots + x_n^2}$ Euclidian norm
  - L^∞ Norm $\|x\|_\infty = \max_i |x_i|$ Max norm
- Diagonal Matrix: $D_{ij} = 0 \quad \forall i \neq j$
- $\vec{x}, \vec{y}$ orthogonal: $x^T y = 0$
- A orthogonal: $A^T A = A A^T = I \Rightarrow A^T = A^{-1}$
- Eigenvector v, Eigenvalues λ $Av = \lambda v, \lambda \in \mathbb{R}$, A quadratic!
  $A \in \mathbb{R}^{n \times n} \Rightarrow$ max n λs
- Determinante: $\det(A) = \prod_i \lambda_i$

Singular Value Decomposition factorization strategy

Factorize $A \in \mathbb{R}^{m \times n}$ in matrices U, D, V such that $A = UDV^T$
with U orthogonal, $m \times m$
D diagonal $m \times n$
V^T orthogonal $n \times n$

- Diagonal elems of D: singular values of A
- columns of U: orthonormal vectors, eigenvectors of AA^T: left-singular vectors
- columns of V: -"- A^TA right-singular vectors

Principal Component Analysis (PCA)

$\frac{w^T A^T A w}{w^T w}$ Rayleigh quotien: w that max. this coincides with eigenvec. of A^TA

$\rightarrow \max w^T X w$, $w^T w = 1$ where $X = A^TA$

- Lagrangian fuction $L = w^T X w - \lambda (w^T w - 1)$
  - $\rightarrow$ Derivative with respect to w: $\nabla_w L = Xw - \lambda w = 0$
    $\Rightarrow$ w = eigenvec. of X

$\rightarrow$ fist J eigenvalues of A^TA corespond to fist J principal components

Definitions

- Continuous probability distribution p for random variable x
  (1) Domain of function p must be set of all possible states of x
  (2) $\forall x \in X : p(x) \geq 0$
  (3) total probability mass $\int p(x) dx = 1$

- Expected value $\mathbb{E}_{x \sim p}[f(x)] = \int p(x) f(x) dx$

- Variance $\text{Var}(f(x)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$
  expected deviation of value from expected value

- Gaussian (Normal) Distribution:
  - μ = mean = expected value
  - σ = standard deviation (σ^2 = variance)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Mixture Models more complex stochastic models by combining simple distributions

- Mixture Distribution $P(x) = \sum_{i=1}^{N} P(z=i) P(x|z=i)$

$P(z)$ = Multinomial distribution
$P(x|$ = Mixture component

- Learning: Expectation-Maximization Algorithm
  alternate between
  (1) Expectation step
  compute likelihoods of each training sample belonging to Gaussian distr.

  (2) Maximization step:
  recompute model params based on assigned training samples

Mathematical optimization

Algo that finds params to $\min/\max f(x)$

$f(x)$ = objective function

Approaches: different types of constraints
- Gradient-based
- derivative free

Gradient Descent

gradient of objective function

$$\nabla_x f(x) = \left(\frac{\delta}{\delta x_1} f(x), \dots, \frac{\delta}{\delta x_n} f(x)\right)^T$$

→ move to gradient $x' = x - \eta \cdot \nabla_x f(x)$ $\quad 0 < \eta < 1$ (learning rate)
→ stop when local extremum reached
⇝ global extr. depends on init of x

Newtons Method

problem with 1st order method: rapidly change of gradient around x_0 may lead to slow convergence or divergence due to overshooting

→ 2nd order derivative
Hessian matrix

$$H(f)(x)_{i,j} = \frac{\delta^2}{\delta x_i \delta x_j} f(x)$$

Approximate $f(x)$ around x_0 using Taylor series:

$$f(x) \approx f(x_0) + \underbrace{(x - x_0)^T \cdot \nabla_x f(x_0)}_{\text{Adjustment according to jacobian}} + \underbrace{\frac{1}{2}(x - x_0)^T \cdot H(f)(x_0) \cdot (x - x_0)}_{\text{2nd order adjustment according to Hessian}}$$

Basic Concepts

Supervised Learning

Dataset (= training set) is labeled → each sample associated with a label or target value

Unsupervised learning

Dataset not labeled → learning algo must establish structure of dataset

- Test set: disjunct from training set to evaluate model performance
- Generalization: Ability of learned model to perform well on previously unobserved inputs
- overfitting: model reproduces training data well, but cannot generalize
  → model might "memorize" training set
- underfitting: model cannot even reproduce training data
- model capacity: "power" of the model, amount of data the model can learn
  - too low: model cannot perform well
  - too high: model prone to overfitting

N-Fold Cross Validation how to devide data into training and test set?

- Static split: 50% of data as training data
  => less data available overall

-> N-Fold CV: - split data into n equally-sized partitions (distribution of labels roughly equal)

- run n folds (rounds) where:
  - one partition used as test data
  - n-1 partitions for training

- combine performance metrics from all n folds

Linear Regression

simple model for regression tasks: estimate value of one variable y given an arbitrary number of input variables (in n-dim vec. x)

-> Prediction $\hat{y} = w^T x + b$

w?

- Least-Squares Method: minimize $\frac{1}{m} \| (\hat{y} - y) \|_2^2$ (sum of squared errors)

for traing data X / Y

-> solve derivative to zero

$\hat{y} = Xw$

$$
\begin{aligned}
& \nabla_w \tfrac{1}{m} \| \hat{y} - y \|_2^2 = 0 \\
\Leftrightarrow\ & \nabla_w \| Xw - y \|_2^2 = 0 \\
\Leftrightarrow\ & \nabla_w (Xw - y)^T (Xw - y) = 0 \\
\Leftrightarrow\ & \nabla_w (w^T X^T X w - 2 w^T X^T y + y^T y) = 0 \\
\Leftrightarrow\ & 2 X^T X w - 2 X^T y = 0 \\
\Leftrightarrow\ & w = (X^T X)^{-1} X^T y
\end{aligned}
$$

=> $w = (X^T X)^{-1} X^T y$

Support Vector Machines

binary classifier for supervised linear classification

-> seperation of linearly seperable data points in two classes

-> classification represented by a seperating hyperplane

specified by $wx - b = 0 \quad \forall x \in$ hyperplane

-> $wx - b$ sign determins classification result for $x \notin$ hyperplane

-> choose hyperplane with max. margin to datapoints on both sides

-> MAXIMUM-MARGIN HYPERPLANE

support vectors: data points close to hyperplane relevant to determine hyp.pl.

-> normalize w,b: $wx^{\oplus} - b = 1$ and $wx^{\ominus} - b = -1$

-> margin is then $\frac{w}{\|w\|}(x^+ - x^-) = \frac{w(x^+ - x^-)}{\|w\|} = \frac{2}{\|w\|}$

Training of SVM is optimization problem:

$$\max_w \frac{2}{\|w\|} \quad \text{or} \quad \min_w \|w\|^2$$

$$wx_i - b \begin{cases} \geq 1 & , x_i \in \text{class } + \\ \leq -1 & , x_i \in \text{class } - \end{cases}$$

Soft-margin SVM non-linearly seperable data

Allow classification error $\xi_i \geq 0$ for x_i

- $\xi_i < 1$: x_i still on correct side, but margin not respected
- $\xi_i \geq 1$: x_i on wrong side

extend optimization problem:

$$\min_{w, \xi} \|w\|^2 + C \cdot \sum_{i=1}^{\hat{n}} \xi_i$$

C - regularization param: tradeoff btw. correct classification and max. of margin

$$wx_i - b \begin{cases} \geq 1 - \xi_i & , x_i \in \text{class } + \\ \leq -1 + \xi_i & , x_i \in \text{class } - \end{cases}$$

Kernel trick n-l. sep. data

project data points into high-dim. space where they are lin. sep.

Clustering trying to establish a structure in an unlabeled data set by assigning data points to clusters

- intra-cluster distances should be low
- inter-cluster distances high
- define distance metrics

Approaches

- Partitional clustering : number of clusters given, ask for optimal assignment of data points to clusters
  k-means, GM
- hierarchical clustering : close points/clusters connected to large cluster(s) are split

k-means k clusters (k given)

- every cluster $j = 1, \ldots, k$ associated with mean (center) m_j
  → init randomly or based on data points

→ min. variance inside each cluster

$$\sum_{j=1}^{k} \sum_i b_{i,j} \|x_i - m_j\|_2^2 \quad \text{with} \quad b_{i,j} = \begin{cases} 1 & , x_i \text{ assigned to } j \\ 0 & , \text{otherwise} \end{cases}$$

Lloyd's Algo iterative approach

(1) assignment step: associate each x_i to closest cluster

$$b_{i,j} = \begin{cases} 1 & , \|x_i - m_j\|_2 = \min_k \|x_i - m_k\|_2 \\ 0 & , \text{otherwise} \end{cases}$$

(2) update step: recompute all m_i from assigned samples

$$m_j = \frac{\sum_i b_{i,j} x_i}{\sum_i b_{i,j}}$$

Terminate when assignment doesn't change anymore

⊕
- simple + efficient
- guaranteed to converge

⊖
- k needs to be known
- initialization of cluster affects result
- no outlier detection

Artificial Neural Networks = ANN

Idea: build mathematical model inspired by the brain

Learning

Inductive
Learning from examples (training data)

Deductive
Generating new rules based on existing ones

knowledge representation

Symbolic
knowledge repr. as set of rules

Subsymbolic
knowledge repr. as params of statistical models

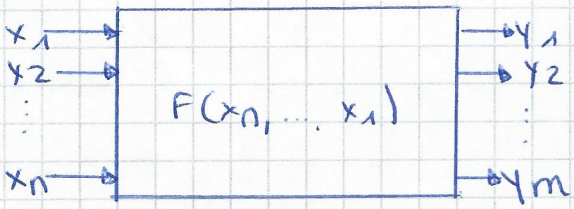
Learning methods

Supervised
Feedback / Label supplied with training data

Unsupervised
no feedback

=> Neural networks are ...
- ... inductive: many training examples
- ... subsymbolic: learning weight params
- ... supervised: positive and negative training data examples

ANN as a black box



F calculated by a net of functions

$F: \mathbb{R}^n \to \mathbb{R}^m$

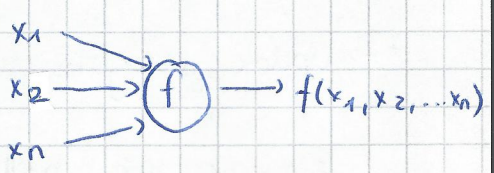
composition of functions

Information flow determined by connection topology structure

- Loop-free: deterministic calculation $x \to (g) \xrightarrow{g(x)} (f) \to f(g(x))$
- with loops: recursive calculation taking time into account $x_n \to (f) \to f(x_t, f(x_{t-1}, f \dots))$

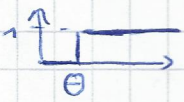
ANN as a graph

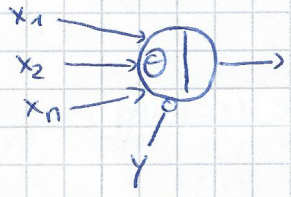
- nodes = calculation units (neurons, cells)
  -> several independent inputs, one output
- edges = directed information channels from one neuron to another arguments of the function



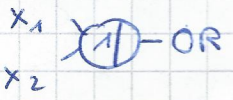
MCP neuron binary input, binary output

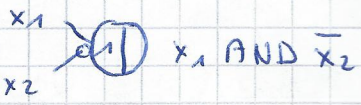
→ threshold function





Boolean functions:

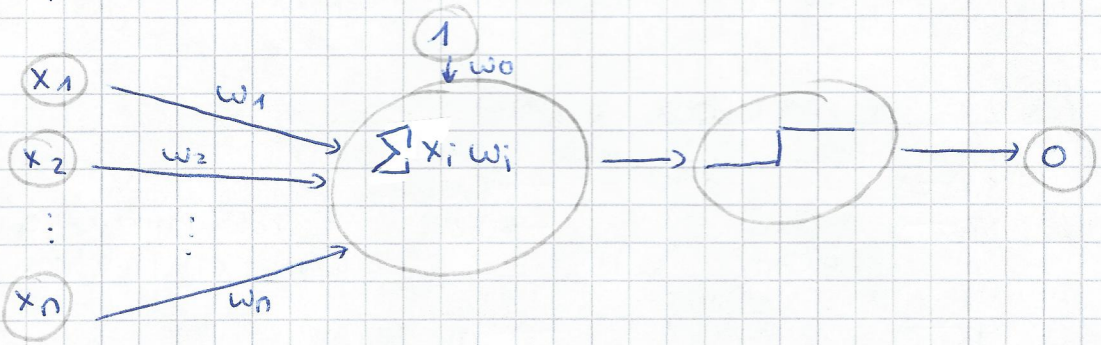




Perceptron drastically simplified computational description of a neuron

$$o(x_1, \dots, x_n) = \begin{cases} 1, & w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ 0, & \text{otherwise} \end{cases} \qquad x_i, w_i \in \mathbb{R}$$

→ binary output, perceptron activated if threshold surpassed



Simplified formalism: $o(x) = \sigma(\langle w, x \rangle)$ homogeneous system of linear equations

$x = (1, x_1, \dots, x_n) \qquad w = (w_0, w_1, \dots, w_n)$

$$\sigma(y) = \operatorname{sgn}(y) = \begin{cases} 1, & y > 0 \\ -1, & \text{otherwise} \end{cases}$$ activation function

⇝ $x_i \in \mathbb{R}$: only linearly seperable data classifyable
→ combine neurons to ANN

Training

$w_i = w_i + \Delta w_i$ where $\Delta w_i = \eta (t - o) \cdot x_i$

→ update weights given $t = t(x)$ = target output
$o = o(x)$ = perceptron output
$\eta \in (0, 1)$ = learning rate

→ converges for small η, fails when data not linearly seperable

Gradient Descent — no activation function

Solution: optimization of parameters, converges to best-fit approximation if data is not linearly seperable

$$o(x) = \langle w, x \rangle = w_0 + w_1 x_1 + \dots + w_n x_n$$

→ Error function $E(w) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$

D = set of training examples

→ update weights:

$$w_i = w_i + \Delta w_i \quad , \quad \Delta w_i = -\eta \frac{\delta E}{\delta w_i} = \eta \sum_{d \in D} (t_d - o_d(x)) \cdot x_{id}$$

where $\nabla E(w) = \left(\frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}, \dots, \frac{\delta E}{\delta w_n} \right)$

⊖ · sometimes slow convergence
· not guaranteed to find global minimum
· all training samples at the same time

δ-Rule — Fundament of Backpropagation algorithm

Calculating error for each individual sample in D

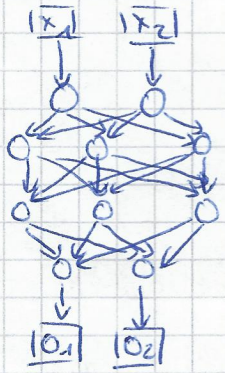
$$E_d(w) = \frac{1}{2} (t_d - o_d(x))^2 = \frac{1}{2} (t_d - \langle x, w \rangle)^2$$

$$\Rightarrow w_i = w_i + \Delta w_{id} \quad \forall d \in D$$

$$\delta_d = (t_d - o_d(x))$$

$$\Rightarrow \Delta w_{id} = \eta \cdot \delta_d \frac{\delta E_d(w)}{\delta w_i} = \eta \cdot \delta_d \cdot x_i$$

Feed-forward network — Multilayer Perceptron



activation function: non-linear, differentiable

- Sigmoid function $\sigma(y) = \frac{1}{1+e^{-y}}$

$$\frac{\partial \sigma(y)}{\partial y} = \sigma(y) \cdot (1 - \sigma(y))$$

- tanh function $\sigma(y) = \tanh(y) = 1 - \frac{2}{e^{2y}+1}$

$$\frac{\partial \sigma(y)}{\partial y} = 1 - \sigma^2(y)$$

- can express non-linear decision surfaces

- any boolean function can be represented with 2 layers
- any bounded continuous function -"-
- any arbitrary function can be approximated with 3 layers

Backpropagation = Gradient descent for feed-forward networks

- generalization of δ-rule for training of multi-layered networks
- Error function: squared error betw. output, target

$$E(w) = \frac{1}{2} \sum_{d \in D} \sum_{k \in O} (t_{kd} - o_{kd}(x))^2$$

D training set, O outputs

⊖ can have local minima ⇒

$$E_d(w) = \frac{1}{2} \sum_{k \in O} (t_{kd} - o_{kd}(x))^2$$

$$w_{ij,u} = w_{ij} + \Delta w_{ij} \qquad \Delta w_{ij} = -\eta \frac{\partial E_d(w)}{\partial w_{ij}} = \eta \cdot \delta_j \cdot x_{ji}$$

Notation:

- Nodes have indices ($\in \mathbb{N}$)
- x_{ji} = connection from node i to node j
- w_{ji} = corresponding weight
- δ_j = error term associated with i-th node
- $net_j = \sum_i w_{ji} \cdot x_{ji}$ weighted sum of inputs
- o_j = computed output, t_j target output for unit j

- propagate error backwards from output to input layer

Algorithm

- Init all weights to random numbers (usually $\in (0,1)$)
- Until satisfied do:
  - For each training example do:
    - compute output
    - for each output unit k: $\delta_k = o_k(1-o_k)(t_k - o_k)$
    - for each hidden unit h: $\delta_h = o_h(1-o_h) \sum_{k \in Out} w_{hk} \delta_k$
  - update network weights $w_{ij} = w_j + \Delta w_{ij}$

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot x_{ij}$$

Radial Basis Function Networks

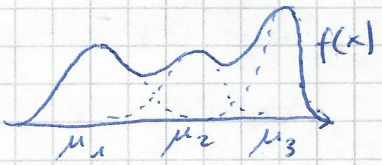
Function approximation: $\hat{f}(x) = w_0 + \sum_{u=1}^{K} w_u \cdot K_u(x)$

Kernel function: $K_u(x) = e^{\frac{1}{2\sigma_u^2} \|\mu_u - x\|^2}$

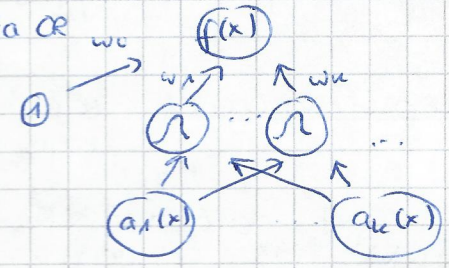
modelled as neural network: one hidden layer, one output neuron

Training

1) determine receptive fields
   - # K of hidden neurons
   - params x_u, σ_u
     - one neuron per sample OR
     - uniformly distributet in x OR
     - based on clustering of many data OR
     - ...



2) determine weights w_u
   - as in regular neural nets
   - x_u, σ_u fixed



Deep Learning

Learning problem:

- Vanishing gradient problem: naive backpropagation → error gets smaller towards the input layer
  → First layers not really affected by learning / learn slower

→ Error in certain layer depends on all layers before
  ↳ training is unstable with many layers

Convolutional Neural Networks

type of feed-forward neural network

Architecture:

- Alterning types of layers:
  - convolutional Layer(s): local receptive field
  - pooling layer: down-sampling
- Final set of fully-connected layers
- often simple activation function

> Convolutional Layer:

- consists of n feature maps
- each neuron inside a feature map considers input values in its local neighborhood → local receptive field
- all neurons in a feature map share same input weights

> Pooling layer

- Downsampling of input data → data reduction
- usually: max pooling → max value per region

> Fully Connected Layer:

- Follows last pooling layer
- determine output based on detected features
- usually few layers (2-3)

Deep Autoencoder search for low-dim. representation (encoding) for high-dim. representation of data

→ input data to NN, NN should reproduce input as output
→ NN has "bottleneck" layer (low-dim.)
→ After training: this layer is efficient encoding with encoders/decoders (layers input - bottleneck / layers bottleneck - output)

Deep Learning for Grasping

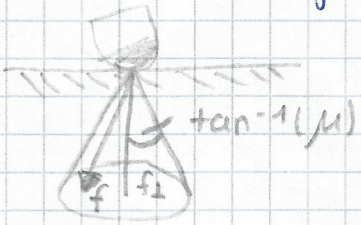
Grasping Problem: find grasp config. that satisfies set of criteria given an object

Classical Grasp Planing

- different metrics to evaluate grasp

Object model
Hand model
Hand config.
Hand pose
} → Grasp metric → Score
↑
Simulator

Evaluation: grasp friction cones
- defined by coefficient of friction btw. hand and object surface
- if force vector is inside the cone, finger tip won't slip
  → $F_{friction} > F_{tangential}$
- if force vector is outside the cone, finger tip will slip
  → $F_{friction} < F_{tangential}$



Training Data for Grasping

Learning by demonstration

- Human teacher
- record grasping data with motion capture and data gloves
- system tries to mimic grasps → generalizes learned data

⊕ uses human knowledge

⊖ · human demo/work needed
· transfer human grasp to robotic hands?
· how to generalize data?

Training data collection on the target system

· reinforcement learning approach
· system explores state space
· reward if object is grasped
· iterative refinement of strategy

⊕ · can deal with any complexity (in theory)
· self-supervised

⊖ · data inefficient → long exploration
· high network complexity/depth

Training data generation in simulation

· simulate hand/object interaction
· simulate robot sensors
· simulate grasp metrics

⊕ · object models known
· use of existing grasp planners/metrics
· inexpensive → big datasets possible

⊖ · Sim2real: no simulator is perfect
· overfitting to the simulator

Hand-labeled data

· Mark good grasps

⊕ good training data

⊖ · huge amount of work
· only available for labels in 2D RGBD images
· only available for grippers, not humanoid hands (yet)

Deep Learning Approaches

- Learning by Demonstration
- Grasp hypothesis generation and scoring
- Sim2real
- Learning on the target system

Grasp hypothesis generation and scoring

- each entry in dataset has: — hand-labeled datasets
  - camera image / point cloud
  - hand pose
  - hand configuration
  - tactile information
  - grasp score

Pipeline

Camera Image → Pre-Processing → Neural Network → Post Processing → Best-rated grasp hypothesis

Generative approaches

no pre-/post processing

typical network architecture: few convolutional layers → some fully connected layers

Camera Image → conv → conv → FC → FC → Grasp pose

⊕
- straight forward, simple setup
- no complicated pre-/post processing
- no "restriction" in search/learning

⊖
- one grasp per object
- prone to averaging → no valid grasp, just "average of all labels"

Mitigation: Anchor boxes: scan NN over image in small patches
→ regress relative grasp pose offset at each anchor
→ classify if good grasp exists
→ take highest rated grasp

Discriminative approaches

- rate grasp according to sensor data
  → feed hypothesis pose and camera image to net

Camera Image → conv → conv → FC → FC → Grasp score

Hypothesis pose → FC

Heat map based approaches

input: depth image

- map images to images
- for each pixel in input: rate grasp quality
  → use img to img techniques from computervision
  → select pixel with highest predicted grasp score

Camera Img → conv → conv → deconv → deconv → Heat-Map

Sim2real reinforcement learning

Domain randomization approaches

- train model in different simulations
  → vary parameters like gravity, friction, noise, sensor offset, ...

→ assumption: policy will be robust, work in all simulated environments
  ⇝ reality should be somewhere in the randomized environments

Challenges with In-Hand Manipulation (Task: reposition object to different orientation inside hand)

- working in the real world
- high-dim. control (many more degrees of freedom)
- noisy and partial observation
- manipulating more than one object

Domain Randomization

- Visual appearance randomizations: camera pos / params
  lighting conditions
  different materials / textures

⊕
- learned policy is robust
- applicable for imperfect simulators
- approach to solve Sim2real problem
- no/little training needed

⊖
- computational power high
- transfer / re-use for different robot hands challenging
- learned policy too careful, non-optimal on target system