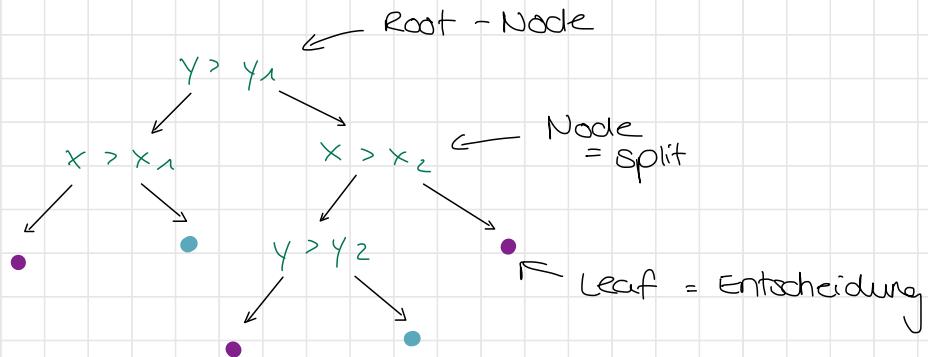


Moravec Paradox

Computer lernen relativ einfach "Erwachsenen-Skills" wie Intelligenztests, Dame spielen aber schwerl nicht "Kinder-Skills" wie Wahrnehmung, Mobilität

Entscheidungsäume

- iterativer Ansatz
- je Schritt eine neue Entscheidung



Training

globale Optimierung

= kleiner Baum mit reinsten Blättern

~ schwer implementierbar

Greedy Suche

- $\forall k$ Dimensionen:
 - \forall möglichen $(n-1)$ Trennlinien:
 - bestimme Unreinheit des Cuts \leftarrow impurity
- wählt Dim. mit geringster impurity als Schnittdimension
- Splitte Daten in Unterguppen
- Für beide Unterguppen: GOTO 1.

→ Iteration bis Stop-Kriterium erreicht
 z.B. max-depth
 min-impurity

Impurity

$$\text{Entropy } i(n) = - \sum_j P(w_j) \log_2(P(w_j)) , \quad 0 \leq i(n) \leq 1$$

$$\text{Klassif. fehler } i(n) = 1 - \max_j P(w_j)$$

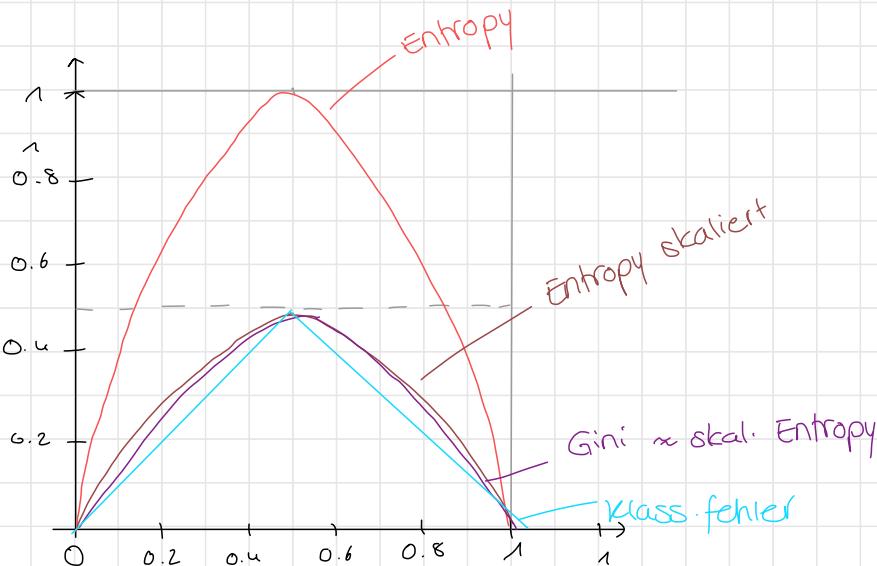
$\kappa = \text{Anteil der Elemente der häufigsten Klasse}$

Gini-Index

= Mittlerer Fehler wenn ein zufälliges Sample als Prediction gezogen wird

$$i(n) = \sum_i (P(w_i) \sum_{j \neq i} P(w_j))$$

$$= 1 - \sum_j P(w_j)^2$$



monothetischer Baum

- = Split nur entlang einer Dimension
- => Schlecht bei korrelierten Daten
- ~ polythetischer Baum = splitte entlang mehrere Dim.

Komplexität

Für n Datenpunkte, k Dimensionen

$$\mathcal{O}(k \cdot n \log(n)^2)$$

Regularisierung

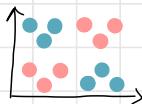
Early Stopping

- bei zu geringer Impurity - Reduktion
- mittels Cross Validation
- bei min. Zahl Samples im Leaf
- falls Tradeoff zw. Impurity + Baumtiefe erreicht
 $\Sigma_N : (N) + c \cdot \text{Stree}$

- ⊖ valley of draught: einzelner Schnitt verringert impurity nicht

→ zwei aber schon

→ early stopping funktioniert hier nicht



Pruning

Baum bis Blätter ohne impurity aufbauen
→ anhand von Kriterien zurückschneiden

Summary

- ⊕ einfach
schnelles Training

- ⊖ Overfittet leicht
korrelationen problematisch

Ensemble Methoden

Alternative zu Regulierung

Idee: mehrere Predictors kombinieren

Ansätze

Bagging = Bootstrapping Aggregation

- unab. Modelle parallel trainieren
 - Training auf zufälligem Subset der Daten
 - Prediction = Kombination der Models
zB average, median, voting, ...

Boosting

- Modelle sequentiell trainiert
→ Fokus auf falsche Predictions aus Vorgänger - Modell

Bias + Varianz

Datenpunkte x_1, \dots, x_n und Labels y_1, \dots, y_n

mit Aldoingung $y_i = f(x_i) + \varepsilon$

und Standardabweichung σ^2 = Bayesian error
= zugrundeliegender Fehler
= min. erreichbarer Fehler

Gesucht: Predictor $\hat{f}(x)$ mit min. loss $L = (y - \hat{f}(x))^2$

→ über k Samples Erwartungswert berechnen:

$$E[(y - \hat{f})^2] = \underbrace{E[\hat{f} - f]^2}_{\text{Bias}} + \underbrace{E[(\hat{f} - E[\hat{f}])^2]}_{\text{Varianz}} + \sigma^2 = \text{Streuung}$$

hoher Bias = Daten schlecht gefüllt
hohe Varianz = Modelle sehr uneinig bei Prediction

Random Forest

Bagging

- Trainingssets durch Ziehen mit Zurücklegen
- pro Set ein Entscheidungsbaum
- alle zusammen = Ensemble

Random Split Selection

- während Training nicht alle Features in jedem Node nutzen
- weniger "greedy"
- "unwichtige" Features benutzt + kombiniert zu wichtigen

Bias + Varianz

Einzelbaum:

$$\mathbb{E}[y_i] = \mu$$
$$\text{Var}[y_i] = \sigma^2$$

Random Forest:

$$y = \frac{1}{m} \sum_{i=1}^m y_i$$

$$\mathbb{E}[y] = \mathbb{E}\left[\frac{1}{m} \sum_i y_i\right] = \mu$$

$$\text{Var}[y] = \text{Var}\left[\frac{1}{m} \sum_i y_i\right] = \frac{1}{m^2} \text{Var}\left[\sum_i y_i\right]$$

→ $\stackrel{!}{=} \frac{1}{m} \sum_i \text{Var}[y_i] = \frac{1}{m} \sigma^2$

Voraussetzung:

unabh. Modelle → kann nicht beliebig viele unabh. trainieren

$$\sim \text{Var}[y] = \frac{1}{m} (1 - \rho) \sigma^2 + \rho \sigma^2$$

mit $\rho = \text{Korrelation}$
 \approx wie gleich sind Samples

Fehlerabschätzung

- je Baum nur Teil der Daten genutzt
 - ↳ Out-of-bag samples zur Generalisierungsfehlerschätzung

Feature Importance

- ↳ Interpretierbarkeit

Gini Importance

- Summe aller impurity reductions der Nodes mit Feature
 - ↳ über alle Nodes + Decision Trees
 - ↑ gewichtet über # samples im Node

Permutation Importance

- Out-of-bag samples: Werte für ein Feature zufällig permutieren
 - ↳ Prediction acc. davor vs. danach vergleichen
- ▷ schlecht bei korrelierten Features

Summary



- hohe Genauigkeit bei wenigen Params
- für kategorische und kontinuierliche Features möglich
- schnelles Training + Prediction
- invariant gegenüber Skalierung
- interpretierbar → Feature Importance



- mathem. nicht ausreichen verstanden + beschrieben
 - keine Garantien

Challenges

- Autom. Feature Generation
- Verbindung mit anderen ML- (KI-) Methoden

Anwendungen

Kosmische Strahlung

- Gamma - Strahlen vs. hadronische Strahlung
 - Gamma-Strahlen: Energie der Sonne von 10 Bio. Jahren in wenigen Sekunden
- Detektion erzeugt Bilddaten
→ daraus Features
→ Ensemble Methoden = gute Klass.

Brustkrebs

- 8 Features (Alter, Tumogröße,...)
 - Rechtszensierte Daten = geben nur Untergrenze der Lebensdauer an
- "Survival Trees" (Besonderer Decision Tree)
- Kleine Bäume
 - hohe Interpretierbarkeit (Feature Importances)

Proteininteraktion

- viele Features (Struktur, Sequenzen, Energie,... ≈ 600)

Mathe Basics

Vektoren & Matrizen

Tensor = multidim. Array

zB Video: 2 räumliche Dim. x, y (zB 100,100)
 1 zeitliche Dim. t (zB 1000 frames)
 1 Farbdim. c (zB 3 Farben)

$$\Rightarrow V \in \mathbb{R}^{t \times x \times y \times c} = \mathbb{R}^{1000 \times 100 \times 100 \times 3}$$

Matrix Multiplikation

$$C = AB, C_{ij} = \sum_k A_{ik} B_{kj}$$

$$m \times n \times n \times p = m \times p$$

- 1. Distributiv
- 2. Assoziativ
- 3. NICHT Kommut.
- 4. Transposition

$$A(B+C) = AB + AC$$

$$A(BC) = (AB)C$$

$$AB \neq BA$$

$$(AB)^T = B^T A^T$$

Hadamard Produkt = el. weise Matrixmult.

$$A \odot B = C$$

$$m \times n \odot m \times n = m \times n$$

$$\text{Inverse } A^{-1} A = I \leftarrow \text{Einheitsmatrix}$$

Lineare Gleichung

$$Ax = b \Leftrightarrow x = A^{-1}b, A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^m$$

→ hat Lsg falls b im linearen Bildraum von A

= notwend. Bed.: $n \geq m$

hinv. Bed.: $\exists m$ Spalten lin. unabh.

A^{-1} existiert falls $n=m$, also exakt eine Lsg.

Längen & Normen

- (i.A.) nicht-negativ: $f(x) = 0 \Rightarrow x = 0$
- Dreiecksungl.: $f(x+y) \leq f(x) + f(y)$
- Distrib.: $\forall x \in \mathbb{R}: f(cx) = |c| f(x)$

L^p Norm $\|x\|_p = \sqrt[p]{\sum_i |x_i|^p}$

L^2 Norm = Euklidische Norm $= \|x\|_2 = \sqrt{\sum_i |x_i|^2}$

oft: quadrierte L^2 Norm wegen Ableitung

∅ steigt um den Ursprung nur langsam

L^1 Norm $\|x\|_1 = \sum_i |x_i|$

∅ steigt linear um Ursprung \rightarrow bessere Unterscheidung zw. ∅ und ≠ 0

Max norm $\|x\|_\infty = \max_i |x_i|$

Frobenius norm (einer Matrix) $\|A\|_F = \sqrt{\sum_{i,j} |A_{i,j}|^2} = \sqrt{\text{Tr}(A A^T)}$

Skalarprodukt $x^T y = \|x\|_2 \|y\|_2 \cos \Theta$

Spezielle Matrizen & Vektoren

Symmetrische Matrix $A = A^T$

Einheitsvektoren $\|x\|_2 = 1$

Orthogonale Vektoren $x^T y = 0$

Orthonormale Vektoren = Orthogonale Einheitsvektoren

Orthogonalmatrix = Quadr. Matrix mit orthonormalen Zeilen, Spalten

$$A A^T = A^T A = I \Rightarrow A^{-1} = A^T$$

Eigenwerte & -vektoren

Eigenwertgleichung

$$A v = \lambda v$$

$\lambda_1, \dots, \lambda_n$
 $v^{(1)}, \dots, v^{(n)}$

Eigenwert
Eigenvektoren

$$\lambda = [\lambda_1, \dots, \lambda_n]^T$$

Eigenwerte = Verzerrung des Raums entlang der Eigenvektoren

$$v = [v^{(1)}, \dots, v^{(n)}]$$

$$A = V \text{diag}(\lambda) V^{-1}$$

Spur & Determinante

$$\text{Tr}(A) = \sum_{i,j} A_{i,j} = \text{Tr}(A^T)$$

$$\text{Tr}(ABC) = \text{Tr}(CAB) = \text{Tr}(BCA)$$

Determinante = Produkt der Eigenwerte

$$\det(A) = 0 \Leftrightarrow \text{Raum entlang } \geq 1 \text{ Dim kontrahiert}$$

= alle Infos entlang dieses EV gehen verloren

PCA Principle Component Analysis

zur Darstellung eines hochdim. Datenraums in niedrigere Dimensionen

Aufgabe

Kompression von m Punkten $\{x^{(1)}, \dots, x^{(m)}\} \in \mathbb{R}^n$ mit möglichst geringem Loss nach Rekonstruktion

Ansatz

1) Encodierung in Dim $\ell < n$ $x^{(i)} \in \mathbb{R}^n \rightarrow c^{(i)} \in \mathbb{R}^\ell$

$$\rightsquigarrow f(x) = c$$

2) Rekonstruktion $x \approx g(f(x))$

PCA

$g(c) = Dc$, $D \in \mathbb{R}^{n \times \ell}$ = lineare Abbildung
↑
Spalten orthogonal mit Norm 1

→ Ziel: minimiere $c^* = \arg \min_c \|x - g(c)\|_2^2$

$$\rightsquigarrow (x - g(c))^\top (x - g(c))$$

$$= x^\top x - x^\top g(c) - g(c)^\top x + g(c)^\top g(c)$$

$$= \underbrace{x^\top x}_{\text{const.}} - 2x^\top g(c) + g(c)^\top g(c)$$

bzgl. $c \rightarrow$ ignorieren

$$\Rightarrow \text{Lösung } c = D^\top x$$

$$\text{Rekonstruktion } r(x) = g(f(x)) = DD^\top x$$

Wahrscheinlichkeiten & Statistik

Uncertainty

Intrinsische Unsicherheit im System oder Experiment

Unvollst. Beobachtbarkeit eines deterministischen Systems

Unvollst. Modell Teile der Beobachtung nicht modelliert / beobachtet

Wahrscheinlichkeiten

Häufigkeitskonzept

- wiederholbares Experiment zB Würfeln
- Wkt als relative Häufigkeit

Bayes'sches Konzept

- Wkt einer bestimmten Beobachtung zB Krankheit

Zufallsvariablen & Verteilungen

Zufallsvariable = kann versch. Werte annehmen
diskret oder kontinuierlich

Wktverteilung = beschreibt Wkt dass ZV bestimmten Wert annimmt

diskrete ZV

$$\forall x \in X : 0 \leq P(x) \leq 1 ; \sum_{x \in X} P(x) = 1$$

$P(X=x)$ = diskrete Verteilung

Kontinuierliche ZV

$$\forall x \in X : p(x) \geq 0 , \quad \int p(x) dx = 1$$

Wkt dass x zw. a, b : $\int_a^b p(x) dx$

Marginalverteilung

Wktverteilung bzgl. Z zw. X, Y : $p(x, y)$

- Summenregel: $\forall x \in X : P(X=x) = \sum_y p(x, y)$
für X, Y diskret
- Kontin.: $p(x) = \int p(x, y) dy$

bedingte Wkt

$$P(Y=y | X=x) = \frac{P(Y=y, X=x)}{P(X=x)}$$

unabh. ZV

X, Y unabh. falls $\forall x \in X, y \in Y : p(x=x, y=y) = p(X=x) p(Y=y)$
 $\Leftrightarrow X \perp\!\!\! \perp Y$

Erwartungswert

$$\begin{aligned} E_{X \sim p}[f(x)] &= \sum_x P(x) f(x) && (\text{diskret}) \\ &= \int p(x) f(x) dx && (\text{kontin.}) \end{aligned}$$

Varianz

$$\text{Var}(f(x)) = E \left[(f(x) - E[f(x)])^2 \right]$$

Kovarianz Cov misst nur lineare abh.

$$\text{Cov}(f(x), g(y)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])]$$

Korrelation

$$\text{Cor}(f(x), g(y)) = \frac{\text{Cov}(f(x), g(y))}{\sqrt{\text{Var}(f(x)) \text{Var}(g(y))}}$$

Bayes Regel

$$P(x|y) = \frac{P(x) P(y|x)}{P(y)}$$

$$P(y) = \sum_x P(y|x) P(x)$$

Informationstheorie

= Quantif. von Infos

Informationsinhalt von $X=x$: $I(x) = -\log P(x)$

Shannon Entropy $H(P) = -\mathbb{E}_{x \sim P} [\log P(x)]$

Kullback-Leibler Divergenz

$P(x), Q(x)$ = Wktrverteilung über gleiche zw. X

\rightarrow KL-Div. misst Grad der Unterschiede zw. P, Q

$$D_{KL}(P||Q) = \mathbb{E}_{x \sim P} \left[\log \frac{P(x)}{Q(x)} \right]$$

- $D_{KL} = 0$ falls $P(x), Q(x)$ gleich
- $D_{KL} \geq 0$
- D_{KL} NICHT symm. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$

Numerische Optimierung

Gradient Descent

$$y = f(x) \rightarrow f'(x) = \frac{dy}{dx}$$

→ optimiere in Richtung des Gradienten

$$x' = x - \varepsilon \nabla_x f(x)$$

↑ ↗ Abl. bzgl. alle Komponenten von x
Lernrate (Nabla-Operator)

- Problem mit lokalen Minima
- geht nicht entlang der sensibelsten Richtung
zB bei länglichen Mulden

Machine Learning Basics

Machine learning = Computer lernt aus Erfahrungen \in respektive einem Task T und einer Performance-Metrik P falls sich T gemessen mit P durch \in verbessert

Tasks

Klassifikation Inputvektoren abbilden auf k Klassen

$$f: \mathbb{R}^n \rightarrow \{1, \dots, k\}$$

zB ImageNet

Regression Werte vorhersagen

$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$

zB Immobilienpreise vorhersagen

Transkription unstrukturierte Eingabe \rightarrow Text

zB Speech to text

Translation = Übersetzung

Outlier Detection zB Betrugserkennung

Synthese + Sampling zB Bildsynthese

Performance Metrik

- für quantitative Analyse der Performance eines ML Algos

Klassifik. / Transkription: Accuracy $\frac{\# \text{ korrekt}}{\# \text{ total}}$

Regression: Fehlertyp, z.B. L1, L2

Testdaten

- vom Algo noch nicht gesehen
 - aus gleicher Verteilung wie Ziellösung
- ↪ P abh. vom Task + Anwendung

Erfahrung E

Dataset $X \in \mathbb{R}^{m \times n}$

- m Datenpunkte mit je n Features als Zeilen

supervised

- jedes m hat Ziellabel: $p(y|x)$

unsupervised

- keine Ziellabels: $p(x)$

↪ Unterscheidung nicht eindeutig!

- unsupervised $p(x)$ als supervised:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

- sup. als unsup.:

$$p(y|x) = \frac{p(x,y)}{\sum_{y'} p(x,y')}$$

- Manche gehören zu keiner Klasse
zB Reinforcement Learning

Linear Regression

geg... - m Samples mit n Features: $x \in \mathbb{R}^n$
 - Labels $y \in \mathbb{R}$

Modell: $\hat{y} = w^T x$ mit $w \in \mathbb{R}^n$

Ziel: finde optimale Werte für w , sodass $\hat{y} \approx y$

Perf. metrik: MSE auf Test-Set

$$MSE_{\text{test}} = \frac{1}{m} \| \hat{y}_{\text{test}} - y_{\text{test}} \|_2^2$$

→ Lösung durch Fehlermin.: $\nabla_w MSE_{\text{train}} = 0$
auf Training-Set

$$w = (X_{\text{train}}^T - X_{\text{train}})^{-1} X_{\text{train}}^T y_{\text{train}}$$

Bias

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

$$\hookrightarrow x = (x_1, \dots, x_n, 1)$$

$$w = (w_1, \dots, w_n, b)$$

Evaluierung

Generalisierung

= wie gut bin ich auf ungesiehenen Daten?

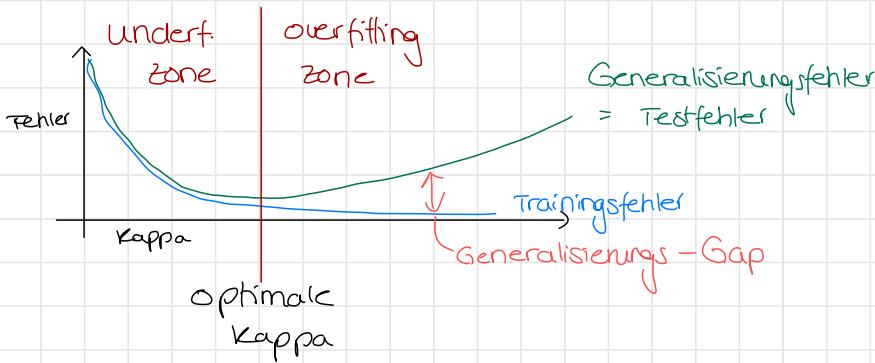
↪ Trainingsziel: min. Fehler auf Trainingsdaten
Gesamtziel: aus gleicher Testdaten Verteilung

Underfitting = Großer Trainingsfehler

Overfitting = Große Differenz zw. Trainings-, Testfehler

→ Kontrollierbar über Modellkapazität

- geringe Kappa = wenige Params → eher underfitting
- hohe Kappa = viele → eher overfitting



- hoher Bias: Underfitting
- hohe Varianz: Overfitting

Nicht-parametrische Modelle

- Keine Limits der Kappa
→ iterativ # Params anpassen

zB Nearest Neighbor Regression

Regularisierung = Methoden zur Limitierung der Kapazität

Idee: Model muss selbst entscheiden wie viele Params
→ bestrafen bei zu vielen

weight decay

Loss ändern zu: $J(\omega) = \text{MSE}_{\text{train}}(\omega) + \lambda \omega^T \omega$

→ geringere Gewichte bevorzugt
⇒ flachere Funktionen
+ einfache Modelle

Hyperparameter

- i. A. durch Regularisierungsterm keine geschlossene Lösung
- brauche numerische Optimierungen
- haben eigene Params = Hyperparams

zB Lernrate ε bei grad. desc.

Optimierung

- automatisch beim Training
- brauche extra Daten dafür = validation data

⇒ 3 Sets: Training, Validation, Test

Cross Validation

- Wissenschaft: Problem: zu wenige Daten (100 - 1000)
→ ungenauer Fehler
 - wiederholtes Training auf k nicht-überlappenden Daten
- zB 1/10 Test + 9/10 Trainingsdaten ⇒ 10 Iterationen

Maximum Likelihood

Ziel: finde Parameter Θ die Daten am plausibelsten erklären

Unsupervised

- m Datenpunkte $\mathcal{X} = \{x_1, \dots, x_m\}$ mit unbekannter Verteilung $p_{\text{data}}(x)$

$$\rightarrow \Theta_{\text{ML}} = \underset{\Theta}{\operatorname{argmax}} \prod_{i=1}^m p_{\text{model}}(x_i, \Theta)$$

$$= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^m \log p_{\text{model}}(x_i, \Theta)$$

→ entspricht der Minimierung des Fehlers

Supervised

Ziel: finde $P(y | x; \Theta)$

$$\hookrightarrow \Theta_{\text{ML}} = \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^m \log P(y^i | x^i; \Theta)$$

$$\stackrel{!}{=} \text{minimieren von } \text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^m \| \hat{y}^i - y^i \|_2^2$$

⇒ für $m \rightarrow \infty$ Datenpunkte wird optimales Modell gefunden

Supervised Learning Methoden

- Log.-Regress.
- SVMs
- Nearest Neighbor
- Decision Trees

Logistische Regression

Annahme: 2 Klassen

→ Voraussage zw. 0 und 1

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

Modell: $p(y=1|x; \theta) = \sigma(\theta^T x)$

→ keine geschlossene Lsg \Rightarrow Gradient Descent

Support Vector Machines

Idee: Binäre Klass. durch Trennung der Daten mittels Hyperebene

Kernel Trick: für nicht-lin. sep. Daten

1) Skalarprod. umschreiben $w^T x + b = b + \sum_{i=1}^m \alpha_i x^T x^i$

2) Nicht-lin. Transformation in höhere Dim. $d_2 > d_1$

$$\phi: \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}, x \mapsto \phi(x)$$

3) Kernel: $k(x, x^i) = \phi(x) \cdot \phi(x^i)$

$$\Rightarrow f(x) = b + \sum_i \alpha_i k(x, x^i)$$

- + · Linear in ϕ und $\alpha \Rightarrow$ lineare Optimierung möglich
- Evaluation der Kernelfunktion einfacher als ϕ berechnen
- · lineare Rechenzeit ($O(m)$, $m = \# \text{ samples}$)

Nearest Neighbor

- unendliche Kapazität
- nicht-parametrisch = keine fixe # Params
- · Feature-Importances nicht berücksichtigt

Unsupervised Learning methods

- PCA (Dim. reduktion) $\xrightarrow{\text{1D}} \rightarrow \xrightarrow{\text{2D}}$
- K-means Clustering
 - · kein Einfluss auf Feature-Importances (Learning Task)
 - · k muss festgelegt werden

Curse of Dimensionality

- je mehr Features (= Dimensionen), desto mehr Punkte für Raumabdeckung nötig
 $\# \text{dim} = 100 \rightarrow 2^{100}$ nur für Extremwerte \emptyset (= Ecken)

Manifold Learning

i. A. nur kleiner Raum enthält sinnvolle Werte
zB Bilder vs. zufällige Pixel

→ folgen zugrundeliegenden Mannigfaltigkeiten

Bewertungsmethoden

Konfusionsmatrix

Predicted

		Pos.	Neg.
Actual	Pos.	TP	FN
	Neg	FP	TN

→ ungeeignet bei ungleichen Labels in den Daten

Sensitivität = Recall = TP-Rate

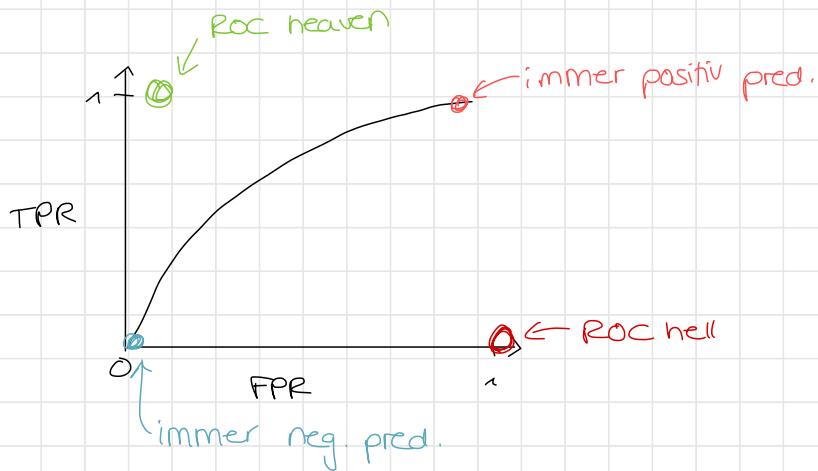
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

False - Pos. - Rate = Fall - Out

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

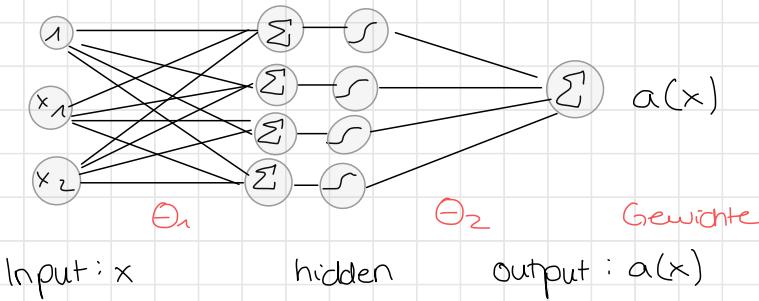
ROC curve

- TPR über FPR auftragen



Neuronale Netze

zur Lösung nicht-linearer Probleme



$$x \rightarrow h_1(x) = \Theta_1^T x \rightarrow a_1(x) = \sigma(h_1(x)) \rightarrow a_2(x) = \Theta_2^T a_1(x)$$

\uparrow
Aktivierungsfunktion

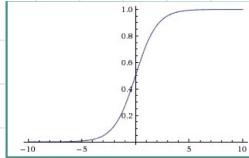
Aktivierungsfunktion

- nicht-linear \rightarrow sonst nur lineare Probleme lösbar
- differenzierbar

Sigmoid:

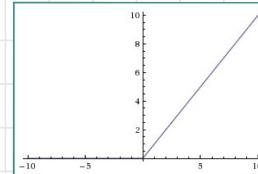
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$



Relu:

$$\text{relu}(x) = \max(0, x)$$



(+) kein van. gradient

(-) vanishing gradient

(-) dying Relu

- Leaky Relu = $\max(\alpha x, x)$ $\alpha \ll 1$ (z.B. 0.1)
oder trainiertes α
- tanh

=> beste durch trial & error finden

Hidden Layers

- degrees of freedom: Anzahl, Größe, Konnektivität der Layer
- > je tiefer desto besser, aber:
 - overfitting
 - mehr Trainingsschritte nötig
 - schwerer zu trainieren

Output Layer

Möglichkeiten:

- Linearer Output + MSE Loss
- Sigmoid Output + Neg. log. Likelihood (binäre Klass.)
- Softmax Output + -"-" (> 2 Klassen)

Softmax

$$\text{Output } z = w^T \cdot h + b \rightarrow \text{softmax}(z)_j = \frac{e^{z_j}}{\sum_i e^{z_i}}$$

Training: Backpropagation

Idee: Fehler durchs Netz propagieren bis zum Input

-> KETTENREGEL

Forward-Pass: $x \rightarrow h = \Theta_1^T x \rightarrow a_1 = \sigma(h) \rightarrow a_2 = \Theta_2^T a_1$

$$\text{LOSS: } J = \frac{1}{2m} \sum_{i=1}^m (a_i^i - y_i^i)^2$$

\nwarrow samples \swarrow Label

Updates:

$$\theta_2 = \theta_2 - \alpha \frac{\partial J}{\partial \theta_2}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$$

=> Kettenregel zum Finden der Ableitungen

$$* \frac{\partial J}{\partial \theta_2} = \frac{\partial J}{\partial a_2} \cdot \frac{\partial a_2}{\partial \theta_2}$$

$$* \frac{\partial J}{\partial \theta_1} = \frac{\partial J}{\partial a_2} \cdot \frac{\partial a_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial h} \cdot \frac{\partial h}{\partial \theta_1}$$

Regularisierung

zur besseren Generalisierung

Weight Decay

- bestrafe große Parameter
→ Zusatz zum Loss

$$\tilde{J}(\theta; x, y) = J(\theta; x, y) + \lambda R(\theta)$$

alle Gewichte

zB L2-Regularisierung = "ridge regression"
 $R(\theta) = \frac{1}{2} \|\theta\|_2^2$

L1-Regularisierung = Lasso regression
 $R(\theta) = \|\theta\|_1 = \sum_i |\theta_i|$

Ergebnis: Sparsification

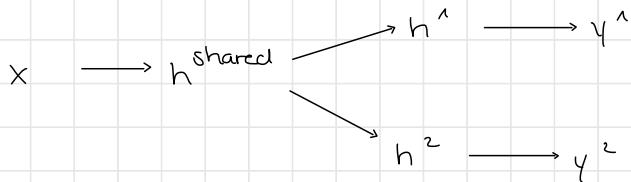
- manche Gewichte werden 0
- Selektiert Features

Data augmentation

- Bilder drehen, ...
- Noise auf Input Daten / hidden layers

Multitask learning

- mehrere Labels + Losses aber gleicher Input
=> lerne gemeinsame Zwischenrepräsentation



Early stopping

- beende Training falls Validation Loss nicht mehr besser wird
→ Patience Hyperparam

Parameter sharing

zB bei Conv Layers

- weniger Params insgesamt → wenige Transf. auf vielen Attributen

Bugging, Ensemble Methoden

siehe oben

Dropout

- beim Training zufällig Verbindungen kappen (= Maske)
 - je Batch andere Maske
 - zwingt alle Neuronen zum Lernen → keine "Hot-Paths"

Adversarial Training

- = gezielt Daten die schwer zu trennen sind erzeugen
→ inkl. Labels als Adversarial Trainingsset ins Netz
- = Daten in Richtung Loss ändern
→ generalisiert besser im Bereich um die Trainingsdaten

Materialsimulation

Problem Mikroskopie:

- Moleküle nur isoliert betrachtbar
 - nicht in natürlicher Umgebung / Arbeitsweise
 - nur statisch
- Ziel: Computersimulationen

Nanoskalierte Simulationen

statisch

nicht-quanten mechanisch

- System: N Atomkerne mit Pos. R_I , Ladung Z_I
 n Elektronen mit Pos. r_i
- Energie (und dadurch Kräfte, Gradienten, Bewegung) beschreiben durch:

1. kinetische Energie der Elektronen
2. Atomkerne
3. Wechselwirkung zw. Elektronen
4. Atomkernen
5. El. + Atomkernen

} Summieren

quanten mechanisch

· Schrödinger-Gleichung + Born-Oppenheimer-Approx.
↓
Atomkerne bewegen sich nicht

→ Lsg über Eigenwertgleichung der Elektronen

zB mit DFT = Density Functional Theory

- skaliert mit # Elektronen
- > numerische Lsg. für viele Elektronen
= große Moleküle unmöglich

Kraftfelder

Annahme: Elektronen passen sich instantan an Pos. der Atomkerne an

→ Energie des Ges. Systems abh. von

- Bindungsängen
- Bindungswinkel
- Torsionswinkel
- Wechselwirkungen (Van-der-Waals, Abstösung)
- elektrost. Wechselwirkungen

dynamisch

- Molekulardynamik basierend auf z. Newton'schen Gesetz

$$F_i = m \cdot \vec{a}_i = \nabla_{x_i} E$$

→ Iterative Lsg mit Schrittweite Δt

Lsg.

1. Atomkoordinaten auf Ges.-Energie + Kräfte mappen
 - Quantenmech. / Kraftfelder
2. Dynamik vorhersagen
 - numerisch + iterativ
 - evaluieren der Kräfte, Beschleunigungen, Temp., Druck

Machine Learning

Schritt 1 durch gelernte Potentiale ersetzen

→ schneller

→ größere Systeme simulierbar

Problem: Mapping von Koord. auf Energie + Kräfte

Lösung: Zerlegen in Teilprobleme

Neuronales Netz

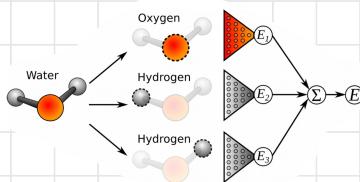
- Idee: 1 NN pro Atom

→ lernt Potential prediction in Umgebungs Kontext

→ je Atom prediction der Energie

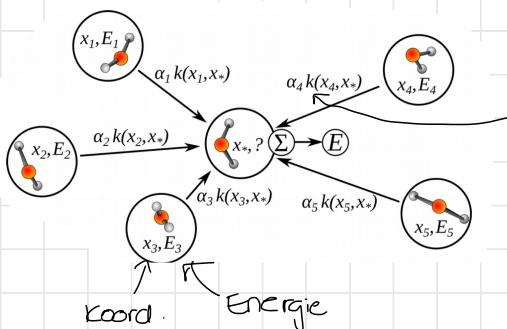
→ am Ende summieren

Training: viele Moleküle + entspr. Energie als Label



Kernel - basiert

- skaliert mit # Trainingssamples



kernelmatrix beschreibt Ähnlichkeit
zu Trainingsdaten &
Querysystem

Datasets

Idee: erzeuge viele molekulare Geometrien
+ berechne Eigenschaften mit Quantenmechanik
= Labels

Erzeugung

Vibrationsbasiert:

- berechne Equilibriumgeometrie (= alles im Gleichgewicht)
- berechne Eigenmodi, -schwingungen ↗
- verändere Ausgangsmolekül mittels ↗

⊖ Torsionen schlecht gesampelt

Trajektorienbasiert

- berechne Trajektorien bei best. Temperatur
- Snapshots in zeitl. Abständen

⊖ zeitl. nahe beieinander - sehr ähnlich

Beispiele

- QM_X = alle chem. relevanten Moleküle mit X Atomen
(H nicht mitgezählt)
+ DFT Eigenschaften
- ZINC = alle kommerziellen chem. Substanzen

Benchmarks:

- ANI-1 : basierend auf QM9

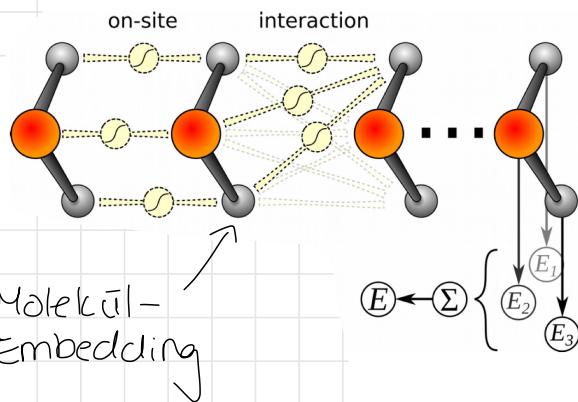
Modelle

Active Learning

- "zufällige" Samples ins Netz
→ falls falsch /unsicher : retraining damit
⇒ verschiebt Unsicherheiten

Beispiele

- ANI-1 = Ansatz wie oben
(1 NN pro Atom + Summe)
- TensorMol-0.1 = gelernte Energien + Kraftfelder
→ Partialladungen des Gesamtsystems
= globalere Wechselwirkungen
- SchNet
 - Training mit Energie + Gradienten
≈ Regularisierung



Interaction Layer:

- lernt Wechselwirkungen zw. Atomen abh.
von Distanz
- = Convolution mit Nachbaratomen

Convolutional Neural Networks

- Speziell für Grid-Daten (1D, 2D, 3D, ...)
zB Bilder, Video

Basis:

- Faltung: $O(i,j) = (\mathbf{K} * \mathbf{I})(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$

\downarrow Input
 ↓
 Output Kernel

- weight sharing → Faltung über alle $m \times n$ mit einem kernel
→ weniger Operationen, weniger Params

- hierarchisches Feature-Learning:

- versch. Layers lernen versch. Features
- frühere Layers: Features auf feiner Auflösung
später
grober
= globaler Kontext

Dimensionen

Convolutions

$$\text{Input: } H^{L-1} \times W^{L-1} \times n_C^{L-1}$$

$$\text{Output: } H^L \times W^L \times n_C^L \leftarrow \# \text{Filter in Layer L}$$

$$H^L = \left\lceil \frac{H^{L-1} + 2p^L - f^L}{S^L} \right\rceil \quad \begin{array}{l} \text{Padding} \\ \downarrow \\ S^L \end{array} \quad \begin{array}{l} \text{Filter} \\ \swarrow \\ \text{Stride} \end{array} \quad \text{analog für } W$$

Pooling

$$\text{Input: } H^{L-1} \times W^{L-1} \times n_C^{L-1}$$

$$\text{Output: } H^L \times W^L \times n_C^L \quad \uparrow \text{bleibt gleich}$$

$$H^L = \left\lceil \frac{H^{L-1} - f^L}{S^L} + 1 \right\rceil \quad \text{analog für } H$$

Parameter

Layer L

$$n_c^L \times \underbrace{f^l \times f^l}_{\text{Filter}} \times n_c^{l-1} \uparrow \quad \# \text{Input Kanäle}$$

\uparrow
Filter

Padding

- verhindert Reduktion der Größe
- zB 0-padding

Pooling

Conv + Pooling = Kombi der Infos mehrerer Pixel in einen

- Max Pooling : max aller Werte im Kernel
- Average Pooling : avg - " -
- L_2 Pooling : L_2 Norm - " -
- ⋮

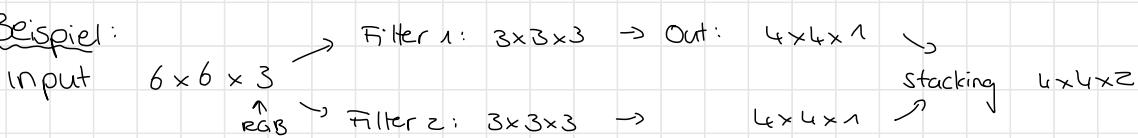
→ typisch : Größenreduktion (**Downsampling**) mit 3×3 , stride = 2

Conv Net

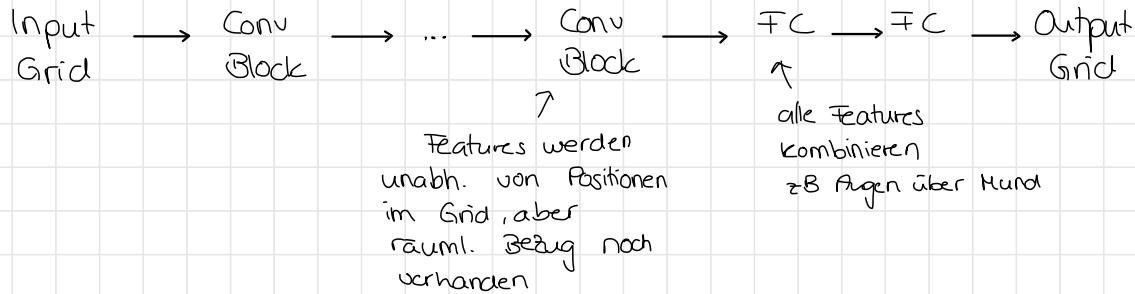
Convolutional Layer = Conv Block

Conv → Normalisierung → Nicht-Linearität → Pooling → Feature Maps

Beispiel:



Conv Net



Data Augmentation

CNNs nicht invariant ggüber
kaum Rotationen
Skalierung, Translation

→ Data Augmentation

- Deformation
- Skalierung
- Rotation
- Scherung
- ...

Anwendungsbeispiele

Medizin

medizinische Bilder

- CT, Röntgen
- MRT
- Ultraschall
- ...

→ 2D, 3D, zeitliche Daten, Oberflächendaten, ...

Brustkrebserkennung

Datensätze: GB (Training) + USA (Testing)

Modell

- 3 Modelle als Ensemble
- jeweils Risikoscore $\in [0, 1]$
- jeweils 3mal trainiert + average

Modell 1: Bildausschnitte

- BBox Prediction mit RetinaNet

↳ Focal Loss: Boxen mit hoher Objectness gehen weniger stark in den Loss ein als unsichere Boxen / Hintergrundboxen

Modell 2: unabh. Vorhersagen linke, rechte Brust

- Klassifikation der Regionen (binär)
- MobileNet V2 (ResNet Basis)

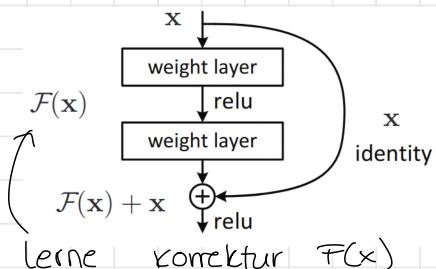
Modell 3:

- alle Bilder pro Patient kombinieren
- ResNet
- am Ende FC Layers

ResNet

- lerne Fehler durch Skip Connections

=> Input weiter ins Netz propagieren
=> tiefere Netze trainierbar



Ergebnis

ROC: besser als Erstgutachter, schlechter als 2. / Konsens

Quantenchemie

Idee: aus Atomposition kann auf Elektronendichte und Energie

- 1. Berechne Elektronendichte mit approx. Methode
- 2. Lerne Korrekturen zu Energie und Dichte

$$p(r) = p_0(r) + \Delta p[p_0(r)] \quad \left. \begin{array}{l} \text{Korrekturen nur auf Basis} \\ \text{von } p_0 \text{ lernbar!} \end{array} \right\}$$

$$E = E_0 + \Delta E[p_0(r)] \quad \left. \begin{array}{l} \text{approx.} \\ \text{Dichte} \end{array} \right\} \begin{array}{l} \uparrow \\ \text{mit Hartree-} \\ \text{Fock-Methode} \end{array} \quad \left. \begin{array}{l} \uparrow \\ \text{mit ML Methode} \end{array} \right\}$$

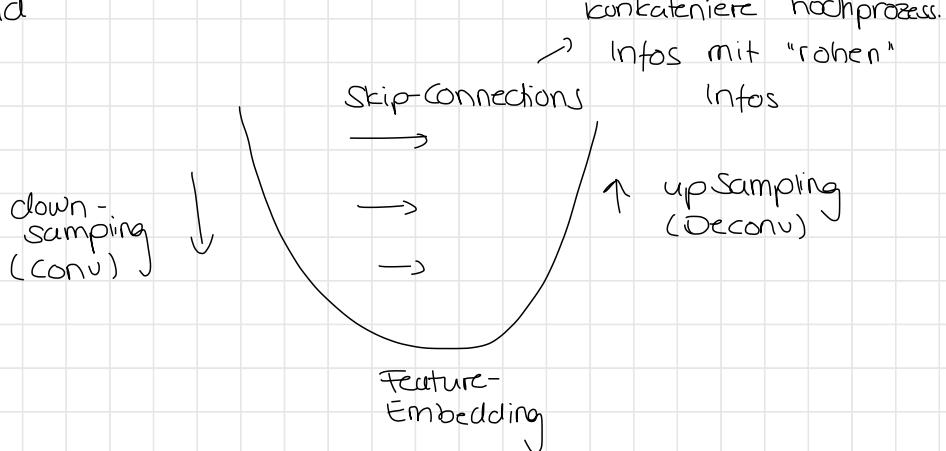
CNN

1. Mapping Dichte → Dichtekorrektur
2. Dichte → Energiekorrektur

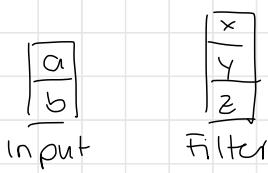
U-Net

Grid → Grid

Architektur:



Deconvolutions:



ax
ay
$az + bx$
by
bz

Output

→ U-Net lernt gemeinsame Repr. für ΔE , Δp

Ergebnis

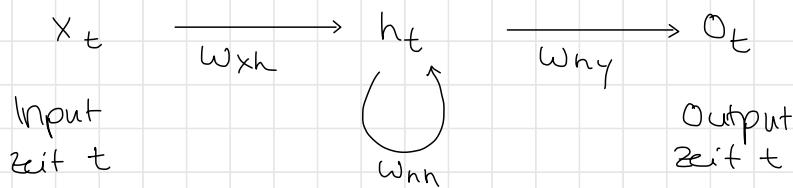
- Speedup Z Größenordnungen ggüber DFT

Recurrent Neural Networks

= Sequence Models

→ für sequentiellen Input/Output variabler Länge

• Infos können zu beliebigem Zeitschritt t auftreten
⇒ Parameter über alle Zeitschritte teilen



$$h_t = \tanh(b_x + \underbrace{w_{xh} x_t}_{\text{bias}} + \underbrace{w_{hh} h_{t-1}}_{\text{letzter hidden state}})$$

$$o_t = w_{yh} h_t + b_y$$

$$\text{Prediction } \hat{y}_t = \text{softmax}(o_t)$$

Deep RNNs

- mehrere hidden states übereinander
- MLPs als Zwischenschritte ($x \rightarrow h$ | $h \rightarrow h$ | $h \rightarrow o$)
- skip connections
- ...

Training

Backprop through time

- zu jedem Zeitschritt t : Loss mit o^t und Label y^t berechnen

→ Netz über die Zeit ausrollen und Losses über alle Zeitschritte zurück propagieren

- ① teuer, da nur sequentiell möglich

Teacher forcing (sequence 2 sequence)

- RNN bekommt beim Training statt h_{t-1} das Label y_{t-1} als Input

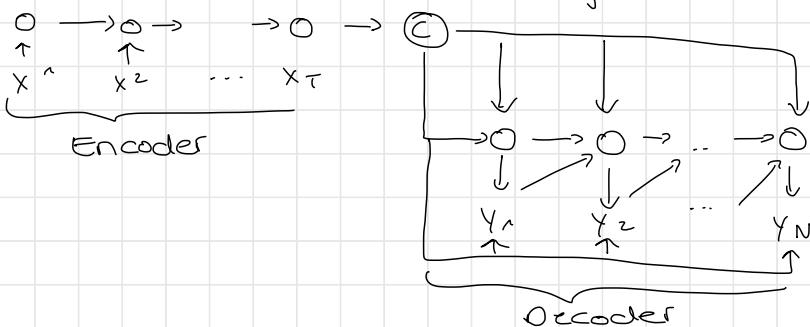
⇒ Zeitschritte unabh. + parallel trainierbar

- ② Label y_{t-1} hat ggf. nicht die relevanten Infos

Encoder - Decoder = Sequence 2 Sequence

→ gut falls $\text{len}(\text{Input}) \neq \text{len}(\text{Output})$

⇒ lerne Repräsentation des ges. Inputs = Kontext $C = h(T)$



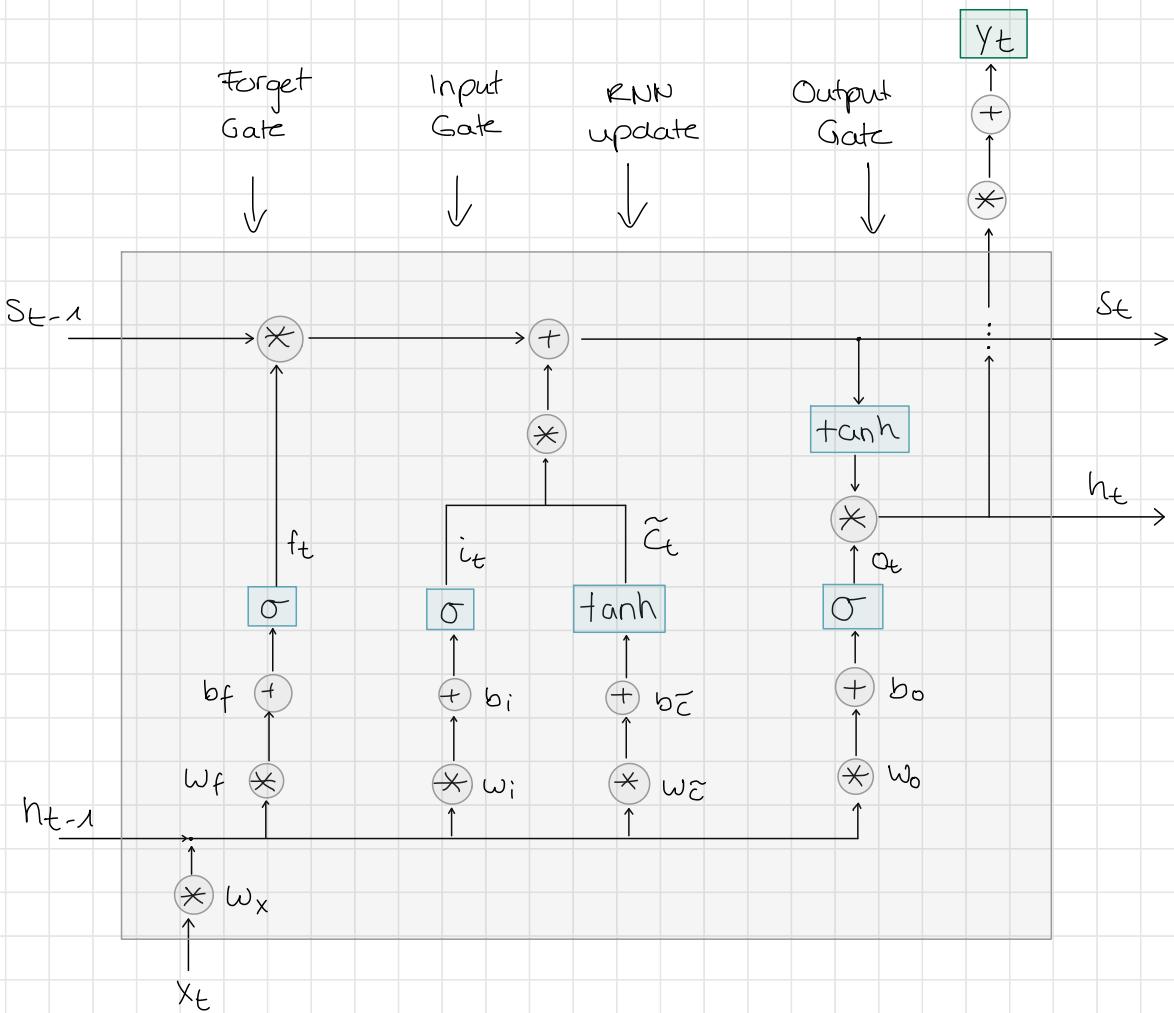
y_1, \dots, y_N durch entfalten von C generieren

$\Rightarrow y_t$ abh. von C und h_t
 h_t
 C, h_{t-1}, y_{t-1}

⊖ Kontext hat fixe Länge
→ je nach Input Länge: C lossy

LSTM

Idee: kontrolliere Informationsfluss durch Gates



S = Gedächtnis

Forget Gate: wie viel altes Gedächtnis behalten?

Input Gate: wie viel vom neuen Input merken?

State update (von hidden state): kombiniert Memory, Input, h_{t-1}

(+) durch Addition weniger exploding Van. Gradient

Attention

Problem S2S: alle Infos in einem Vektor, kein
Zurückschauen

→ Idee: lerne je Zeitschritt welcher Teil des Inputs
für Decoding wichtig ist

⇒ lernt wie Output zur Zeit t vom Input abhängt

→ zusätzlicher Kontextvektor c je Schritt

$$c_t = \sum_{j=1}^{T_x} \alpha_{i;j} h_j \quad \text{- linearkomb. aller hidden states während encoding}$$

$$\alpha_{i;j} = \frac{\exp(e_{i;j})}{\sum_{k=1}^{T_x} \exp(e_{i;k})}$$

$$e_{i;j} = v_a^T \tanh(w_a s_{i-1} + u_a h_j)$$

↑ ↑ ↑
lernbare Gewichte

Transformer Modell

- keine RNN-Teile, nur Attention
- parallelisierbar => schneller

zB GPT-n

Anwendungen

Medizin: EKG

- Klassifizierung in gesund + 4 Krankheiten

Modell:



Teilchenphysik: CERN

- Simulationen von Experimenten als Trainingsdaten
- Ziel: schnelles Filtern relevanter Ereignisse

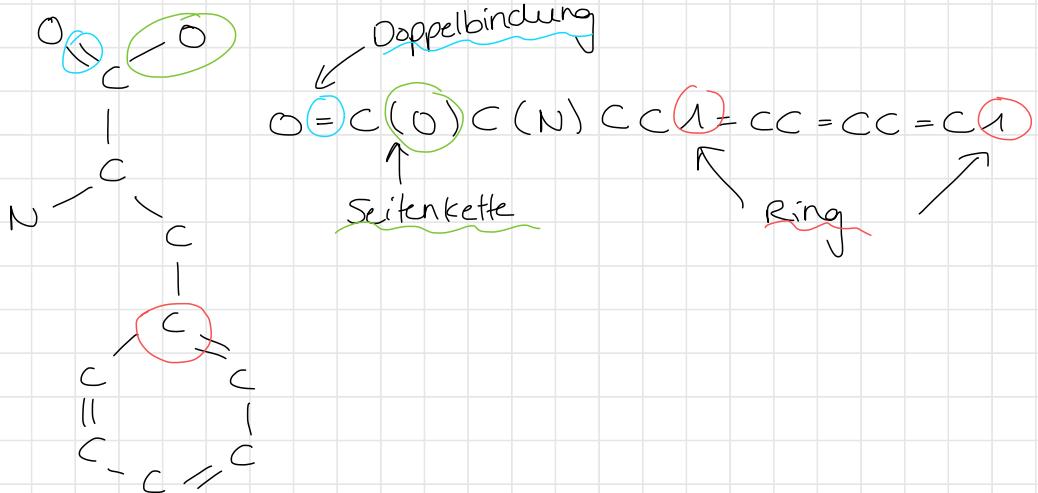
Chemie

- Vorhersage chem. Reaktionen
- Transformer Modell
- Input: SMILES Darstellung der Moleküle

Graph - Neural Networks

SMILES = simplified molecular input line-entry system

Molekül als String darstellen



- (+) · lesbar
- beliebige Moleküle darstellbar
- relativ einfach

- (-) · nicht robust \rightarrow nur bedingt einsetzbar für ML Modelle
- sehr viele invalide strings
- \hookrightarrow kleine Änderung = größerer Effekt (anderes Molekül?)

Fingerprints

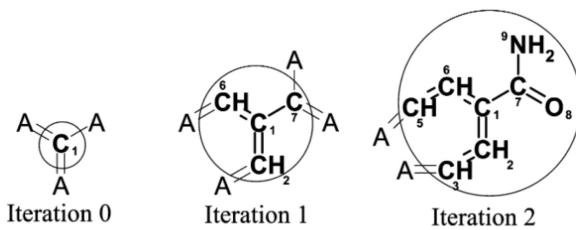
- Repr. als Bitvektor
- \rightarrow jedes Bit beschreibt ob best. Subgraph vorhanden

Problem:

- viele mögl. Subgraphen müssen auf fixe Länge gemappt werden

Lösung: Extended - Connectivity Fingerprints

- Iterativs generieren molekul. Repr. die so eindeutig wie möglich sind
- jedes Atom 1 Identifier
- update Identifier durch Aggregation mit 1-NN
- Iterieren bis max. Detaillevel erreicht



Atomidentifier

= Atomeigenschaften

- # Nachbaratome
- # Wasserstoffatome
- Masse
- Ladung
- ...

}

Hashing auf 32 Bit int

Iteration

- Aggreg. der Nachbaridentifier + Bindungstypen

⇒ Falten in Binären Bitvektor fixer Länge (zB 1024)

⊖ Bit-Collisions

Graph Convolutions

Message Passing

Init: Nodes = x_v = Eigenschaften von Atom v

Edges = e_{vw} = Verbindung Atom $v \leftrightarrow w$

hidden states :

- node vectors h_v^t , $h_v^o = x_v$

Message Passing: Akkumuliert Nachbarinfos im Node

$m_v^t = \sum_{w \in N(v)} m_w^t$ = Summe der Nachrichten
der Nachbarn von v

$m_w^t = f_{\theta_m, t}^m(h_w^t, e_{vw})$ = Message, abh. von
verbindung $v \leftrightarrow w$,
zustand von Nachbar w

$h_v^{t+1} = f_{\theta_u, t}^u(h_v^t, m_v^t)$ = State update $t \rightarrow t+1$

Final readout:

$g = R(\{h_v^T | v \in G\})$ = Akkumulation aller hidden
states zu finaler Zeit T

Embedding in Vektor fixer Länge

Prediction:

$\hat{y} = NN(g) \leftarrow$ Embedding in NN werfen

Beispiele

Duvenaud (2015)

- $M(h_v, h_w, e_{vw}) = (h_w, e_{vw})$ (concat)
- $F^u(h_v^t, m_v^t) = \sigma(H_t^{\deg(v)} m_v^t)$
 \nwarrow Gewichtsmatrix
- $R = \sum_{v,t} \text{softmax}(w_t h_v^t)$
 \nwarrow skip connections

Gated-GNNs

- $M^t(h_v^t, h_w^t, e_{vw}) = A_{e_{vw}} h_w^t$
 \nwarrow Gewichtsmatrix
- $F^u = \text{GRU}(h_v^t, m_v^t)$
- $R = \sum_{v \in V} \sigma(i(h_v^t, h_v^t)) \odot (j(h_v^t))$
 \nwarrow NNS \nearrow

Google MPNN (state of the art)

- $M^t(h_v^t, h_w^t, e_{vw}) = A(e_{vw}) h_w^t$
- $U_t = \text{GRU}(h_v^t, m_v^{t+1})$
- $R = \text{set2set}(\{h_v^t \mid v \in G\})$
 \nwarrow Verbindung ALLER Atome zueinander

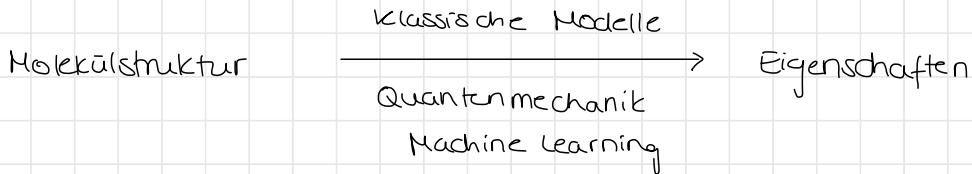
Benchmarking

- Standard: QM9
- PubChem QC
- versch. Toxikologie
- Genüche
- ...

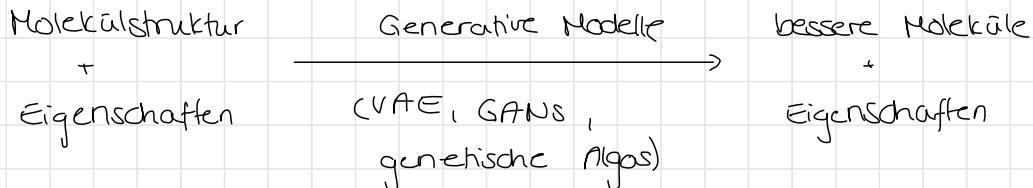
Generative Modelle

Inverses Design

Bisher:

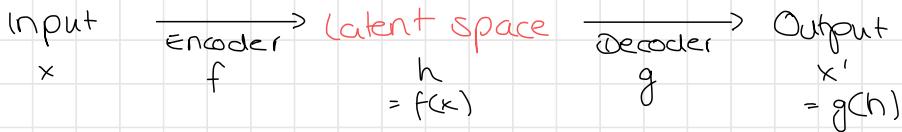


Invers:



Autoencoder

- Sanduhrmodell:



⇒ lerne niedrigdim. interne Repräsentation

Ziel: $x' \approx g(f(x))$

⇒ Loss $L(x, x')$ minimieren

· g, f linear: lernt PCA

Sparse Autoencoder

Ziel: h möglichst wenige Dimensionen

$$\rightarrow L(x, x') + \lambda R(h)$$

bestraft hohe Dim.

Kontraktive Autoencoder

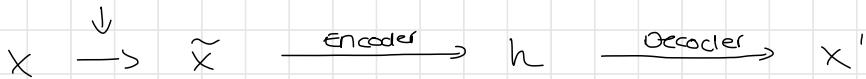
Ahnliche $x \rightarrow$ ähnlich auch im Latent space
(nahe beieinander)

$$\rightarrow L(x, x') + \lambda R(h, x)$$

$$= L(x, x') + \lambda \sum_i \| \nabla_{x_i} h_i \|^2$$

Beispiel: Denoising

Ziel: lerne Störungsterm
verrauschen



=> Loss zw. x' und x berechnen

=> lernt denoising

- ⊖ viele Punkte im Latent Space ohne Bedeutung

Variational Autoencoder

Latent Space probabilistisch modellieren

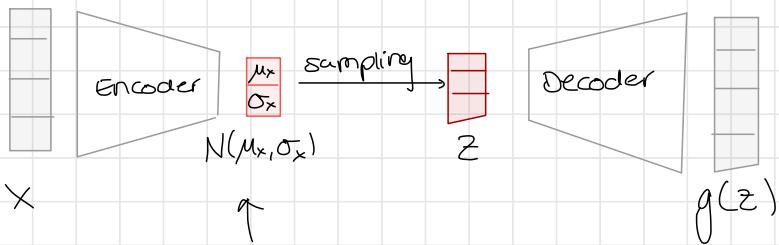
Input \rightarrow Latent Distribution \rightarrow Sample \rightarrow Output

x

$$p(z|x)$$

$$z \sim p(z|x)$$

$$g(z)$$



lerne Params der
Normalverteilung

$$\text{Loss} = \|x - g(z)\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, 1)]$$

↑
KL-Divergence vergleicht Verteilungen

ziel: Verteilung soll möglichst nah an $\mu=0, \sigma=1$ sein
 \Rightarrow lerne Zusammenhänge im latent space

\Rightarrow erlaubt sampling + decoding beliebiger Punkte im latent space

Beispiel Moleküle

- sampling im latent space: finde neue Moleküle

Eigenschaften

Idee: sortiere latent space nach Moleküleig.

\Rightarrow gezieltes sampling für bestimmte Eig. möglich

\rightarrow trainiere weitere NNs, die auf dem latent space Eigenschaften vorhersagen

\rightarrow Multi-task learning der NNs + VAE

GAN Generative Adversarial Networks

2 Modelle:

- Generator: generiert Daten aus Noise $x = g(z, \Theta_g)$
- Diskriminatator: entscheidet ob x echt oder generiert $d(x, \Theta_d)$

→ Training:

$$g^* = \arg \min_g \max_d \left[E_{x \sim p_{\text{data}}} [\log(d(x))] + E_{x \sim p_{\text{model}}} [\log(1 - d(x))] \right]$$

Bayes Optimierung & Gauß'sche Prozesse

Autonome Experimente

Problem manueller Experimente:

- alle Kombis ausprobieren zu teuer
= Curse of Dimensionality
- viele Kombis sind ähnlich
→ gezielter Suchen?

Bayes Optimierung

- Iterativ Experimente durchführen

↳ jeweils den Datenpunkt mit max. Infozugewinn finden

Voraussetzungen:

- Unsicherheit vorhersagen / abschätzen
- Acquisition Function (= wie nützlich ist welcher Punkt?)

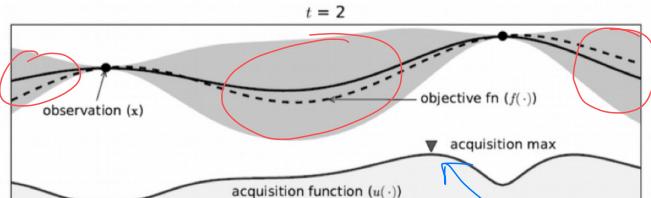
Algorithmus

1. Training / Retraining

2. Evaluiere Acquisition Funktion

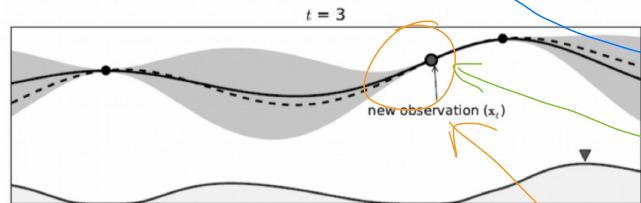
→ finde Maximum = informatiuester Punkt

3. Evaluiere diesen neuen Punkt



Zielfunktion

Bereiche mit großer Unsicherheit



finde Maximum der Acq. function
→ Werte Punkt in Zielfunktion aus

⇒ Bereich der Unsicherheit verkleinert

Unsicherheit vorhersagen

Gauß'scher Prozess

= Wkt.-Verteilung über (kontinuierliche) Funktionen

→ beschrieben durch Mittelwert & StdDev (= Unsicherheit)

Grundlage: multidim. Normalverteilung

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

↑
Kovarianzen
zw. Dimensionen

Standardabweichungen

Schnitt multidim. Normalverteilungen = Normalverteilungen

$$\rightarrow p(x_1 | x_2) = N(x_1 | \mu_{1|2}, \Sigma_{1|2})$$

$$\text{mit } \mu_{1|2} = \mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (x_2 - \mu_2)$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21}$$

=> Beschreibe Datenpunkte als multidim. Normalverteilung

ZB 3 Punkte $(x_1, f_1) \quad (x_2, f_2) \quad (x_3, f_3)$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \sim \left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \right)$$

mit $k_{ij} = e^{-\|x_i - x_j\|^2}$ $\Rightarrow \begin{cases} 0, \|x_i - x_j\| \rightarrow \infty \\ 1, \|x_i - x_j\| \rightarrow 0 \end{cases}$

Funktion, die Kovarianz der Punkte abh.
vom Abstand beschreibt

=> neuer Datpunkt x^* hinzufügen durch
Hinzufügen einer Zeile, Spalte:

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f^* \end{bmatrix} \sim \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \\ \boxed{k_{*1}} & \boxed{k_{*2}} & \boxed{k_{*3}} \end{bmatrix} \right)$$

K
 K^T
 K_{**}
 K_{*+}
 K_{++}

mit $k_{*+} = K(x_1, x^*)$

=> neue μ, σ :

$$p(f^* | x, f) = N(f^* | \mu^*, \sigma^*)$$

$$\mu^* = E(f^*) = K_*^T K^{-1} f \quad \text{Cholesky Zerlegung}$$

$$\sigma^* = K_{**} - K_*^T K^{-1} K_* = L L^T$$

$$\Rightarrow f^* \sim \mu^* + N(0, 1) L$$

Acquisition Function

- definiert wie informativ Datenpunkte sind
- Tradeoff zw. Exploration und Exploitation
 - (global)
 - (lokal)

$$u(x) = \mu(x) + \beta \sigma(x)$$

- β klein: Exploitation aktueller Optima
- β groß: Exploration möglicher neuer Optima

Erwartete Verbesserung

- bester aktueller Wert: f'

$$\text{Improvement } i(x) = \max(0, f(x) - f')$$

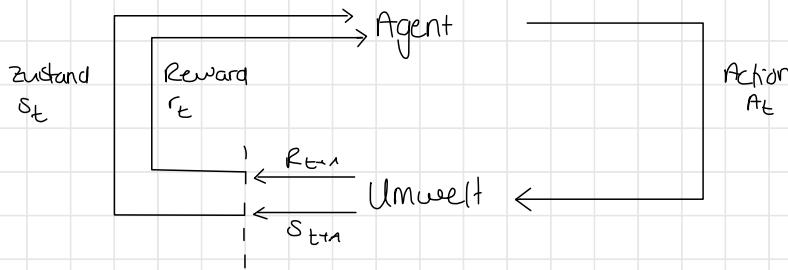
$$\rightarrow u(x) = \mathbb{E}(i(x)) \quad \text{Erwartungswert des Improvements}$$

Anwendungsbeispiele

- * Bayes'sche Optimierung von Hyperparams
 - effizienter als Grid-Search / random search
 - langsam da sequentiell
- * Project Ada
 - Ziel: optimierte Leistungsfähigkeit
 - Vollautom. Experiment (Lösung vorbereiten, auftragen, messen, anpassen)

Reinforcement Learning

Idee: lerne Aktionen so zu wählen, dass der Reward möglichst hoch ist



- = Lernen von sequentiell guten Entscheidungen unter Unsicherheit
 - sequentiell \rightarrow Trial & Error
 - gut \rightarrow muss definiert werden
 - Lernen \rightarrow variable + lernbare Entscheidungsstrategie
 - Unsicherheit \rightarrow unvollst. Wissen
 - = Agent kennt nur Bedo. + Rewards

Herausforderungen

Optimierung

Ziel: finde optimaten weg gute Entscheidungen zu treffen
 \rightarrow Frage: was ist die beste Strategie?

Delayed Consequences

- Konsequenzen erst deutlich später als Action
 (z.B. sparen für Rente, lernen für Klausur)

Planning: Trade-off zw direkt und späteren Vorteilen

Learning: welche vorherige Aktion führt zum Vor-/Nachteil

\Rightarrow Ziel: maximiere gesamten erwarteten zukünftigen Reward

Exploration

- Agent lernt durch Treffen von Entscheidungen
→ sieht dadurch aber nur Teile der Welt

Generalisierung

- großer state-space → muss mapping von Entscheidung auf reward lernen

Markov Prozesse

Markov Prozess

Nächster State s_{t+1} nur abh. vom aktuellen Zustand s_t

$$P(s_{t+1} | s_t, s_{t-1}, \dots, s_0) = P(s_{t+1} | s_t)$$

Markov Reward Process

- states + Übergangswkt + Rewards

Return-function = kumulativer Reward einer Episode
finite Sequenz von Zuständen

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

↑
Summe aller zukünftigen Rewards R_{t+n}

γ = Discount factor < 1
= zukünftige Rewards unwichtiger

Value-function = Beurteilen des Werts einer Episode

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

→ Lösung mit Bellman Exp. Eq.

Bellman Expectation Equation

$$v(s) = \mathbb{E}[G_t | S_t = s] = \dots$$

$$= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

Reward
im nächsten
Schritt

value \approx id
des nächsten Zeitschritts

als
Matrix

schreiben

$$v = R_s + \gamma P v$$

Übergangsmatrix

$$\Rightarrow \text{Lösung } v = (I - \gamma P)^{-1} R$$

(Lesbar falls γ , Rewards, Überg.wkt bekannt,
states begrenzt und $k < n$)

Markov Decision Process

States + Übergangswkt + Actions (=Entscheiduregen)

ABER: Konsequenzen der Actions ggf. unbekannt

Policy π = welche Action in welchem State ausführen

$$\pi : A \times S \rightarrow [0, 1]$$

$$\pi(a, s) = P(A_t = a | S_t = s)$$

- Wkt. neuer Zustand s' : $P_{ss'}^a = P(s' | s, a)$

- Reward abh. von s, a : $R_s^a = \mathbb{E}[R_{t+1} | s, a]$

=> Markov Reward Process mit $P_{s,s'}^\pi = \sum_{a \in A} \pi(a|s) P_{ss'}^a$

$$R_s^\pi = \sum_{a \in A} \pi(a|s) R_s^a$$

State-value function

- finde Policy die den Wert maximiert

$$V_*(s) = \max_{\pi} V_{\pi}(s), V_{\pi}(s) = \mathbb{E}_{\pi}[G_t(s)]$$

Action-value function

- finde beste Action

$$q_*(s) = \max_{\pi} q_{\pi}(s, a)$$

→ berechne Wert für alle Actions und nehme beste

→ beobachte Umgebung

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t(s, a)] = R_s + \gamma \sum_{s' \in S} P_{s \rightarrow s'} V_*(s')$$

Q-Learning löst Bellman optimality equation

Q-Table: # States x # Actions (diskrete Räume)

$$Q^{\text{new}}(s_t, a_t) = Q(s_t, a_t) \leftarrow \text{alter Wert} \quad$$

$$+ \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

↑ ↑ ↑ ↑
Lernrate Reward Discount geschätzter optimaler zukünftiger Wert

- zufälliges Init der Q-Table
- je Schritt: zufällige Action (mit Wert ϵ)
- iter. bis Konvergenz

Deep Q-Learning

NN statt Q-Table → predicte Wert einer Action gegeben State

⊖ instabiles Training

Herausforderungen

Wahl von Modell + Herangehensweise

Abh. von:

- Art der Daten
 - Sequenzen, Grid, ...
- Menge der Daten
 - eher wenige : Gauß'sche Prozesse, Ensemble Methoden
 - sehr viele : NNs, Deep learning
- gibt es die Daten schon?
 - nein: Bayes Optimierung möglich?
Unsicherheit quantifizieren? active learning?
 - ja: supervised learning
- Fragestellung / Ziele
- Designaspekte : generativ vs. screening vs. understanding
- müssen Daten verständlich sein? → Interpretierbare Modelle
- Wie schnell muss predicted werden?

Herausforderungen in Natural Sciences

- Experimentelle Daten
 - Outliers + Noise
 - Nicht-standardisierte Datenanalyse
 - inhomogene Abdeckung des Raums
- Datenverfügbarkeit
 - Simulierte Daten?
 - Data augmentation möglich?
 - Transfer Learning
- ! wenige Daten: wähle Methoden die robuster gg. Overfitting
- Standardisierte / heterogenisierte Daten

- Standardisierung
 - AutoML: autom. Modellauswahl, Training, Hyperparamoptm.
 - Benchmark Datasets
 - stand. Vergleich versch. Modelle / Repräsentationen
- Modellentwicklung
 - bessere Modelle + Generalisierung
 - bessere Repr. der Domäne + Manifold Learning
→ Ziel: minim. Fehler mit wenigen Daten
- Erklärbarkeit, Interpretierbarkeit
 - wissensch. Hypothesen → verständlich für Menschen
 - Domänenwissen, semantisches + symbolisches Wissen, Kontextverständnis
 - Few-shot learning, One-shot learning