KARLSRUHE INSTITUTE OF TECHNOLOGY

Maschinelles Lernen II

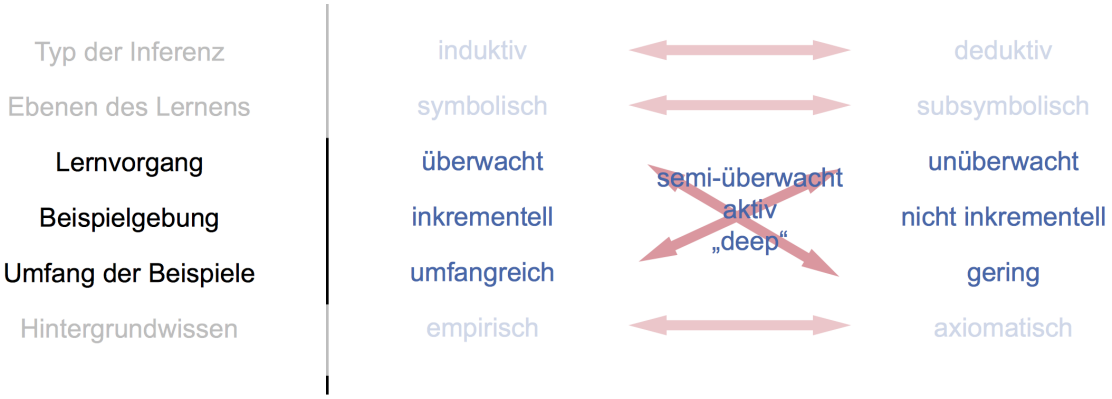
Manuel Lang

5. November 2017

INHALTSVERZEICHNIS

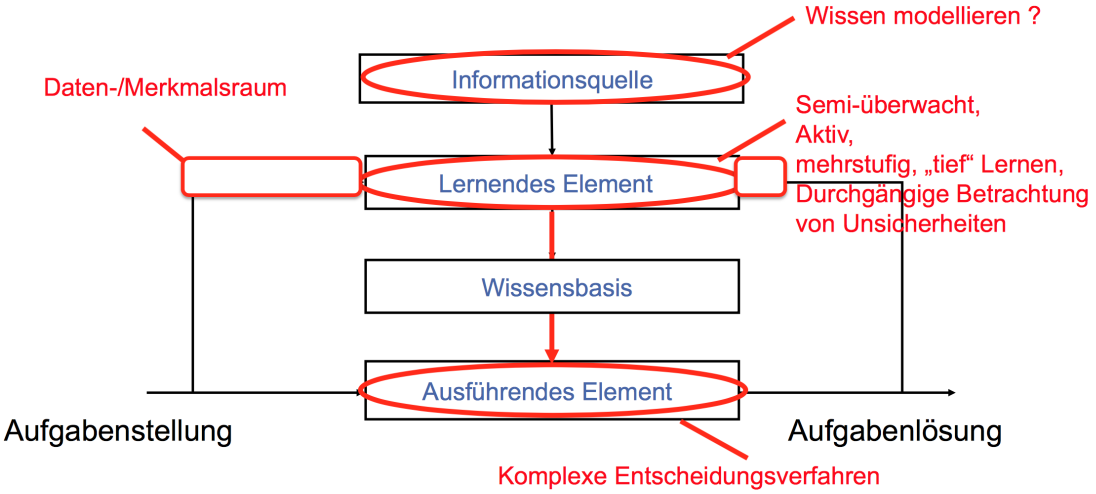
1 Einführung

1.1 Einordnungskriterien für die Einordnung von Grundverfahren



Erweiterte Verfahren
kombinieren verschiedene Methoden oder Architekturen

1.2 Komponenten eines lernenden Systems



2 SEMIÜBERWACHTES LERNEN

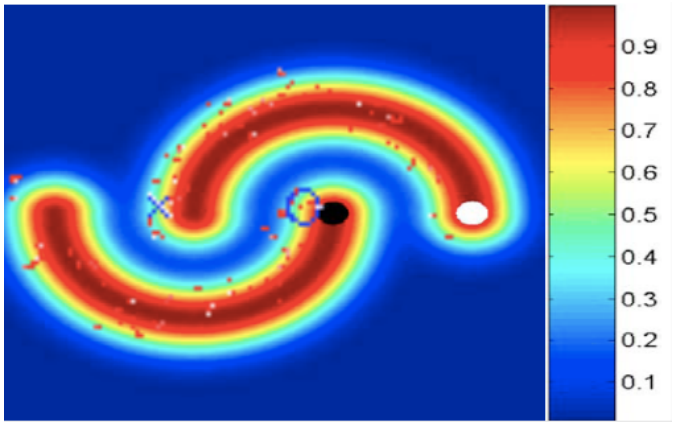
2.1 DEFINITIONEN

2.1.1 DAS SSL-PROBLEM

- Überwachtes Lernen
  - Gelabelte Trainingsdaten: Paare (X, Y)
  - Finde Funktion X (Merkmalsraum) $\Longrightarrow$ Y (z.B. Klassen)
- Unüberwachtes Lernen
  - Ungelabelte Daten aus Merkmalsraum X
  - Strukturen und Labels der Daten finden (Ballungsanalyse), oft auch Dichte-(Träger)-Schätzung
- Semi-Überwachtes Lernen
  - wenige gelabelte Lerndaten, viele ungelabelte Daten
  - Finde Funktion X (Merkmalsraum) $\Longrightarrow$ Y (z.B. Klassen)
  - Bessere Performanz mit minimalen Kosten wird angestrebt: ungelabelte Daten sind billig (Sprache, Videoaufzeichnung, Bilder/Videos im Web), gelabelte Daten sind relativ teuer und schwer zu erzeugen (Annotation von Sprache 400h für 1h Sprachdaten, Experten nötig, Annotationen fehleranfällig)
- SSL: Verwendung von gelabelten und ungelabelten Daten um bessere Hypothese zu erzeugen
- Bessere Hypothese heißt:
  - SSL kann und sollte verglichen werden und besser sein als Ergebnis des überwachten Lernens mit ausschließlich gelabelten Daten und Ergebnis des unüberwachten Lernens mit allen Lerndaten (ohne Labels).
  - Vergleich mit diesen beiden unteren Schranken liefert Informationen darüber, ob die Annahme, die beim semi-überwachten Lernen gemacht wird gilt oder ob der verwendete Ansatz (Modell) diese verletzt.

2.1.2 Grundannahmen

- Gleichmäßigkeit für überwachtes Lernen (Smoothness Assumption): Wenn zwei Datenpunkte x_1, x_2 nahe beieinander sind, dann sollten auch die Ausgaben y_1, y_2 ähnlich sein.
- Gleichmäßigkeit für Semi-überwachtes Lernen: Wenn zwei Datenpunkte x_1, x_2 in einer dichten Region nahe beieinander sind, dann sollten auch die Ausgaben y_1, y_2 ähnlich sein $\implies$ wenn zwei Datenpunkte durch einen Pfad hoher Dichte verbinden sind (i.A. gehören dem gleichen Cluster an), dann sind ihre Ausgaben ähnlich



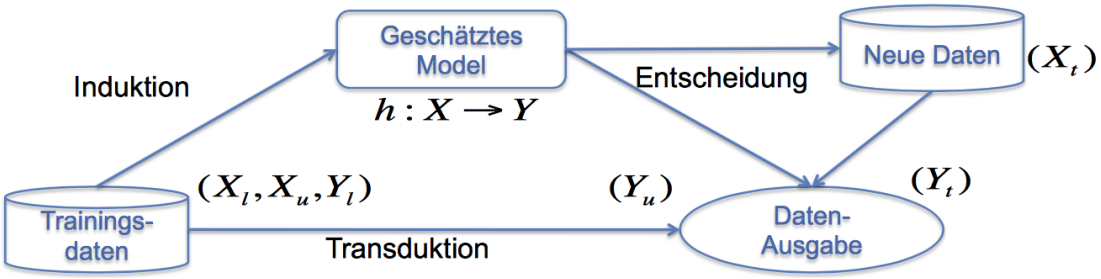
- Cluster oder Dichte-Annahme: Wenn zwei Datenpunkte im selben (dichten) Cluster sind, dann sind sie in derselben Klasse $\implies$ eine Trennung sollte in einer Region niedriger Dichte (zwischen den Clustern liegen)
- Manigfaltigkeit-Annahme (Manifold Assumption): Hochdimensionale Daten haben eine Abbildung in einen i.A. anders dimensionalen Raum (Manigfaltigkeitsraum) in dem sich ihre Strukturen abbilden (unterscheiden/erhalten) $\implies$ Dieser Raum kann dann für die Berechnung des geodäsischen Abstands benutzt werden, approximative Implementierung der Gleichmäßigkeitsannahme

2.1.3 Formalisierung

- Instanzen: Feature-Vektor $x \in X$, Label $y \in Y$
- Hypothese: $h : X \implies Y$
- Gelabelte Daten: $(X_l, Y_l) = (x_{1:l}, y_{1:l})$
- Ungelabelte Daten: $X_u = x_{l+1:n}$
- Üblicherweise gilt $l << n$
- Neue Daten $X_{test} = x_{n+1:...}$

2.1.4 Induktiv vs Transduktiv

- Gegeben die Daten
  - Gelabelte und ungelabelte Lerndaten
  - Ungelabelte Daten
- Induktives Lernen (d.h. auch semi-überwachtes Lernen): Ziel ist das Schätzen einer Hypothese $h : X \implies Y$, die auch unbekannte Daten gut abbildet
- Transduktives Lernen: Ziel ist das Labeln der ungelabelten Daten (kann auch das Labeln neuer Daten sein), das Finden einer Hypothese ist nicht das Ziel, kann aber erfolgen



2.1.5 Problemstellungen

- Überwachtes Lernen (Klassifikation/Regression): $(x_1 1 : n, y_{1:n})$
- Semi-überwachte Klassifikation/Regression: $(x_{1:l}, y_{1:l}), x_{l+1:n}, x_{test}$
- Transduktive Klassikation/Regression: $(x_{1:l}, y_{1:l}, x_{l+1:n})$
- Semi-überwachtes Clustern: $x_{1:n}, must-links, cannot-links$
- Unüberwachtes Lernen: $x_{1:n}$
- **Fokus in ML II: Algorithmen für die Klassifikation**

2.2 SSL-METHODEN

2.2.1 SELF-LEARNING UND CO-TRAINING

2.2.1.1 SELBST-LERNEN / SELF-TRAINING

- Grundidee: Sukzessive Verwenden der ungelabelten Daten, die durch die gelernte Hypothese eine hohe Konfidenz der Prädiktion erreichen (Konfidenz ≈ Wahrscheinlichkeit der Klassenzugehörigkeit)
- Grundalgorithmus (Wrapper): Wiederhole
  - Trainieren mit gelabelten Daten: $h \Leftarrow (X_l, Y_l)$
  - Auswerten der ungelabelten Daten $h(x), \forall x \in X_u$
  - Einfügen konfidenter Daten zu den Trainingsdaten $(X_l, Y_l) \Leftarrow (X_l, Y_l) \cup (x, h(x)) | x \in X_u, h(x) - hoheKonf.$
- Variationen - Hinzunehmen neuer Daten: nur konfidente neue Beispiele, alle Beispiele, mit Gewichtung anhand der Konfidenz (Lernverfahren muss dies erlauben)
- weit verbeiret und vermutlich älterester SSL-Ansatz (1965-1970)
  - Wrapper Algorithmus anwendbar für alle überwachten Lernmethoden
  - Startet auf gelabelten Daten
  - In jeder Iteration werden ungelabelte Daten gelabelt, abh. von der Entscheidungsfunktion
  - Bezeichnung: self-learning, self-labeling, decision-directed learning
- Diskussion
  - Kann effektiv sein
  - Aber es ist nicht genau bestimmt, wie das Ergebnis ist (und abhängig von der Methode des überwachten Lernens)
  - Nicht methodisch festgelegt, welche Annahmen über das Problem getroffen werden
- Vorteile
  - Sehr einfache semi-überwachte Lernmethode
  - Wrapper - passend zu existenten auch komplexten Klassifikatoren
  - Oft angewandt in realen Anwendungen wie z.B. Sprachanalayse
- Nachteile
  - Frühe Fehlentscheidungen können sich verstärken - Heuristische Lösung: Daten un-labeln sofern ihre Konfidenz unter einen Schwellwert fällt
  - Generelle Analyse kompliziert - nur für Spezialfälle ist eine geschlossene und formlose Analyse möglich, in Spezialfällen entspricht Selbst-Lernen dem EM-Ansatz

2.2.1.2 Mit-Lernen / Co-Training

- Annahme: Features können in 2 voneinander unabhängige Mengen aufgeteilt werden $x = [x^{(1)}, x^{(2)}]$ (Feature-Split), somit gilt für jeden Vektor: Jede Untermenge ist ausreichend um eine gute Lernmaschine (im weiteren Klassifikator) einzutrainieren
- Lernansatz: Verwenden den Wrapper-Ansatz mit 2 unabhängigen Klassifikatoren für beide Featuremengen $(X_l^{(1)}, Y_l) \implies h^{(1)}, (X_l^{(2)}, Y_l) \implies h^{(2)}$ (z.B. Bild und Textklassifikation von Webseiten)
- Vorteile
  - Wrapper Methode - anwendbar auf alle existierenden Klassifikatoren
  - Weniger anfällig für Missentscheidungen als Selbstlernen
- Nachteile
  - Natürliche Featureaufteilung ggf. nicht vorhanden
  - Modelle die die vollständige Featuremenge benutzen erreichen oft bessere Ergebnisse
- Varianten
  - Fake Feature Split: zufällig, künstliche Aufteilung der Merkmale, Co-Training wie bisher
  - Multi-View-Ansatz: kein Feature Split, trainiere mehrere Klassifikatoren, Klassifizierung der ungelabelten Daten mit allen Klassifikatoren, verwende Mehrheitsentscheidung für neue Labels
  - Co-EM: Nutzung aller Daten, jeder Klassifikator labelt die Daten X_u probabilistisch, Daten (x, y) werden probabilistisch genutzt mit Gewicht $p(y|x)$

2.2.2 Generative probabilistische Modelle

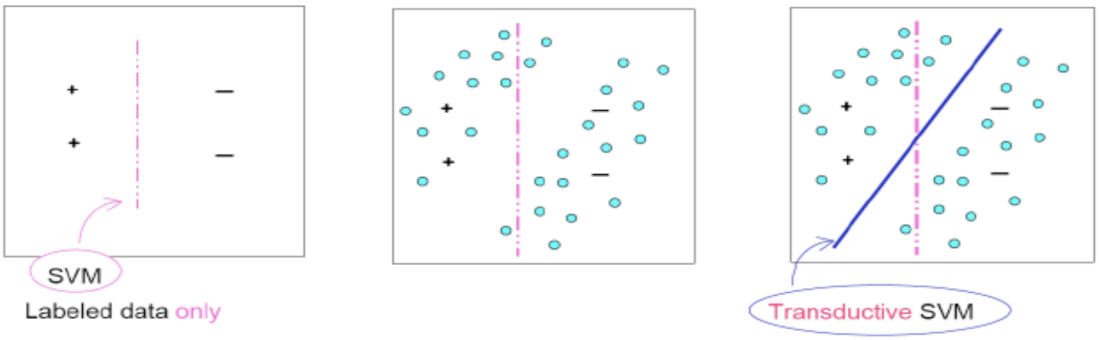
- Generative Algorithmen nutzen eine Schätzung der Verteilung der Daten für die Klassen ⇒ zusätzliche Information der Verteilung der Daten ist sinnvoll.
- Grundverfahren
  1. Wähle ein generatives Modell $p(x, y|\theta)$
  2. Finde Maximum Likelihood Schätzung (MLE) auf gelabelten und ungelabelten Daten $\theta^* = \arg\max_\theta p(X_l, Y_l, X_u|\theta)$ mit Expectation Maximization Algorithmus
     - Starte mit MLE $\theta = \pi_1, \pi_2, \mu_1, \mu_2, \Sigma_1, \Sigma_2$ auf (X_l, Y_l)
     - E-Schritt
       * Berechne Wahrscheinlichkeit für alle $x \in X_u$, $p(y|x, \theta) = \frac{\pi_y p(x|\theta_y)}{p(x|\theta)}$
       * Setze Label (je nach Maximum) $y_u = 1$ bzw. $y_u = 2$

- M-Schritt
  - Update von θ jetzt auch mit (X_u, Y_u) (soft label)
  - A-priori-Klassenwahrscheinlichkeit = Anteil der Daten mit Klasse $i = \pi_i$
  - (gewichteter) Mittelwert der Klasse $i = \mu_i$
  - (gewichtete) Kovarianz der Daten der Klasse $i = \Sigma_i$
- Iteriere
- Kann als Sonderfall des Selbstlernens gesehen werden

3. Bestimme Klassenzugehörigkeit entsprechend der Bayes'schen Regel
   $p(y|x,\theta^*) = \frac{p(x,y|\theta^*_y)}{\sum_{y'} p(x,y'|\theta^*)} = \frac{p(x|\theta^*)p(y|X,Y)}{p(x|\theta^*)}$

- Grundsatz
  - Maximierung von $p(X_l, Y_l, X_u|\theta)$
  - Verwende optimales Modell für die Trennung
  - EM ist nur eine Variante, andere Methoden existieren
- Vorteile
  - Klares, wohl definiertes Framework
  - Kann sehr effektiv sein wenn das Modell korrekt ist
- Nachteile
  - Verifikation der Korrektheit des Modells meist nicht möglich
  - EM kann zu lokalen Minima führen
  - Ungelabelte Daten können schaden wenn das Modell nicht korrekt ist

2.2.3 Low-Density Separation / Dichte Trennung - "Transduktive" SVM



- Annahme: Ungelabelte Daten unterschiedlicher Klassen werden mit großen Rand getrennt - aber wie?

- Naiver Ansatz: Alle 2^u Möglichkeiten der Labels v. $X_u = x_{l+1:l+u}$ betrachten, trainiere SVM für alle Möglichkeiten, wähle SVM mit größtem Rand
- Besser: integriere ungelabelte Daten in das Optimierungsproblem
- Realer Fehler: $R(w) = \int L(y, f(x;w))p(x,y)dxdy$ mit der realen Verteilung der Daten $p(x,y)$
- "Transduktive" SVM - Anpassung
  - Einbinden der ungelabelten Daten
    - Problem: y_i unbekannt für alle ungelabelten Daten
    - Für korrekte Labels würde gelten: $y_i = signf(x_i) for x_i \in X_u$
    - Hinge-Funktion für ungelabelte Daten: $(1-y_i f(x_i))_+ = (1-|f(x_i|)_+$ weil $sign(f(x_i))f(x_i) = |f(x_i)|$
    - Minimierungsproblem kann erweitert werden
      $\min_{w,b} \frac{1}{2}||w||^2 + C_1 \sum_{i=1}^{l}(1-y_i f(x_i)) + C_2 \sum_{i=l+1}^{n}(1-|f(x_i)|)_+$
- SVM^{light}-Ansatz
  - Heuristisches, iteratives Labeln mit Ausbalancierung
  - Trainiere SVM auf (X_l, Y_l)
  - Labeln von X_u durch $f(x_i)$, Label $\hat{y}_i \Leftarrow -1/1$
  - Von $\tilde{C} \Leftarrow 10^-5 C_2$ bis C_2 (schrittweise iteratieren)
    - Trainiere SVM mit
      $\min_{w,b} \frac{1}{2}||w||^2 + C_1 \sum_{i=1}^{l}(1-y_i f(x_i)) + C_2 \sum_{i=l+1}^{n}(1-|f(x_i)|)_+$
    - Tausche Labels $\hat{y}_i, \hat{y}_j$ wenn $\exists(i,j) switchable$
  - Bis keine tauschbaren Labels.
- S^{3}VM (Semi-Supervised Support Vector Machines) probabilistische Sicht für überwachtes Lernen
  - Wahrscheinlichkeit für die Ausgabe y gegeben x: $p(y|x) = \frac{1}{(1+exp(-yf(x)))}$ mit $f(x) = w^T x + b$
  - Log-Gesamtwahrscheinlichkeit (über alle Daten): $\sum_{i=1}^{l} log p(y_i|x_i, w, b)$
  - MAP-Training (inklusive Rand): $\max_{w,b} \sum_{i=1}^{l} log(\frac{1}{(1+exp(-y_i f(x_i)))}) - \lambda||w||^2$ bzw. $\min_{w,b} \sum_{i=1}^{l} log(1+exp(-y_i f(x_i))) + \lambda_1||w||^2$
  - Wenn zwei Klassen gut trennbar sind, dann sollte $p(x|y)$ nahe 0 oder 1 sein für alle Instanzen ungelabelter Daten ⇒ Entropie ist klein
  - Probabilistische Sicht führt zur Kostenfunktion

  - Entropie Betrachung / Regularisierung führt zur Kostenfunktion

- "Transduktive" SVM/S^3VM - Diskussion
  - Vorteile
    * Anwendbar wenn SVM anwendbar
    * Klar formuliertes mathematisches Rahmenwerk
  - Nachteile
    * Optimierungsproblem nicht mehr konvex
    * Optimierung - kompliziert
    * Lokale Minima
    * Schwächere Annahme (Dichte) als generative Modelle oder graphbasierte Methoden ⇒ möglicherweise schlechtere Ergebnisse

2.2.4 Graph basierte Modelle

Nicht in Vorlesung behandelt.

2.2.5 Änderung der Repräsentation

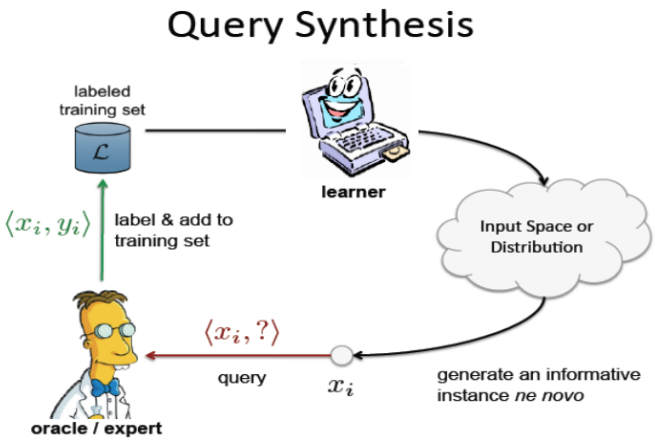
Nicht in Vorlesung behandelt.

3 Aktives Lernen

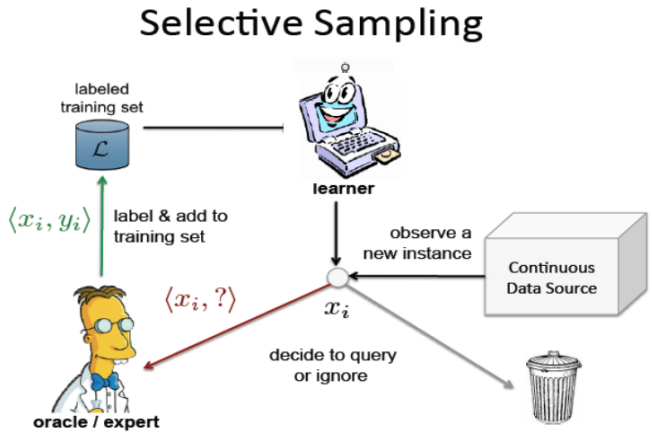
- SSL
  - Lernmaschine die mit wenigen überwachten und vielen unüberwachten Daten lernt
  - Annahme: zusätzliche Informationen über Datenverteilung ermöglicht Hypothesenfindung

- Aktives Lernen (allgemeiner)
  - Lernmaschine die ggf. mit wenigen überwachten aber wesentlich mit selektiv gewählten unüberwachten Daten lernt
  - Annahme: Einige Daten enthalten wesentlich mehr Informationen als andere

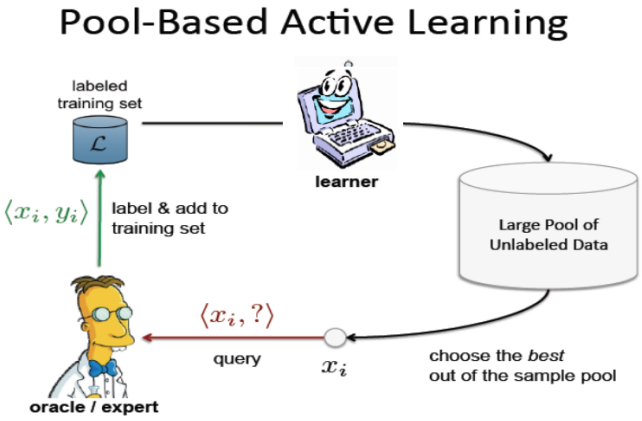
- Query Synthesis (Erzeugung synthetischer Daten)



- Selective Sampling (Selektive Entnahme aus Daten-Strom)



- Pool Based (Auswahl aus Daten-Pool)



3.2 Wahl der Lerndaten

- Wähle die Daten für die Lernmaschine die größte Unsicherheit haben, z.B.
  - Niedrigste Konfidenz (bester Klassenzugehörigkeit) bzw. höchste Unsicherheit $\phi_{LC}(x) = 1 - P_\theta(y^*|x)$
  - Kleinster Rand/Margin (bester Klassenzugehörigkeiten) $\phi_M(x) = P_\theta(y_1^*|x) - P_\theta(y_2^*|x)$
  - Entropie
    * Als Erwartungswert des Informationsgehalts, definiert durch $E_y[-logP_\theta(y|x)]$
    * Maß (zu maximieren): $\phi_{ENT}(x) = -\sum_y P_\theta(y|x) log_2 P_\theta(y|x)$
  - Für binäre Klassifikation sie diese Maße identisch, sonst nicht

3.3 Version Space

- Version Space: Menge aller Hypothesen die konsistent sind mit den Daten
- Annahme: Um so größer der Version Space v ist, um so schlechter ist jede mögliche Hypothese (Klassifikator)
- Ziel beim aktiven Lernen: Reduktion des Version Space
- Verfahren
  - Simpler (naiver) Version Space Algorithmus
    * Bestimme alle konsistenten Hypothesen oder bestimme $|v|$ analytisch
    * Optimales neues x reduziert die Größe vo v am stärksten
      · als Erwartungswert über y (weil Label zunächst unbekannt)
      · über alle Lerndaten inklusive der neuen Daten $L \cup < x, y >$
      $x_{VS}^* = \arg\min_x E_y |v^{L \cup <x,y>}|$
    * Idealerweise lässt sich der Version Space halbieren
    * Binäre Suche implementiert dies in 1D
    * Problem - effiziente Realisierung
      · V kann sehr groß werden oder ist analytisch nicht beschreibbar
      · Idee: Extremen des Hypothesenraums betrachten, wenn die Modelle sich stark widersprechen -> Daten (mit hoher Unsicherheit) reduzieren v
      · Allgemeiner Ansatz: Query-by-Committee
  - Query-by-Committee (QBC)
    * Allgemeiner Ansatz
      · Trainiere eine Menge C von Maschinen (Klassifikatoren)
      · C kann beliebiger Kardinalität sein

        - Wähle neue Daten wenn die Hypothesen (Klassifikatoren) widersprüchlich sind
    - Selektive Entnahme
        - Beobachte neue Instanzen (Auswerten)
        - Abfrage falls Widerspruch
        - Neutrainieren, Iterieren
    - Pool-based Lernen
        - Messung des Widerspruchs für alle Instanzen x
        - Ranking
        - Abfrage der k widersprüchlichsten Instanzen
        - Neutrainieren, Iterieren
    - Design
        - Wahl des Ausschusses C z.B. sampling von zulässigen Modellen geg. Lerndaten entsprechend $P(\theta|L)$ oder Lernen der Modelle <- Datenabhängig
        - Bestimmung des Widerspruchs einfach (z.B. XOR) oder korrekt - Betrachten der Einzelentscheidungen als Wahrscheinlichkeitsverteilung und Unsicherheitsmaß darauf anwenden, z.B. Entropie

- Ausreißerproblem
    - Problem: Eine Instanz kann widersprüchlich sein weil es sich um einen Ausreißer handelt, Ausreißer sind nicht geeignete Lerndaten
    - Mögliche Lösung: Gewichten der Unsicherheit einer Instanz x anhand der Dichte im Datenraum

- Aktives Lernen mit SVM - Version Space
    - Gegeben ungelabelte Instanzen (-> Hyperebenen in H) suchen wir diejenigen, die den VS maximal verringern
    - Einfache Lösung Simple Margin: Daten deren entsprechende Hyperbene die Hyperkugel gültiger Gewichtsvektoren möglichst zentral schneiden, dies sind die Daten die im nächsten zur Trennhyperebene im Merkmalsraum liegen
    - Besser: Nutzen der Eigenschaft, dass der SVM Rand proportional zur Version Space Fläche ist
    - MaxMinMargin: Für jeden Datenpunkt berechne den Rand $m+$ und $m-$ nach potentieller Teilung in $V+$ und $V-$, Abfragen der Instanz $x = \arg\max_x min(m+, m-)$
    - Ratio Margin: Für jeden Datenpunkt berechne den Rand $m+$ und $m-$ nach potentieller Teilung in $V+$ und $V-$, Abfragen der Instanz $x = \arg\max_x min(\frac{m-}{m+}, \frac{m+}{m-})$

- Vorteile

  - Anwendbar wenn SVM anwendbar
  - Klar formuliertes mathematisches Rahmenwerk
  - Berechnung des Randes jeweils nach Trainieren der SVM möglich
  - Praktische Ergebnisse zeigen, dass aktive SVM besser als passive SVM
- Nachteile: MinMax und Ratio sind aufwändig in der Berechnung

4 Tiefes Lernen

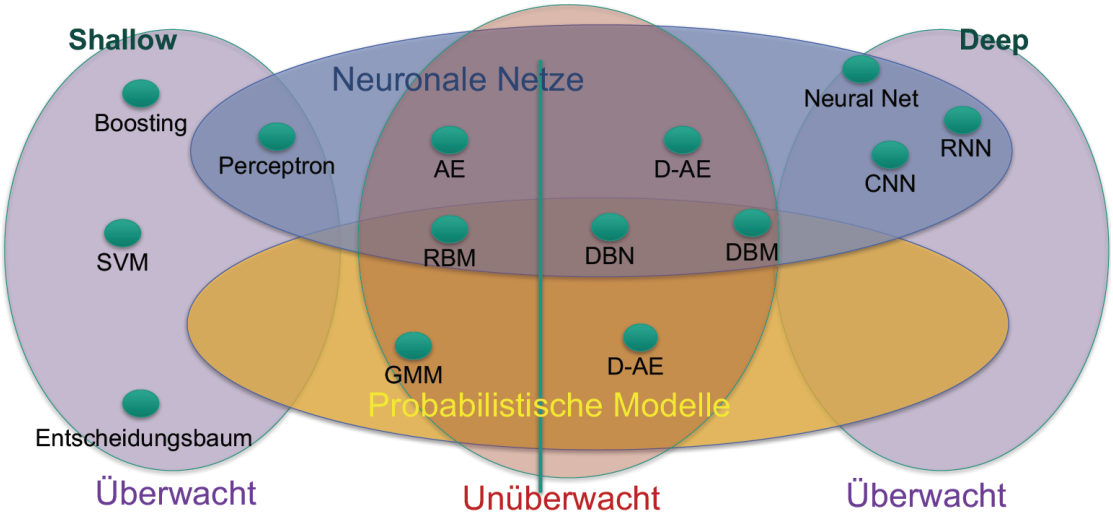
4.1 A temporary disgression

- Vapnik und Kollegen entwickeln die “very clever type of pereception” [Hinton] - Support Vector Machine: “a clever optimization technique is used”, “but it’s just a perceptron and has all the same limitations”
- Hinton: “In the 1990’s, many researchers abandoned neural networks with multiple adaptive hidden layers because Support Vector Machines worked better.”
- Problem von Backpropagation
  - Überwachtes Lernen benötigt zu viele Trainingsdaten mit Zielwerten, aber die meisten realen Daten sind ohne Zielwerte
  - Performanz skaliert oft nicht wirklich gut: sehr langsames Lernen (Konvergenz) in Netzen mit vielen Hidden Layern <- diese werden leider benötigt
  - Lokale Minima
  - Overfitting
- Positive Aspekte von Backpropagation
  - Effizienz und Einfachheit (z.B. Gradienten-Abstieg für das Backprop Lernen) -> Ziel: beibehalten!
  - Aber: Die Struktur der (Input) Daten ist wichtig und sollte gelernt werden, Ziel: Generatives Modell nutzen!

4.2 Definition tiefe Netze/Lernverfahren

- Hinton: “It is deep, if it contains at least two layers with non-linear transformations.”
- LeCun (additional): “It is deep, if it represents a feature hierarchy.”

4.3 Einteilung Lernverfahren



4.4 Belief Netze

- Ein Belief Netz ist ein gerichteter azyklischer Graph mit stochastischen Variablen (graphisches probabilistisches Modell).
- Einige Variable sind beobachtbar (Evidenzen, Daten)
- Inferenz-Problem: Schließen auf den Zustand der verdeckten Variablen
- Lern-Problem: Anpassen der Variablen um das Netz zu befähigen beobachtete Daten generieren zu können
- Einfaches Netz mit stochastischen binären Einheiten (Bernoulli Variablen)
  - Mögliche Zustände der Knoten: 1 oder 0
  - Aktivierungswahrscheinlichkeit: bestimmt durch den gewichteten Input von anderen (Vorgänger) Einheiten (plus Bias) $b_i + \sum_j s_j w_{ji}$, z.B. für sigmoide Einheiten entsprechend der Sigmoid Regel: $p(s_i = 1) = \frac{1}{1+exp(-b_i - \sum_j s_j w_{ji})}$
- Lernen in “Deep” Belief Nets
  - Einfach (unbiased) Beispiele an den Endknoten zu generieren um zu sehen woran das Netz “glaubt” -> Belief Net
  - Schwer die a-posteriori Wahrscheinlichkeit für alle Konfigurationen der verdeckten Ursachen zu inferieren
  - Wie kann man dann in Deep Belief Netzen mit vielen Ebenen und Millionen von Parametern lernen?
- Lernen für Sigmoide Belief Netze

  - Lernen einfach wenn Daten der verdeckten Zustände (samples der a-posteriori Verteilung) gegeben der beobachteten Daten vorliegen
  - Idee des Lernens: Für jede Einheit: Maximiere die Wahrscheinlichkeit, dass der beobachtete Zustand s_i von den binären Zuständen der Vorgänger generiert wird (Log Likelihood)
  - Problem: Wenn überhaupt nur für unterste Ebenen (micht sichtbaren Daten) möglich

4.5 Restricted Boltzmann Maschinen (RBM)

- Verbindung der binären stochastischen Einheiten in einem gerichteten azyklischen Graphen führt zu einem Sigmoid Belief Netz -> problematisch
- Verbindung der binären stochastischen Einheiten mit symmetrischen Verbindungen führt zu einer Boltzmann Maschine -> bei Restriktion der Verbindung kann eine Boltzmann Maschine (gut) gelernt werden
- Beschränkung der Verbindungen der Restricted Boltzmann Machine um lernen zu können: nur ein hidden Layer, keine Verbindung zwischen Einheiten gleicher Layer
- In einer RBM sind die hidden Einheiten unabhängig, wenn die beobachtbaren Variablen (Zustände) gegeben sind: Man erhält "einfach" den Zustand (sample der a-posteriori Verteilung) der hidden Einheiten gegeben ein Daten-Vektor <- sigmoid-Regel (aufwärts), wesentlicher Vorteil gegenüber den gerichteten Belief Netzen

Lernalgorithmen für RBM

- Maximum Likelihood Lernalgorithmus
  - Start mit einem Trainingsvektor an den sichtbaren Einheiten
  - Alternieren zwischen update aller sichtbaren Einheiten (parallel). (Implementiert alternating Gibbs <- MCMC Ansatz um ein sample der Daten zu erhalten)
  - Dann gilt (Gradient): $\Delta w_{ij} \approx \frac{\partial log p(v)}{\partial w_{ij}} =< v_i h_j >^0 - < v_i h_j >^\infty$
- Schneller Lernalgorithmus für RBM - Contrastive divergence learning
  - Start Trainingsvektor an den sichtbaren Einheiten
  - Update aller hidden Einheiten - parallel
  - Update aller sichtbaren Einheiten (parallel) um eine Rekonstruktion zu erhalten
  - Update der hidden Einheiten
  - Update der Gewichte (nach update der hidden Einheiten): $\Delta w_{ij} = \epsilon(< v_i h_j)^0 - < v_i h_j >^1)$
  - Entspricht nicht ganz dem Gradienten des log likelihood aber funktioniert gut

4.6 Tiefe Netze mit RBM (DBM)

- Grundidee: mehrere RBM stapeln
- Trainieren des layer, welche die Eingaben direkt von den Pixeln enthält (W_1)
- Dann Aktivierung der trainierten Merkmale so verwenden als wären es Pixel und Lernen der zweiten verdeckten Schicht (W_2) usw.
- Es kann gezeigt werden, dass jedes Mal wenn eine weitere Schicht hinzukommt, die untere Schwelle des log-likelihood der trainierten Daten erhöht wird (Beweis ist nicht trivial).
- Äquivalenz zwischen RBM und gerichteten Deep Netzen

4.6.1 Wieso funktioniert greedy Lernen?

- Die Gewichte W, in der unteren Ebene des RBM definieren $p(v|h)$
- Es gilt für die (untere) RBM: $p(v) = \sum_h p(h)p(v|h)$
- Verbesserung von $p(h)$ wird $p(v)$ verbessern
- Um $p(h)$ zu verbessern benötigen wir ein besseres Modell um die a-posteriori Wahrscheinlichkeit der hidden Einheiten zu generieren <- d.h. die oberen RBM lernen

4.6.2 Fine-tuning des Lernens: contrastive "wake-sleep" Algorithmus

- Nach Lernen der einzelnen Layer -> Verbesserung durch fine-tuning
- Lösen der symmetrischen Bindung zwischen den auf- und abwärts-Gewichten
- Algorithmus
  1. Wake (Aufwärtsschritt): Starte Anregung mit sichtbaren Daten, Anpassen der top-down Gewichte um die Aktivität der Merkmale in tiefere Ebenen zu rekonstruieren.
  2. Einige Iterationen in der höchsten RBM -> Anpassen der Gewichte in der obersten RBM
  3. Sleep (Abwärtsschritt): Anpassen der bottom-up Gewichte um die Aktivität der Merkmale in die höhere Ebene zu rekonstruieren

4.6.3 Funktionsweise kaskadierte RBM - Generativ

- Gegeben z.B. ein (eingelerntes) Modell mit 3 Schichten
- Daten generieren
  1. Anlegen Daten obere Schicht

2. Einstellen eines Gleichgewichts der obersten RBM durch alternierendes “Gibbs sampling” (abwechselnd je ein Knoten aktualisieren, über mehrere Iterationen).
3. Ein Abwärtsschritt um Zustand der anderen Ebenen zu erhalten. -> Fertig

- Die Verbindungen der unteren Schicht nach oben sind nicht Teil des generativen Modells sondern werden für die Inferenz verwendet.

4.6.4 Fine-tuning für Diskrimination

- Vorgehensweise
  - Lernen je eines layers - “greedily”
  - Fine-tuning durch contrastive wake-sleep (auf dem vortrainierten Netz)
  - Verwenden von Backpropagation: Zusätzliche Ebene der Entscheidung (oben), Daten anlegen -> Gleichgewicht, Label -> Backprop -> Lernen oberer Gewichte
- Netz -> erreicht eine bessere Diskriminationsfähigkeit
- Besser als Standard-Backpropagation auf NN

4.6.5 Zwischenfazit

- RBM liefern eine einfache Methode um mehrere Ebenen von Merkmalen unüberwacht zu lernen: echtes Maximum Likelihood Lernen ist aufwendig (nicht möglich), aber s.g. Contrastive Diveregence Lernen ist schnell und gut
- Mehrere Ebenen können gut gelernt werden wenn die verdeckten Zustände unterer RBM als sichtbare Einheiten der nächsten RBM betrachtet werden
- Dies führt zu guten generativen Modellen die noch verfeinert werden können (fine-tune): Contrastive wake-sleep, diskrimantives fine-tuning

4.7 Äquivalenz zu Sigmoiden Belief Netzen

- Es gibt unendliche Sigmoide Belief Netze die äquivalenz zu einem RBM sind
- W - gekoppelt, Dimensionen der v und h Ebenenen gleich
- die bedingten Wahrscheinlichkeiten $p(v|h)$ und $p(h|v)$ sind definiert durch W
- Ein Abwärtsschritt im gerichteten Netz ist genau äquivalent dazu, das RBM Netz ins Gleichgewicht zu bringen.
- Inferenz in einem gerichteten Netz mit wiederholten Gewichten
  - Inferenz: Multipliziere v_0 mit W^T + Sigmoid Aktivierung
  - Das Modell oberhalb h_0 implementiert zusätzliche a-priori Daten

  - Inferenz in einem gerichteten Netz ist äquivalent dem Fall eine RBM ins Gleichgewicht zu bringen beginnend mit sichtbaren Daten.

- Lernen in einem gerichteten Deep Netz
  - Zunächst Lernen mit gekoppelten Gewichten: entspricht Lernen von RBM, Ignorieren kleiner Abweichungen durh die gekoppelten Gewichte der höheren Ebenen
  - Anschließend Einfrieren des ersten Layer in beide Richtungen und Lernen der nächsten Gewichte (diese sind immer noch gekoppelt): äquivalent dem Lernen einer weiteren RBM unter Verwendung der aggregierten a-posteriori Verteilung von h_0 als Daten
  - Ist dies korrekt? -> Nicht ganz, aber hinreichend: Inferenz mit entkoppelten Gewichten der Layer ist nicht wirklich korrekt denn es entspricht nicht mehr einer RBM, Hinton et al. zeigten, dass dies eine gute Approximation (Verbesserung) ist, Fazit: kaskadierte RBM - gute Annäherung an Deep Belief Netze

5 Convolutional Neural Networks (CNNs)

- Daten werden in Form von Features Maps von Layer zu Layer übertragen
  - Jeder Layer erhält aks Eingabe r Feature Maps (mxm)
  - Im Input Layer gilt: Feature Maps = Eingabebild (RGB Eingabe: $mxmx3$)
  - Alle Feature Maps zwischen zwei Layern entsprechen einem Tensor

- Pooling Layer
  - Zusammenfassung kleiner Bildbereiche: pxp Bereich, p gewählt nach Größe des Eingabebildes, meist $p \in [2,5]$, 2 für eher kleine Inputs, 5 eher große
  - Verschiedene Strategien: Max Pooling $maxa \in A$, Mean Pooling $\frac{1}{|A|} \sum_{a \in A} a$, Stochastic Pooling
  - Nutzen: Lokale Translationsinvarianz, Invarianz gegen leichte Veränderungen und Verzerrungen, Datenreduktion

- Convolutional Layer
  - Anwendung von Faltungsoperationen auf die Eingabe
  - Eingabe: Feature Maps des vorherigen Layers
  - Ausgabe: Feature Maps des Layers entstanden durch Faltung der Eingabe mit k Filtern

- Stride
  - “Schrittweite” des Faltungskernels in beiden Dimensionen
  - Stride muss zur Größe der Eingabe passen (in Abhängigkeit zur Größe des Filterkernels)

  - Bei Convolutions- und Pooling Layern (bei Pooling Layern meist s=1)

- Stride vs Pooling
  - Größenreduktion durch ConvLayer Stride möglich
  - Wieso extra Layer?
  - Unterschied: ConvLayer Stride erreicht Reduktion durch Auslassen, Max Pooling betrachtet alle Werte bei Reduktion

- Randbetrachtung
  - Relevant v.a. bei Convolutional Layern (bei Convolution Filtern größer 1x1)
  - Ursprungsgröße geht durch Randeffekte bei Faltung verloren: mögliche Strides für jeden Convolution Layer andere
  - Möglichkeiten zur Behandlung
    * Don't care
      · Berechnung des inneren Bereichs
      · Reduktion der Ergebnisgröße
    * Padding
      · Auffüllen des Randbereichs
      · Stabilisiert Größenverlauf der Layer
      · Varianten: Zero Padding (Auffüllen der Randbereiche mit 0), Nearest Padding (Duplizieren der Randpixel), Reflect Padding (Matrix nach außen spiegeln)
      · Mögliches Problem: Aliasing am Rand

- Dropout
  - Ziel: Reduktion von Overfitting
  - Methode: Während des Trainings Deaktivieren einzelner Neuronen mit Wahrscheinlichkeit p, Entscheidung über Deaktivierung in jedem Trainingsschritt
  - Dropout zäglt zu den Regularisierungsmethoden für CNNs

- Aktivierungsfunktionen
  - Rectified Linear Unit (ReLU): $f(x) = max(0, x)$
  - Leaky ReLU (LReLU): $f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
  - Parametric ReLU (PReLU): Generalisierung von LReLU $f(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
  - Exponential Linear Unit (ELU) $f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$

- Features in CNNs
  - Klassisch werden Features manuell erstellt (SIFT, HOG, FREAK, ...)
  - CNNs lernen gute Feature für eine bestimmte Domäne lernen (in den Convolutional Layern)
  - Features sind hierarchisch (low-level->mid-level->high-level)
- Weight Sharing
  - klassische neuronale Netze: kein Weight Sharing -> schnell sehr viele Gewichte
  - CNN: Gewichte nur in Filtern und lokale Wiederbenutzung von Gewichten -> deutliche Reduzierung der zu lernenden Gewichte
- Initialisieren der Gewichte
  - Random
  - Fixed Feature Extractor: Übernahme von trainiertem Netz, Fixieren der Gewichte der Feature Extraction Schichten
  - Fine-Tuning: Übernahme von trainiertem Netz: geringere Lernrate für ausgewählte Schichten
  - Pretrained Initialization: Übernahme von trainiertem Netz lediglich zur Initialisierung, normale Lernrate
- Objektdetektion mit CNNs
  - Ansatz: Auswahl von Regionen, Klassifikation der Regionen durch Klassifikationsnetze
  - Regionsauswahl: Sliding Window, Interest Point Detektor
  - Nachteile: eine Netzausführung pro Region, zeitintensiv
- Fully Convolutional Networks (FCN)
  - Konvertierung der Fully Connected Schichten in Convolutional Schichten
  - Netze äquivalent bei gleicher Eingabegröße
  - variable Eingabegröße
  - Ausgabe: Wahrscheinlichkeit dafür, dass ein Bereich/Pixel im Bild zur entsprechenden Klasse gehört -> Segmentierung damit und mit implizitem Sliding Window

6 Reinforcement Learning

6.1 Erweiterte Methoden

6.1.1 Effiziente Funktionsapproximation

- Große Zustandsräume erfordern eine gute Generalisierung der gelernten V/Q-Funktion (i.A. inkrementelles Lernen nötig)

- Grundlegende Idee (Q-Lernen): Überwachtes Lernen
  $((s_1, a_1), Q(s_1, a_1)), ((s_2, a_2), Q(s_2, a_2)), ..., ((s_M, a_m), Q(s_M, aM))$
- Endliche Trainingsmenge aus Eingaben und Zielwerten wird durch ein Funktionsapproximator auf eine Funktion abgebildet
- Realisierung: Fitted Q-Iteration
  - Pseudo Code
    - Require: Q-Funktion $\hat{Q} : (SxA) -> R$
    - loop
      - Berechne Strategie π aus $\hat{Q}$ (z.B. ϵ-greedy)
      - Sampling von Übergängen (s_t, a_t, r_t, s_{t+1}) mit $a_t = \pi(s_t)$
      - Erstellung der Trainingsmenge $(s_t, a_t), r_t + \alpha \max_{a \in A} \hat{Q}(s_{t+1}, a)$
      - Trainieren eines Funktionsapproximators auf Trainingsmenge ergibt einen neuen (aktuelle) Approximator $\hat{Q}$
    - end loop
  - Vorteile
    - Für bestimmte Klassen von Funktionsapproximatoren lässt sich Konvergenz beweisen (z.B. im Zusammenhang mit Linearisierung/Diskretisierung)
    - Generell stabilere Approximation der Q-Funktion und besonders dateneffiziente Exploration (d.h. die Fähigkeit anhand einer sehr begrenzten Zahl von Interaktionen zu lernen)

6.1.2 Hierarchisches Reinforcement Learning

- Möglichkeit um spezielle Strukturen (Hierarchien) in den MDP einzufügen
  - Verhalten, Skills, Unteraufgaben
  - z.B.: Ausdruck aufnehmen, kollisionsfreie Navigation, Ausliefern, Türe öffnen
  - RL mit temporär erweiterbaren Aktionen
- Integration von zusätzlichem Wissen
- Zustands-/Aktionsabstraktion
  - Zustände mit weniger Zustandsvariablen
  - verschiedene Umweltzustände -> auf einen abstrakten Zustand abbilden
  - Unterschiedlich abstrakte Zustände in verschiedenen Makro-Aktionen
  - -> Lernen beschleunigen
- Options
  - Ansatz: Verwende s.g. Options = wohl definiertes Verfahlten $o := (I_o, \pi_o, \beta_o)$

- I_o: Menge von Zuständen in denen o gestartet werden kann
- $\pi_o(s) : S \Rightarrow A^*$: Policy, während o ausgeführt wird: A^* kann ebenfalls über eine Policy über weitere Optionen definiert sein
- $\beta_o(s)$: Wahrscheinlichkeit dass o in s beendet wird
- Auf Options o kann eine weitere MDP aufgesetzt werden: Policy μ
- Lernen auf Options
  * Eine Option o ist eine zeitlich erweiterte Aktion mit wohl definierter interner policy
  * Ein-Schritt-Options sind bisherige Aktionen (die in jedem Zustand zulässig sind und danach enden)
  * Die Menge der Options O ersetzt die Menge der Aktionen
  * Lernen (RL) kann auf Options stattfinden -> Gelernte Policy $\mu : S- > 0$
  * ABER: Lernen auf Optionen erfordert die Erweiterung auf Semi-MDP
- Semi-MDP
  * Im klassischen MDP spielt nur die sequentielle Natur der Entscheidung eine Rolle, nicht die Zeit, die zwischen zwei Entscheidungen liegt (= Länge der Aktion)
  * Eine Verallgemeinerung ist der Semi-MDP: Gegeben: T - Dauer einer Aktion, dann betrachtet man für den Zustandsübergang die Wahrscheinlichkeit $P(s', T|s, a)$ und die Bellmann Gleichungen
    $Q(s,a) = r(s,a) + \sum_{s',T} \gamma^T P(s', T|s, a) V^*(s') = r(s,a) + \sum_{s',T} \gamma_T P(s', T|s, a) \max_{a'} Q(s', a')$
  * MDP-Sarsa: $Q_{k+1}(s,a) = (1-\alpha_k) Q_k(s,a) + \alpha_k [r + \gamma Q_k(s', a')]$
  * Semi-MDP: Wenn a eine zusammengesetzte Aktion ist und a wird in s ausgeführt, zum Zeitpunkt t, die Transition benötigt τ und man erhält die Teilrewards r_{t+i}
    $Q_{k+1}(s,a) = (1-\alpha_k) Q_k(s,a) + \alpha_k [r_{t+1}, \gamma r_{t+1} + ... + \gamma^{\tau-1} r_{t+\tau} + \gamma^\tau \max_{a' \in A_{s'}} Q_k(s', a')]$
- Options - nichtdeterministischer Semi-MDP
  * Es gilt: R und ein P (keine Wahrscheinlichkeit) lassen sich neu definieren
  * Damit kann man RL auf Optionen durchführen
    $R(s,o) = E r_{t+1} + \gamma r_{t+2} + ... + \gamma^{\tau-1} r_{t+\tau} | \epsilon(o,s,t)$
    $P(s'|s,o) = \sum_{\tau=1}^{\infty} p(s', \tau) \gamma^\tau$
    $V_O^*(s) = \max_{o \in O_s} [R(s,o) + \sum_{s'} P(s'|s,o) V_O^*(s')]$
    $Q_O^*(s,o) = R(s,o) + \sum_{s'} P(s'|s,o) \max_{o' \in O_{s'}} Q_O^*(s', o')$
  * Lernen auf Options bleibt gleich (r - akkumulierter Reward nach o): $Q_{k+1}(s,o) = (1-\alpha_k) Q_k(s,o) + \alpha_k [r + \gamma^\tau \max_{o' \in Os'} Q_k(s', o')]$

  - Options-Ansatz strukturiert S, A, π, G
  - Lernen findet zunächst auf Makro-Ebene statt und wird vereinfacht <- lediglich Start-und Terminalzustände sind für Options relevant: Hintergrundwissen wird (derzeit) benötigt um sinnvolle Options und primitive Aktionen zu definieren, Option-Policies werden meist vorgegeben
  - Intra-Option-Lernen kann für Optionen verwendet werden -> Reduzierung des Lernens zunächst auf eine Untermenge der Zustände und Aktionen (z.B. Bewegung im Gang), Automatische Erkennung von Zwischenzielen ist Gegenstand aktueller Forschung

6.2 Deep Reinforcement Learning

- Deep Q-Learning
  - verwendet CNN um effiziente Q-Funktion zu approximieren
  - arbeitet mit hochdimensionalem Zustandsraum S
  - gute Performance in unterschiedlichen Domänen
- Schwierigkeiten
  - großer Datenbedarf
  - hohe Korrelation der Daten
  - beobachtete Daten beeinflusst durch aktuelle Policy
  - Lösung: Experience Replay, Sollwert-Netzwerk
- Experience Replay: Speichern und Sampling von Übergängen (s_t, a_t, r_t, s_{t+1})
  - Übergänge können mehrfach genutzt werden
  - Korrelation der Beobachtungen wird gebrochen
  - verhindert Feedback-Loop
- Training mit Experience Replay
  - Initialisiere Deep-Q-Netzwerk (DQN) mit zufälligen Gewichten θ
  - Iteriere über Trainingsepisoden
    * Wähle Aktion $a_t = \arg\max_a(Q(s_t, a; \theta))$ (ϵ-greedy)
    * Führe a_t aus und erhalte Reward r_t und Zustand s_{t+1}
    * Speicher Übergang (s_t, a_t, r_t, s_{t+1}) in M
    * Sample zufälligen Batch von Übergängen (s_i, a_i, r_i, s_{i+1})
      · Definiere Sollwert $y_i = \begin{cases} r_i & \text{falls } s_{i+1} \text{ terminal} \\ r_i + \gamma \max_a(Q(s_{i+1}, a; \theta)) & \text{sonst} \end{cases}$
      · Gradientenabstieg mit Error $(y_i - Q(s_i, a_i; \theta))^2$ auf Q-Netzwerk

- Verwendung eines Sollwert-Netzwerks
  - Initialisiere Deep-Q-Netzwerk (DQN) mit zufälligen Gewichten θ
  - Initialisiere Sollwert-Netzwerk $\hat{Q}$ mit $\hat{\theta} = \theta$
  - Iteriere über Trainingsepisoden
    * Wähle Aktion $a_t = \arg\max_a(Q(s_t, a; \theta))$ (ϵ-greedy)
    * Führe a_t aus und erhalte Reward r_t und Zustand s_{t+1}
    * Speicher Übergang (s_t, a_t, r_t, s_{t+1}) in M
    * Sample zufälligen Batch von Übergängen (s_i, a_i, r_i, s_{i+1})
      · Definiere Sollwert $y_i = \begin{cases} r_i & \text{falls } s_{i+1} \text{ terminal} \\ r_i + \gamma \max_a(\hat{Q}(s_{i+1}, a; \hat{\theta})) & \text{sonst} \end{cases}$
      · Gradientenabstieg mit Error $(y_i - Q(s_i, a_i; \theta))^2$ auf Q-Netzwerk
      · Alle n Schritte: Setze $\hat{Q} = (1 - \alpha)\hat{Q} + \alpha Q$

7 Spiking Neural Nets

7.1 Einblick in biologische Prinzipien

- Das menschliche Gehirn
  - 86 Milliarden Neuronen (Nervenzellen)
  - Pro Neuron zwischen 5.000 und 10.000 Synapsen
  - Wissen encodiert in den Gewichten der Synapsen
  - Gelernt wird: 100 Billionen Synapsen, 100 Milliarden Bits genetischer Code
  - 600 Millionen Jahre Evolution vor dem menschlichen Gehirn
- Was uns menschlich macht - der Kortex
  - Subregionen des Kortex spezifisch für Funktionen
  - Regionen können andere trotzdem ersetzen
  - Unterstützt “single learning algorithm” Theorie
- Neuronen - Nervenzellen
  - verschiedene Typen
  - Aufbau eines Neurons: Dendriten (Eingabe), Soma (Aufsummierung), Axon (Ausgabe), Synapsen (Verbindungen)

- Wieso auf Spike-Level modellieren?
  - Viele Funktionen basieren auf preziser Pulszeit, z.B. Tonwahrnehmung im 3D, Vision, ...
  - Viele Funktion basieren zudem auf Frequenz-Coding, z.B. Muskelbewegungen
- Klassisches Spiking neuron model
  - Differentialgleichungen w.r.t. Zeit
  - PSP-förmig
  - Eingabe: aktuell
  - Ausgabe: Spikes
  - Variablen: Membran-Potential: $V(t)$
  - Parameter: Schwellwert V_{th}, Ruhepotential (Neuron nicht beeinflusst durch Spikes) V_{rest}, "Leak": τ_m, "Refractory period" (Zeit wie lange das Neuron nichtmehr pulsieren kann): τ_{ref}
- Populäre Spiking Neuron Modelle
  - Integrate-and-fire model
    - Beliebtestes Modell für Neuro-Ingenieure
    - Basiert auf elektrischen Stromkreisen
    - Einfach und schnell
    - Modelliert nicht die Form des Aktionspotentials
    - Membranpotential: Kondensator
    - Schwellwert: Gatter
    - Ruhepotential: Batterie
    - Leak: Widerstand
      $\tau_m \frac{dV(t)}{dt} = V_{rest} - V(t) + R \cdot I(t)$
  - Hodgkin-Huxley model
    - Modelliert das Potential der Membran eines Oktopus neurons
    - Basiert ebenfalls auf einem elektrischen Schaltkreis
  - Izhikevich's neuron model
    - Differentialgleichung 2. Ordnung
      $\frac{dV(t)}{dt} = 0.04 \cdot V(t)^2 + 5 \cdot V(t) + 140 - w(t) + I(t)$
      $\frac{dw(t)}{dt} = a \cdot (b \cdot V(t) - w(t))$
    - Konstanten gefittet auf biologische Daten
    - a und b können für verschiedene Dynamiken angepasst werden

    - Schnell zu simulieren
  - Spike Response model
    - Generalisierung des leaky integrate-and-fire Modells
    - Formulierung mit Filtern anstatt von Differentialgleichungen
      $V(t) = \eta(t - \hat{t}) + \int_0^\infty \kappa(t - \hat{t}, s) \cdot I(t - s) ds$
    - $\hat{t}$: letzte Spikezeit des Neurons
    - η die Antwort des Modells zu eigenen Spikes
    - κ die Antwort auf eingehende Spikes
    - Schwellwert ist abhängig vom letzten Spike $\theta(t - \hat{t})$

7.3 Neural Coding

- Analoges neuronales Netz: statisch, räumlich, keine zeitliches Konzept
- Spiking neural network: dynamisch, raum-zeitlich, definiert w.r.t. Zeit
- Rate coding
  - Pulsrate wird berechnet über diskrete Zeitintervalle
  - Eingabevektoren gemappt zu Ausgabevektoren
  - Rate-basierte Netzwerke = analoge Netze
  - Berechne der Pulsraten ist langsam
  - Ineffizient
  - Verwendung: Kognition und Bilder
- Binary coding
  - Wenn ein Neuron feuert, ist es aktiv für eine gegebene Zeit Δt
  - Spike kann zu jeder Zeit erfasst werden
  - Verwendung: stochastische Inferenz
- Gaussian Coding
  - Neuronen haben räumliche Positionen
  - Gaussian gefittet auf Spikingrates
  - Anwendung: Propriezeption in Muskeln

7.4 Zwischenfazit

- Gehirn entwickelte sich zusammen mit dem Körper
- Neuronen verwenden aktuelles als Eingabe, geben Spikes aus und maintainen ein Membran-Potential
- Synapsen transformieren Spikes zu "current" (Post-Synaptic Potential, PSP)
- Spiking neural networks modellieren Spikes, analoge Netze modellieren Spikeraten
- Spiking neural networks sind dynamische Systeme definiert w.r.t. Zeit
- Viele Möglichkeiten Spike-Informationen zu kodieren

7.5 Lernen mit gepulsten Netzen

- Backpropagation funktioniert nicht, weil Spikes nicht differenzierbar sind (Input->Output Mapping nicht vorhanden) -> nur mit Rate-Coding (Intervall über Zeit) möglich (ist aber auch eher ein analoges Netz)
- Backpropagation nicht biologisch plausibel (forward->backward->forward->backward nicht logisch)
- Synaptic plasticity
  - Hebb'sches Lernen
  - Änderung nur bei verbundenen Neuronen (local)
  - Änderung nur bei aktivierten Neuronen (cooperative)
  - Short-term oder Long-term (nur Long-term wird weiter betrachtet <- Änderung für einige Minuten/Stunden im Gehirn)
- Spike-timing-dependant plasticity
  - Synaptische Stärke ändert sich mit Differenz von pre- und post-synaptischen Neuronen $\delta w_{ij} = \sum W(t_j^{post} - t_i^{pre})$
  - Kernelfunktion W initialisiert durch Exponentialfunktion mit Länge τ multipliziert mit Amplitude
- Rate based plasticity
  - Frequenz statt Spikezeit
  - $\frac{d}{dt} w_{ij} = F(w_{ij}, v_i^{pre}, v_j post)$ mit Kernelfunktion F
  - Andere Repräsentation der Kernelfunktion
- Reward based plasticity
  - Gewichtsupdates zusätzlich abhängig von globalem Reward (z.B. Dopamin)

  - $\Delta w_{ij} = \sum W(t_j^{post} - t_i^{pre}, reward)$
  - Problem: Reward ist global, welche Synapsen sind dafür verantwortlich? (Credit Assignment Problem)
  - Neuromodulatoren: Dopamin, Serotonin, Acetylcholine, Norepinephrine, ...
  - Liefern Informationen über Änderungen der synaptischen Gewichte
- Structural plasticity
  - Bisher: Wie Gewichte der Synapsen ändern?
  - Nun: Erzeugen und Entfernen von Synapsen (Lernen: mehr Änderungen)
  - Lernen neue Verbindungen erzeugen statt Gewichte der Synapsen ändern
  - Neue Verbindungen nötig (Rewiring)
- Unsupervised learning with STDP
  - Voll-berbunden zwischen Eingabe und excitatory Neuronen
  - Aktivstes -> Label
  - 95% auf MNIST
- Supervised learning mit STDP
  - Überwacht: Teaching Signal (force neuron to spike)
  - Evaluieren ohne Teaching Signal
- SPORE (Synaptic Plasticity with Online Reinforcement Learning)
  - Reward-based
  - Rauschen
  - Lernregel zum Lernen neuer Policies
- Spiking networks als Kernel-Methoden
  - Lineare Klassifikatoren um nicht-lineare Probleme zu lösen (Projektion in höheren Raum)
  - Drei Komponenten: Eingabe, Hidden, Ausgabe
  - Trainiert werden die Verbindungen von Hidden zu Output
  - Neural Engineering Framework: Ausgabeschicht wird trainiert w.r.t. Spiking Rates (mit klassischen Algorithmen z.B. RBM -> Komponenten mit Spiking Netzwerk ersetzen)
  - Liquid State Machines: Trainieren w.r.t. post-synaptic Potential, Ausgabe kann schon kommen bevor Eingabe verarbeitet wurde (anytime computing)

7.6 Hardware

- Sehr schnelle Kommunikation zu jeder Zeit nötig (GPUs nicht geeignet - good for parallel not distributed)
- SpiNNaker (ARM), BrainScales (LIT), TrueNorth (porting trained deep nets,

8 Objektorientierte Probabilistisch Relationale Modelle

8.1 Grundlagen

8.1.1 Bayes'sche Netze (BNs)

- Knoten: Zufallsvariablen
- Kanten: Bedingte Abhängigkeiten
- Graph-Struktur definiert Verbundverteilungen und beinhaltet Aussagen über (bedingte) Unabhängigkeiten der Zufallsvariablen
- Zu jedem Knoten X_i existiert eine bedingte Wahrscheinlichkeitsverteilung $P(X_i|Pa_i)$ = Verteilung über X_i für jeden Elternknoten Pa_i
- Wesentlicher Vorteil BN = kompakte Repräsentation: Gegeben n Knoten (binäre Zufallsvariablen), Knoten haben $\leq k$ Eltern -> $2^k n$ vs 2^n Parameter
- Inferenz: BN definiert über Verbundwahrscheinlichkeit Lösung zu verschiedenen Anfragen: P(Variable|Aussage über andere)
- BN Modelle können anhand empirischer Daten gelernt werden
  - Vollständige Daten: Parameterschätzung durch numerische Optimierung, Strukturlernen durch kombinatorische Suche (NP-vollständig)
  - Unvollständige Daten: Expectation Maximization (EM) und Structural EM

8.1.2 Probabilistische Relationale Modelle (PRMs)

- Zusammenführen der Vorteile relationer Logik & Bayes'scher Netze
- Vorteile der Prädikatenlogik
  - Natürliche Domänenmodellierung: Entitäten, Eigenschaften, Relationen
  - Möglichkeit zur Generalisierung über viele Instanzen derselben Domäne Schließen über Klassen von Objekte (Entitäten)
  - Erlaubt kompakte Beschreibung
- Integrieren von Unsicherheit mit relationalem Modell
  - Kompakte, natürliche probabilistische Modelle

  - Eigenschaften von Entitäten der Domäne können auf Eigenschaften von anderen Entitäten beruhen
  - Unsicherheit über relationale Struktur der Domäne
- Klassenmenge $X = X_1, ..., X_n$ (auch Menge der Entitäten genannt)
- Für jede Klasse X Menge von Attributen X, A, Wertebereich $V(X, A)$ für jedes Attribut, Attribute können Zufallsvariablen sein
- Eine Instanz I (der Welt) definiert für jede Klasse X die Obejkte der Klasse $I(X) = o_1, ..., o_n$ (die zu betrachten sind)
- Relationales Schema
  - Beschreibt Klassen von Objekten (und Attribute) und Relationen
  - Entspricht der Prädikatenlogik
- Probabilistisches Relationales Modell: Beschreibung der Attribute z.B. als Zufallsvariablen und bedingte Abhängigkeiten im relationalen Schema
- Relationales Skelett
  - Das relationale Skelett σ definiert für eine Instanz: Menge an Objekten jeder Klasse und Abhängigkeiten zwischen diesen
  - Unsicherheit über Attribute (Zufallsvariablen)
  - PRM definiert für entsprechende σ die Wahrscheinlichkeitsverteilungen (über alle Instanzen) der Attribute
  - PRM + Skelett = Äquivalenz zu einem entsprechenden Bayes'schen Netz
- PRM Inferenz
  - Über konstruiertes, äquivalentes BN (<- PRM+Skelett) -> kann sehr groß werden, d.h. nicht effizient, Struktur wird vernachlässigt
  - Besser: Structured Variable Elimination (SVE): Nutzen der Kapselung und Wiederverwendung, Strukturmodell gibt die Kapselung, Wiederverwendung der inferierten Teilgraphen wenn Evidenzen gleich bleiben
  - PRM mit mehreren Objekt-Instanzen -> Abhängigkeiten aggregieren
  - Lernen von PRMs mit Attributunsicherheit
    * Wahrscheinlichkeit einer Instanz I der Welt: $P(I|\sigma, S, \Theta) = \prod_{x \in \sigma} \prod_{x.A} P(x.A|parents_{S,\sigma}(x.A))$
    * Verwenden von Lerndaten um Parameter (i.A. lokale Wahrscheinlichkeiten) zu Schätzen und Struktur zu bestimmen
    * Parameterschätzung in PRMs
      · Annahme, dass Abhängigkeitsstruktur S bekannt ist und Daten vollständig beobachtbar sind

- Ziel: Schätzung der PRM Parameter $\theta_{x.A|parents(x.A)}$
- θ ist gut, wenn es wahrscheinlich ist, dass die beobachteten Daten der Instanz I erzeugt $l(\theta : I, S) = logP(I|S, \theta)$
- MLE Prinzip: Wähle θ^* so, dass l maximal ist $\theta^* = \frac{N(C.T=f, P.H=f, C.C=t)}{N(P.H=f, C.C=t)}$, vollständige Daten: Schätzung basierend auf Häufigkeit in Lerndaten

9 Gaussian Processes