Learning, Inference, and Planning

INF581 Advanced Machine Learning and Autonomous Agents

Jesse Read

Last Updated: January 15, 2025

Goals for Today

Last week: Complete pipeline for an agent, from perception to action, $s \mapsto a$, via probabilistic reasoning (on P, as a Bayesian network) and decision making ($\max Q(a)$). In brief:

$$a_* = \pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \underbrace{\mathbb{E}_{S \sim P(S)}[r(S, a)]}_{Q(a)}$$

Problem 1: Where did the knowledge P come from?

Problem 2: What if $\mathcal{A}$ and/or $\mathcal{S}$ is large/complex?

Objectives:

1. learning strategies for P (focus on 'meta-strategies'; assume you know about neural networks), a look at the overlap between learning and inference, practical considerations
2. study search algorithms for $\mathcal{A}$; a set of tools, useful for inference, planning; 'setting up' the problem (state, observations, reward, ...), trade-offs.

Agenda for this Lecture

1. Multi-Label Classification
2. Classifier Chains
3. Loss Metrics
4. Inference as Search
5. Planning as Search
6. Learning as Search
7. Learning and Inference Together
8. Summary

Using multi-label classification and classifier chains (Bayesian networks again) as a platform, we move to advanced inference, autonomous agents and planning.

From Richard Sutton[1]:

> *As a field [...] we are continuing to make the same kind of mistakes. [...] The bitter lesson is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning. [...]*
>
> *One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great.* ***The two methods that seem to scale arbitrarily in this way are search and learning****.*

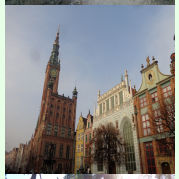
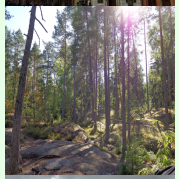
(my emphasis, JR)

Keywords for today: Searching and Learning.

[1] `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`

Multi-Label Classification

Multi-Label Image Classification

Input	Beach	Sunset	Foliage	Urban
	1	0	1	0
	0	1	0	0
	0	1	0	1
	0	1	1	0
	0	0	1	1
	?	?	?	?

Given an instance $\boldsymbol{x}$, we obtain predictions $\widehat{\boldsymbol{y}} = h(\boldsymbol{x})$.

Multi-Label Text (Media/...) Classification

The Lord of the Rings: The Fellowship of the Ring (2001)

Top 500

PG-13 | 178 min | Adventure, Fantasy | 19 December 2001 (USA)

8.8

Your rating: -/10

Ratings: **8.8**/10 from 1,110,948 users Metascore: 92/100

Reviews: 4,988 user | 294 critic | 34 from Metacritic.com

A meek hobbit of the Shire and eight companions set out on a journey to Mount Doom to destroy the One Ring and the dark lord Sauron.

Director: Peter Jackson

Writers: J.R.R. Tolkien (novel), Fran Walsh (screenplay), 2 more credits »

Stars: Elijah Wood, Ian McKellen, Orlando Bloom |

See full cast and crew »

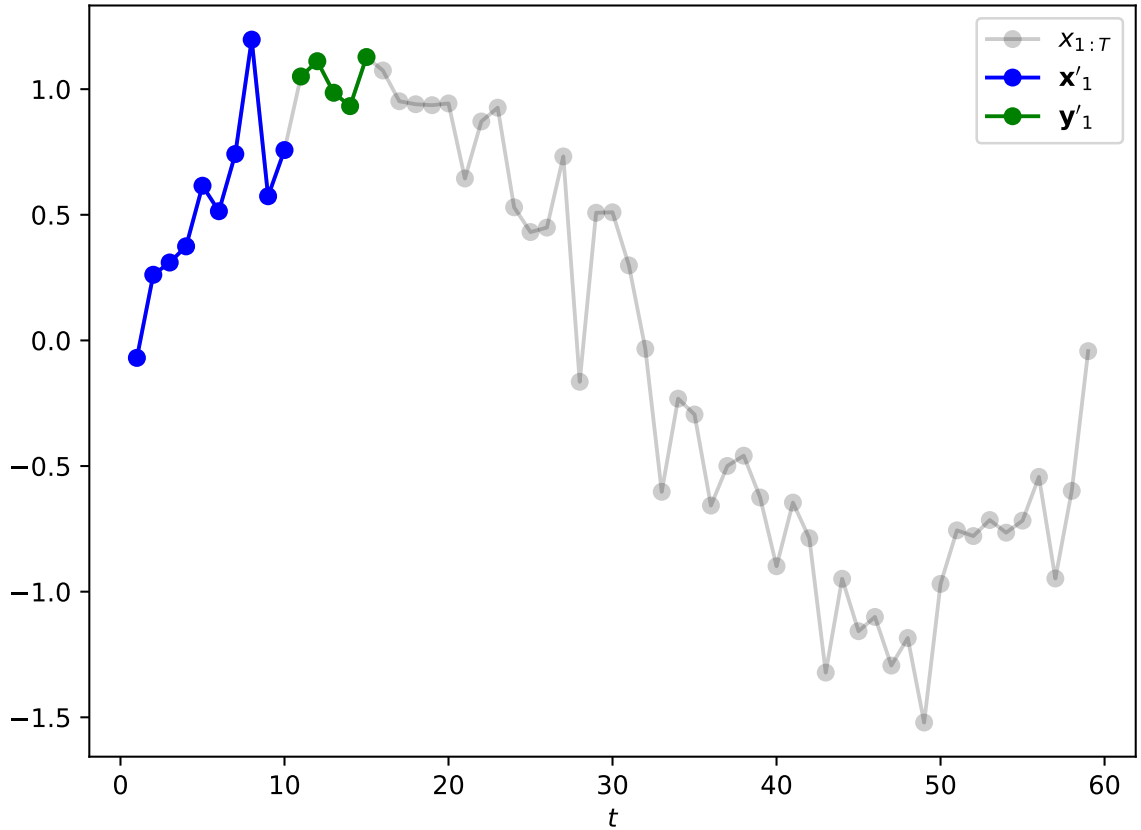
Labels as Keywords

Source: [1]

Missing-data Imputation / Recommender Systems

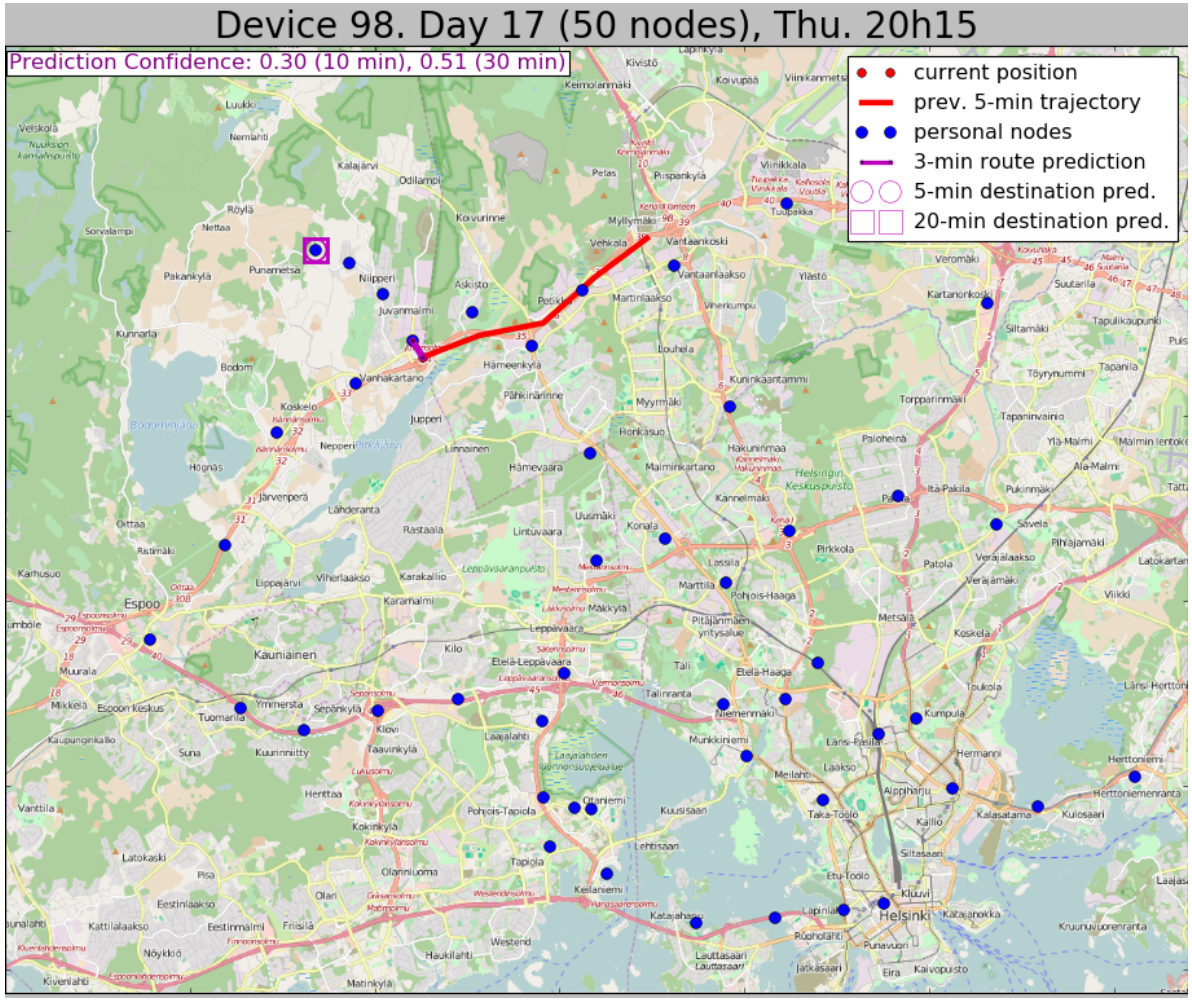
	Film 2	Film 3	Book 1	Book 2	Song 5
	0	0	1	1	0
	1	1	?	0	?
	0	0	1	0	0
	1	1	?	0	1
	0	0	0	?	?
	1	0	?	1	?

Time Series Forecasting



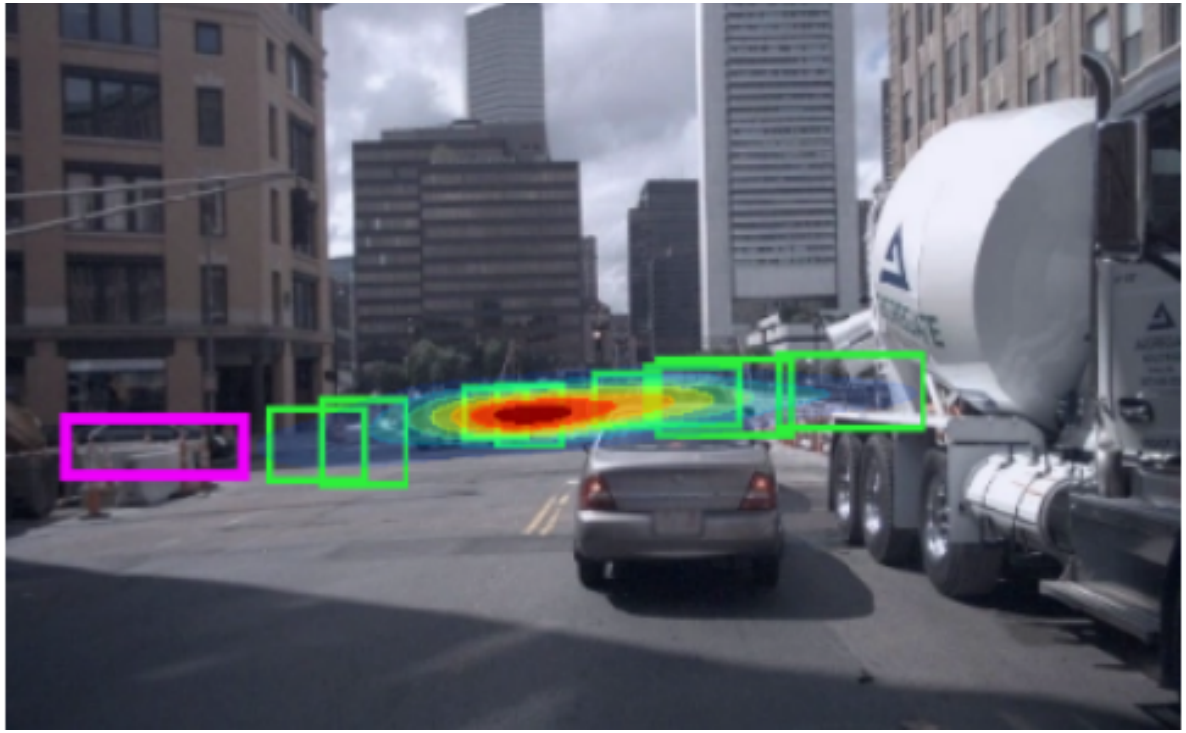
(including multi-dimensional time series; generalization to $\boldsymbol{y} \in \mathbb{R}^m$).

Trajectory Prediction



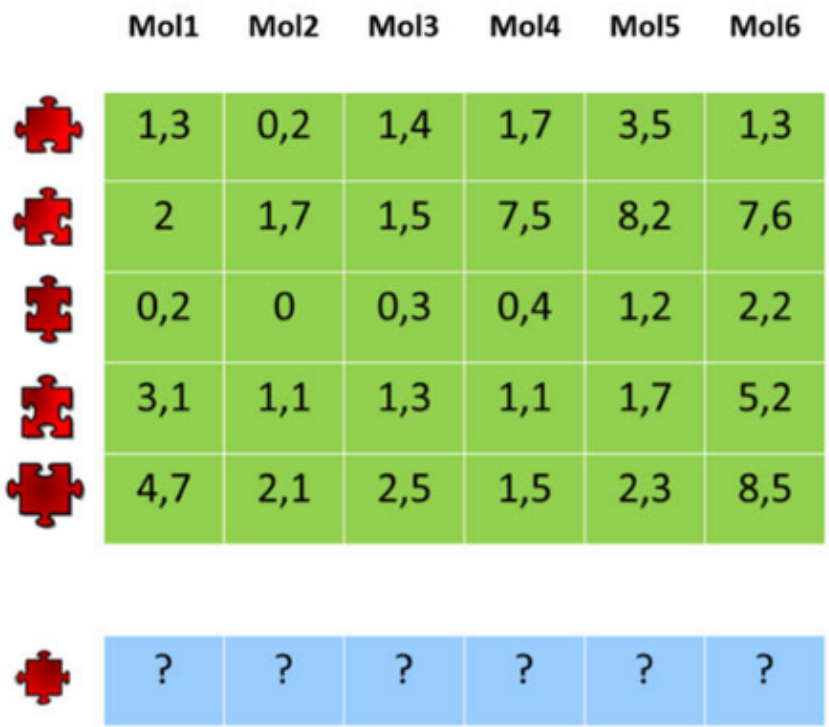
Trajectory prediction in urban environment using mobile phone data

Localization and Bounding in Images



Object prediction: [2]

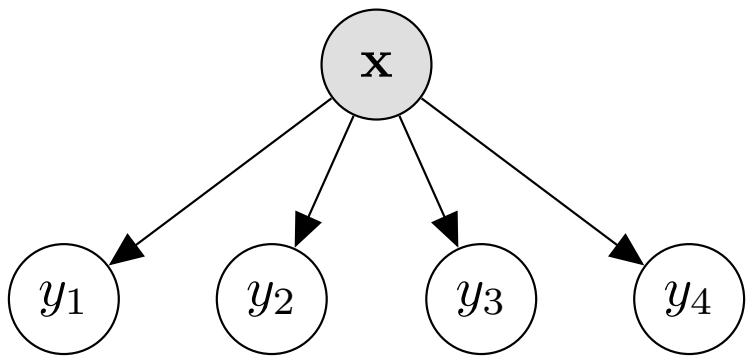
Drug Design



Molecule design prediction (binding affinities (Y) of molecules (X) to new proteins): [3]

Independent Models: The Baseline

$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	0
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	0
$\boldsymbol{x}^{(4)}$	1	0	0	1
$\boldsymbol{x}^{(5)}$	0	0	0	1
$\tilde{\boldsymbol{x}}$	?	?	?	?



One model classifier trained for each label; independent models.

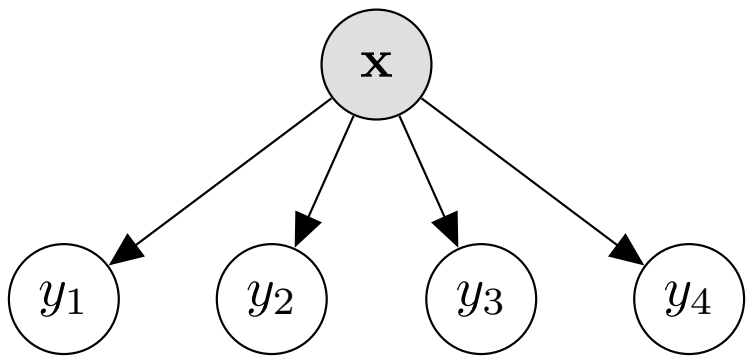
Independent Models: The Baseline

$\boldsymbol{X}$	Y_1
$\boldsymbol{x}^{(1)}$	0
$\boldsymbol{x}^{(2)}$	1
$\boldsymbol{x}^{(3)}$	0
$\boldsymbol{x}^{(4)}$	1
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?

$\boldsymbol{X}$	Y_2
$\boldsymbol{x}^{(1)}$	1
$\boldsymbol{x}^{(2)}$	0
$\boldsymbol{x}^{(3)}$	1
$\boldsymbol{x}^{(4)}$	0
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?

$\boldsymbol{X}$	Y_3
$\boldsymbol{x}^{(1)}$	0
$\boldsymbol{x}^{(2)}$	1
$\boldsymbol{x}^{(3)}$	0
$\boldsymbol{x}^{(4)}$	1
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?

$\boldsymbol{X}$	Y_4
$\boldsymbol{x}^{(1)}$	1
$\boldsymbol{x}^{(2)}$	0
$\boldsymbol{x}^{(3)}$	1
$\boldsymbol{x}^{(4)}$	0
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	?

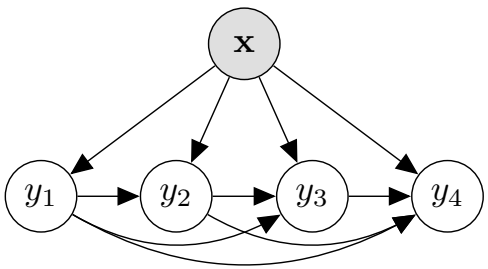


One model classifier trained for each label; independent models.

Classifier Chains

Classifier Chains (and Greedy Inference)

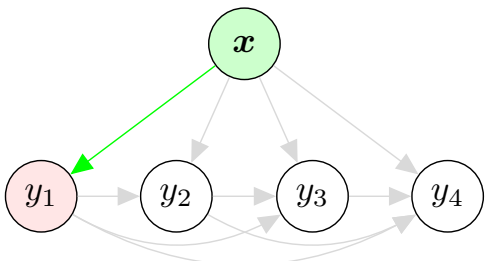
A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. In is also a neural network. Each $P(Y_j \mid \text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y_1}$	$\widehat{y_2}$	$\widehat{y_3}$	$\widehat{y_4}$

Classifier Chains (and Greedy Inference)

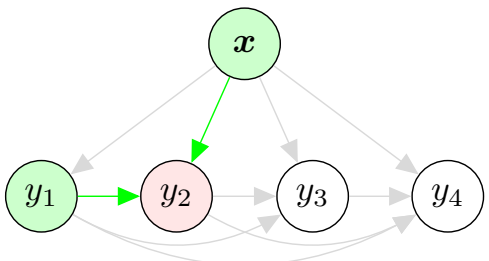
A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. In is also a neural network. Each $P(Y_j \mid \text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y_1}$			

Classifier Chains (and Greedy Inference)

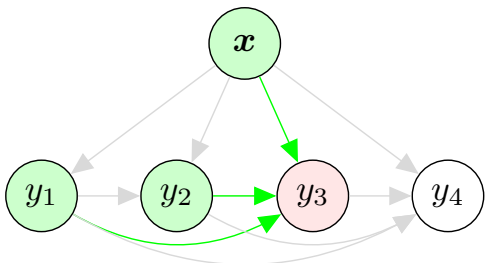
A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. In is also a neural network. Each $P(Y_j \mid \text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y_1}$	$\widehat{y_2}$		

Classifier Chains (and Greedy Inference)

A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. In is also a neural network. Each $P(Y_j \mid \text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$	

For example,

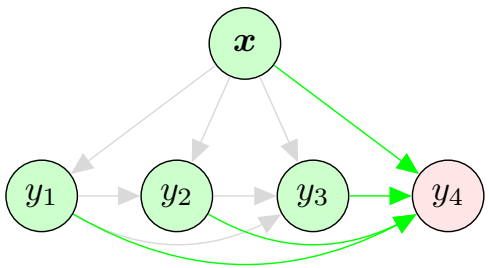
$$\widehat{y}_3 = h_3(\boldsymbol{x}, \widehat{y}_1, \widehat{y}_2) = \underset{y_3 \in \{0,1\}}{\text{argmax}}\, P(y_3 | \boldsymbol{x}, \widehat{y}_1, \widehat{y}_2)$$

Use training data to fit base classifier h_3.

Inference: $\widehat{y}_1, \widehat{y}_2, \ldots$ are greedy *predictions* from $h_1, h_2, \ldots$.

Classifier Chains (and Greedy Inference)

A classifier chain is a Bayesian network, with a fully-cascaded structure across output variables. In is also a neural network. Each $P(Y_j \mid \text{pa}(Y_j))$ is provided by a base classifier (e.g., logistic regression, decision trees, neural network, . . .). Motivation: Model label dependence with structure.



$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	1
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	1
$\boldsymbol{x}^{(4)}$	1	0	0	0
$\boldsymbol{x}^{(5)}$	0	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$	$\widehat{y}_4$

For example,

$$\widehat{y}_3 = h_3(\boldsymbol{x}, \widehat{y}_1, \widehat{y}_2) = \underset{y_3 \in \{0,1\}}{\text{argmax}}\, P(y_3 | \boldsymbol{x}, \widehat{y}_1, \widehat{y}_2)$$

Use training data to fit base classifier h_3.

Inference: $\widehat{y}_1, \widehat{y}_2, \ldots$ are greedy *predictions* from $h_1, h_2, \ldots$.

Recap: At training time, a multi-label dataset is transformed into m standard binary classification problems:

$\boldsymbol{X}$	Y_1	Y_2	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	0
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	0
$\boldsymbol{x}^{(4)}$	1	0	0	1
$\boldsymbol{x}^{(5)}$	0	0	0	1
$\tilde{\boldsymbol{x}}$	?	?	?	?

At test time, each base classifier h_j provides $\widehat{y}_j \in \{0, 1\}$:

$$\widehat{y}_1 = h_1(\tilde{\boldsymbol{x}}) \quad \triangleright \text{ for } j = 1$$
$$\widehat{y}_j = h_j(\tilde{\boldsymbol{x}}, \widehat{y}_1, \ldots, \widehat{y}_{j-1}) \quad \triangleright \text{ for } j = 2, \ldots, m$$

Multi-output classification:

$$\widehat{\boldsymbol{y}} = [\widehat{y}_1, \ldots, \widehat{y}_m]$$

Recap: At training time, a multi-label dataset is transformed into m standard binary classification problems:

$\boldsymbol{X}$	Y_1
$\boldsymbol{x}^{(1)}$	0
$\boldsymbol{x}^{(2)}$	1
$\boldsymbol{x}^{(3)}$	0
$\boldsymbol{x}^{(4)}$	1
$\boldsymbol{x}^{(5)}$	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$

$\boldsymbol{X}$	Y_1	Y_2
$\boldsymbol{x}^{(1)}$	0	1
$\boldsymbol{x}^{(2)}$	1	0
$\boldsymbol{x}^{(3)}$	0	1
$\boldsymbol{x}^{(4)}$	1	0
$\boldsymbol{x}^{(5)}$	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$

$\boldsymbol{X}$	Y_1	Y_2	Y_3
$\boldsymbol{x}^{(1)}$	0	1	1
$\boldsymbol{x}^{(2)}$	1	0	0
$\boldsymbol{x}^{(3)}$	0	1	0
$\boldsymbol{x}^{(4)}$	1	0	0
$\boldsymbol{x}^{(5)}$	0	0	0
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$

$\boldsymbol{X}$	Y_1	Y_3	Y_3	Y_4
$\boldsymbol{x}^{(1)}$	0	1	1	0
$\boldsymbol{x}^{(2)}$	1	0	0	0
$\boldsymbol{x}^{(3)}$	0	1	0	0
$\boldsymbol{x}^{(4)}$	1	0	0	1
$\boldsymbol{x}^{(5)}$	0	0	0	1
$\tilde{\boldsymbol{x}}$	$\widehat{y}_1$	$\widehat{y}_2$	$\widehat{y}_3$	$\widehat{y}_4$

At test time, each base classifier h_j provides $\widehat{y}_j \in \{0, 1\}$:

$$\widehat{y}_1 = h_1(\tilde{\boldsymbol{x}}) \quad \triangleright \text{ for } j = 1$$
$$\widehat{y}_j = h_j(\tilde{\boldsymbol{x}}, \widehat{y}_1, \ldots, \widehat{y}_{j-1}) \quad \triangleright \text{ for } j = 2, \ldots, m$$

Multi-output classification:

$$\widehat{\boldsymbol{y}} = [\widehat{y}_1, \ldots, \widehat{y}_m]$$

Classifier Chains with Bayes Optimal Inference

Each base classifier has a probabilistic interpretation:

$$\widehat{y}_j = h_j(\boldsymbol{x}) = \underset{y_j \in \{0,1\}}{\operatorname{argmax}} P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})$$

(e.g., logistic regression).

From Bayesian network:

$$\begin{aligned}
\widehat{\boldsymbol{y}} &= \underset{\boldsymbol{y} \in \{0,1\}^m}{\operatorname{argmax}} P(\boldsymbol{y} | \boldsymbol{x}) \\
&= \underset{\boldsymbol{y} \in \{0,1\}^m}{\operatorname{argmax}} P(y_1 | \boldsymbol{x}) \prod_{j=2}^{m} P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})
\end{aligned}$$

This is not the same as greedy inference.

Classifier Chains with Bayes Optimal Inference

Each base classifier has a probabilistic interpretation:

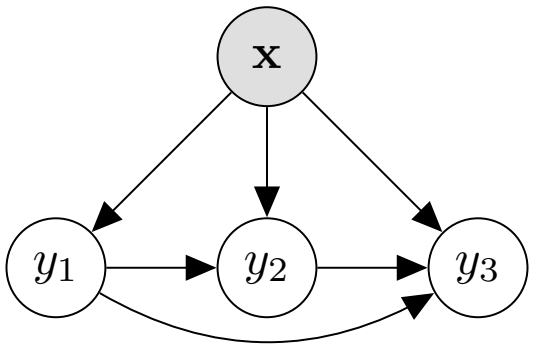
$$\widehat{y}_j = h_j(\boldsymbol{x}) = \underset{y_j \in \{0,1\}}{\operatorname{argmax}} P(y_j | \boldsymbol{x}, y_1, \ldots, y_{j-1})$$

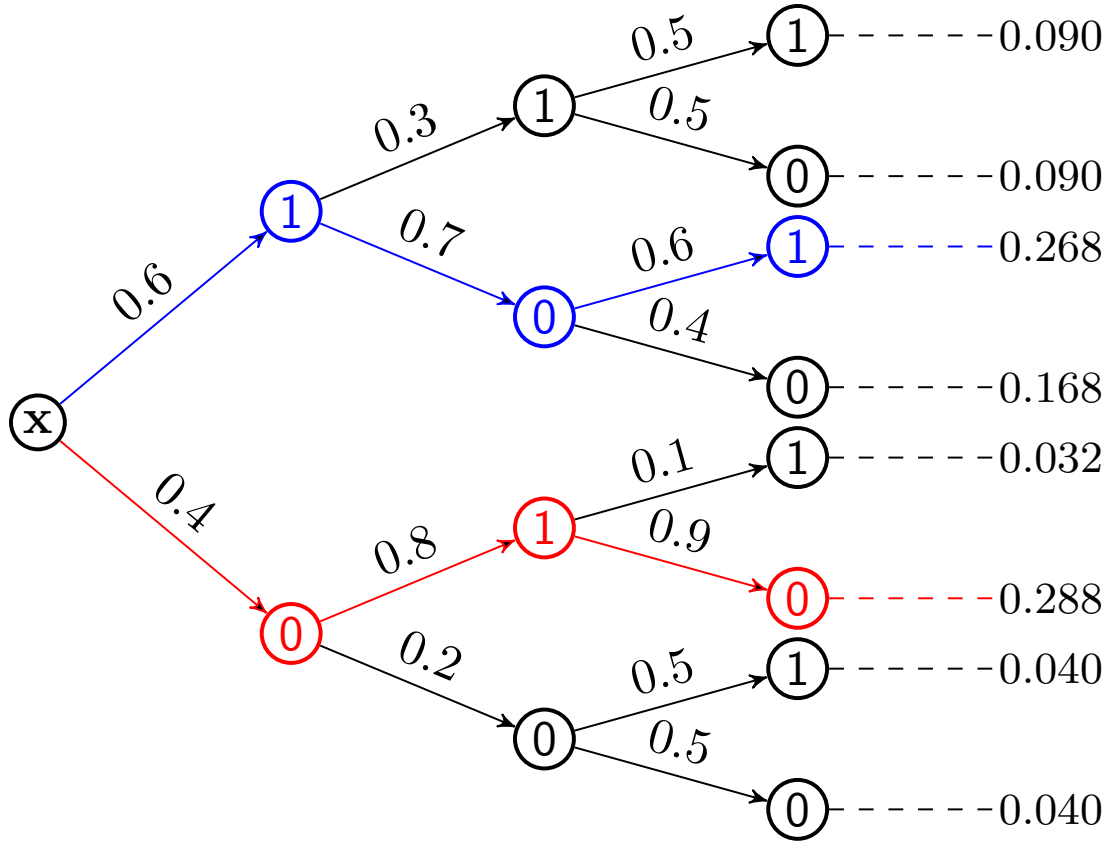
(e.g., logistic regression).

From Bayesian network:

$$\begin{aligned}\widehat{\boldsymbol{y}} &= \underset{\boldsymbol{y} \in \{0,1\}^m}{\operatorname{argmax}} P(\boldsymbol{y}|\boldsymbol{x}) \\ &= \underset{\boldsymbol{y} \in \{0,1\}^m}{\operatorname{argmax}} P(y_1|\boldsymbol{x}) \prod_{j=2}^{m} P(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1})\end{aligned}$$

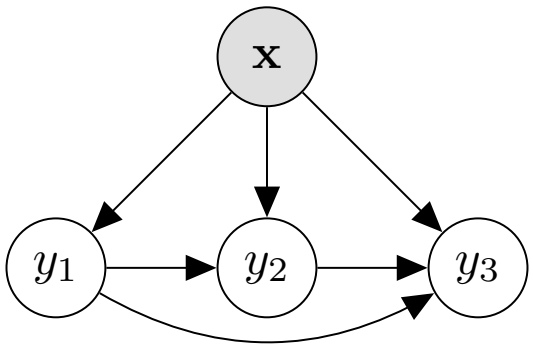
This is not the same as greedy inference.
First problem: Exponential complexity.

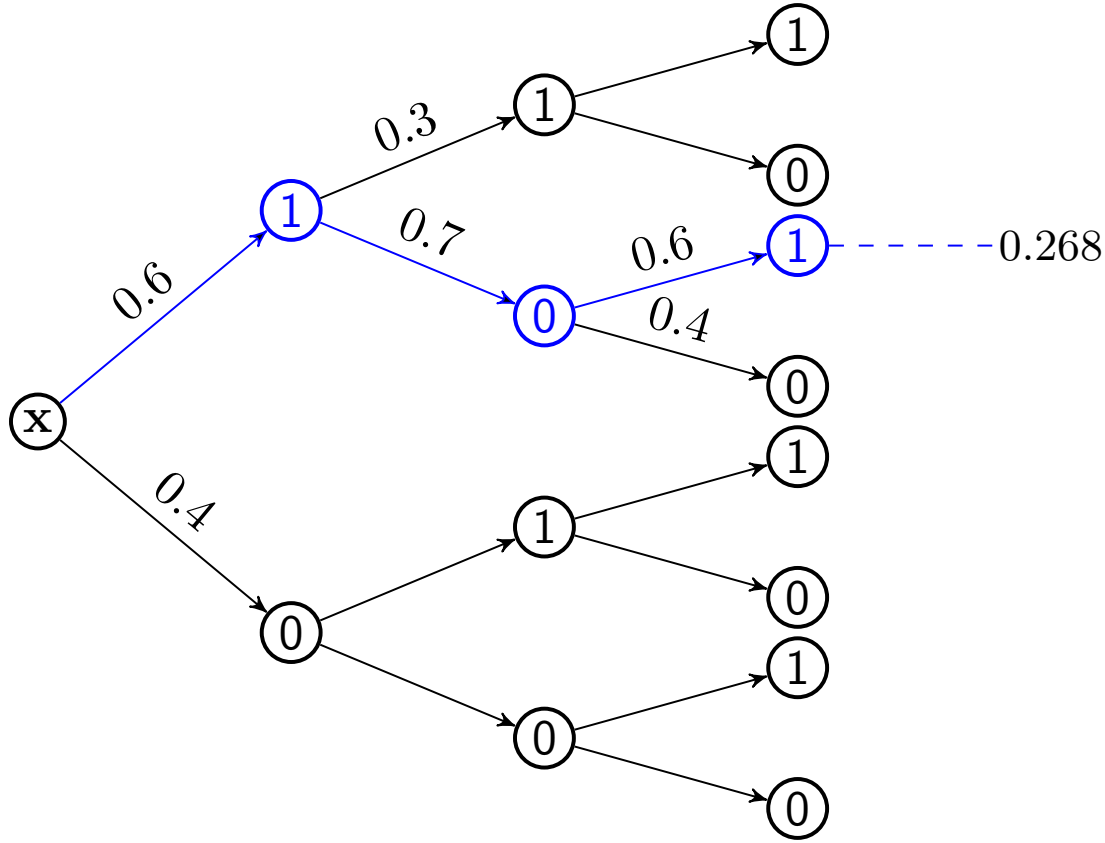




e.g., $P([1,0,1]|\boldsymbol{x}) = 0.62 \cdot 0.72 \cdot 0.60 = 0.268$

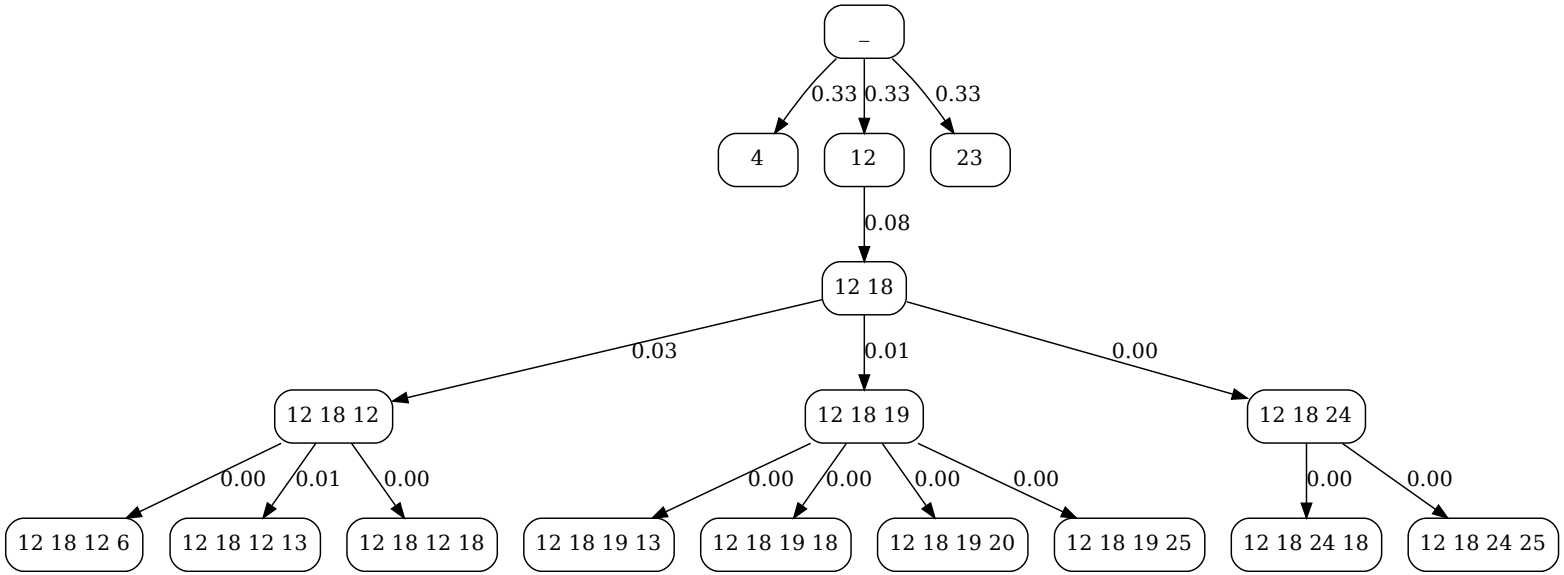
- Exhaustive search $\in \{0,1\}^m$, i.e., 2^m paths,
  Bayes optimal, usually intractable
- Greedy search, $m \times \{0,1\}$, i.e., 1 path,
  Fast, but not optimal.

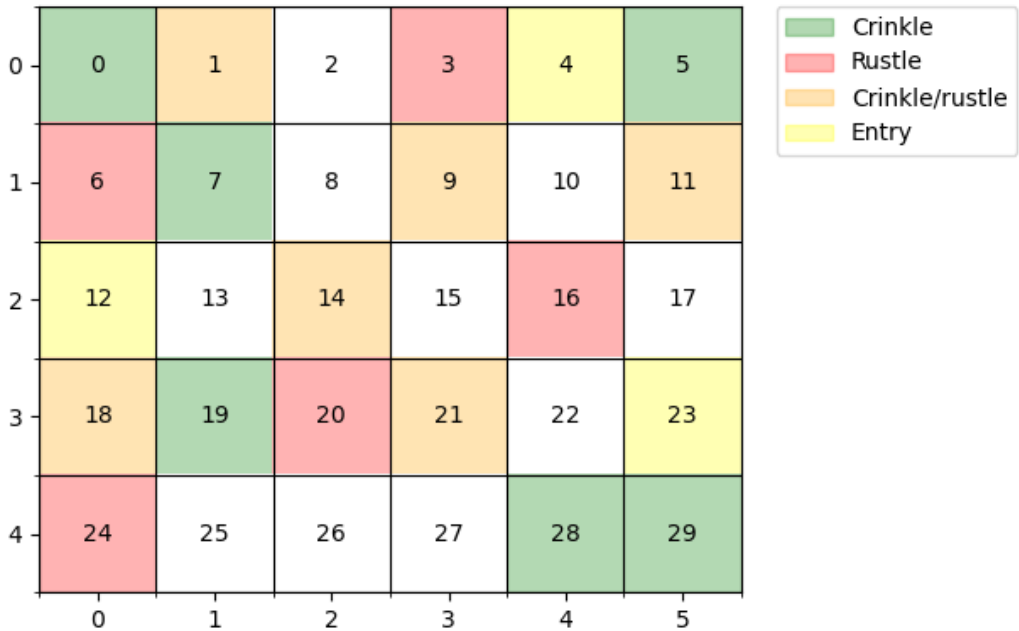




e.g., $P([1,0,1]|\boldsymbol{x}) = 0.62 \cdot 0.72 \cdot 0.60 = 0.268$

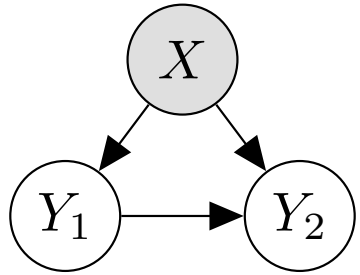
- Exhaustive search $\in \{0,1\}^m$, i.e., 2^m paths, Bayes optimal, usually intractable
- Greedy search, $m \times \{0,1\}$, i.e., 1 path, Fast, but not optimal.


_
0.33
0.33
0.33
4
12
23
0.08
12 18
0.03
0.01
0.00
12 18 12
12 18 19
12 18 24
0.00
0.01
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
12 18 12 6
12 18 12 13
12 18 12 18
12 18 19 13
12 18 19 18
12 18 19 20
12 18 19 25
12 18 24 18
12 18 24 25


Crinkle
Rustle
Crinkle/rustle
Entry
0
1
2
3
4
5
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
0
1
2
3
4
5

Loss Metrics

First Question: Are Independent Models Sufficient?



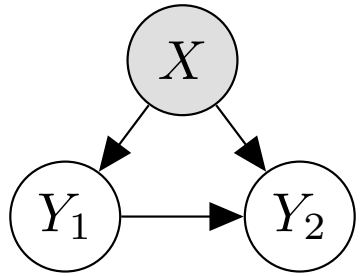
Suppose we want to predict and evaluate under Hamming loss (sum of errors, $\sum_{j=1}^{m} \ell(y_j, \widehat{y}_j)$). For a given label, e.g., y_2:

$$\ell(y_2, \widehat{y}_2) = \begin{cases} 1 & y_2 \neq \widehat{y}_2, \\ 0 & y_2 = \widehat{y}_2 \end{cases}$$

y_1 does not appear in our query, nor as evidence $\widehat{y}_1$, so,

$$P(y_2|\boldsymbol{x}) = \sum_{y_1 \in \{0,1\}} P(y_1|\boldsymbol{x}) P(y_2|\boldsymbol{x}, y_1)$$

i.e., we can model $P(y_2|\boldsymbol{x})$ however we want, directly from $\boldsymbol{x}$. $P(y_1, y_2|\boldsymbol{x})$ is not required!



Now suppose we are minimizing subset 0/1 loss,

$$\ell_{0/1}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \begin{cases} 1 & \boldsymbol{y} \neq \widehat{\boldsymbol{y}}, \\ 0 & \boldsymbol{y} = \widehat{\boldsymbol{y}} \quad \text{(exactly, i.e., } \sum_{j=1}^{m} \ell(y_j, \widehat{y}_j) = 0\text{)} \end{cases}$$

Example:

$P(y_1, y_2)$	$y_1 = 0$	$y_1 = 1$
$y_2 = 0$	0.00	0.50
$y_2 = 1$	0.50	0.00

$P(Y_2 = 1|\boldsymbol{x}) = 0.5$, but $P(Y_2 = 1|Y_1 = 1, \boldsymbol{x}) = 0$!

If we ignore the joint (use independent classifiers), we get worst case loss (of 1)! We need to model label dependence!

Decision Theory in Multilabel Learning

Recall: optimal strategy (rational agent): minimize [conditional] expected loss (also known as risk[2]),

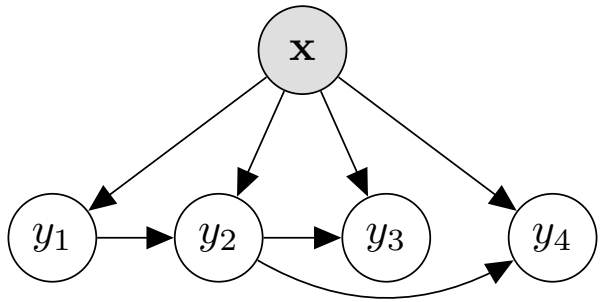
$$\begin{aligned}\widehat{\boldsymbol{y}} = h(\boldsymbol{x}) &= \operatorname*{argmin}_{\boldsymbol{y}' \in \mathcal{Y}} \mathbb{E}_{\boldsymbol{Y} \sim P(\boldsymbol{Y}|\boldsymbol{x})}[\ell(\boldsymbol{Y}, \boldsymbol{y}')|\boldsymbol{x}] \\ &= \operatorname*{argmin}_{\boldsymbol{y}' \in \mathcal{Y}} \sum_{\boldsymbol{y} \in \mathcal{Y}} \ell(\boldsymbol{y}, \boldsymbol{y}') P(\boldsymbol{y}|\boldsymbol{x})\end{aligned}$$

Note: We assume $P \approx P_\star$.

[2] Recall: risk is the opposite of value (expected reward); therefore minimizing risk *is equivalent to maximizing value*

Minimizing 0/1 loss = MAP estimate

To minimize $\ell_{0/1}(\cdot, \widehat{\boldsymbol{y}})$ loss of decision $\widehat{\boldsymbol{y}} = [\widehat{y}_1, \ldots, \widehat{y}_4]$, where $y_j \in \{0, 1\}$, given knowledge P:



our decision function, is a MAP estimate:

$$\widehat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \{0,1\}^4}{\operatorname{argmax}} P(\boldsymbol{y}|\boldsymbol{x}) = h(\boldsymbol{x})$$

Exact inference requires:

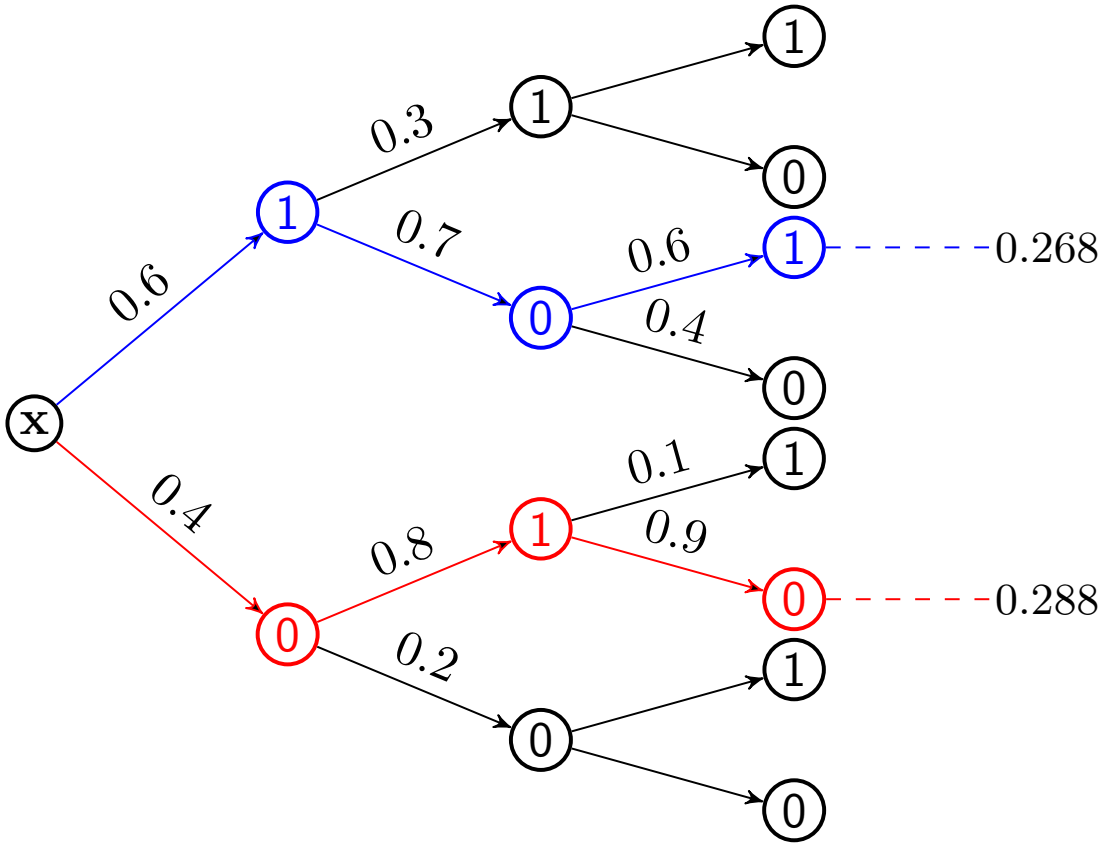
$$\boldsymbol{y} \in \{[0, 0, 0, 0], [0, 0, 0, 1], \ldots, [1, 1, 1, 1]\}$$

and, in general, $\boldsymbol{y} \in \{0, 1\}^m$ for m labels.

Hence: exponential complexity!

Inference as Search

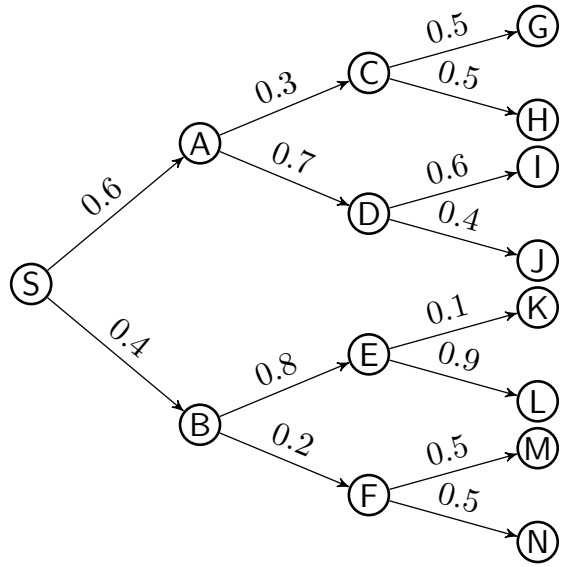
Inference as Tree-Search



- Node (state) $s := [y_1, \ldots, y_j] \in \{0,1\}^j$ (depth j)
- Start state $s = []$
- Accumulated reward: $g(s) = P(y_1, \ldots, y_j \mid \boldsymbol{x})$, which decomposes into edge payoff $P(y_j \mid \boldsymbol{x}, y_1, \ldots, y_{j-1})$
- Maximizing value: $\widehat{\boldsymbol{y}} = \text{argmax}_s\, r(s)$
- Reward only back-propagated from the goal state ($j = m$)

Uniform Cost Search[4] (UCS)

Expanding node n will incur a path reward or gain $g(n)$.



- A priority queue of nodes and respective reward $\{(n, g(n))\}$
- Expand lower-cost[3] nodes first (n giving smallest $g(n)$)
- Goal test is applied *when selected for expansion* rather than first encountered (in case a better path there can be found)
- Optimal in general: solution found is the best

[3] small path cost = big reward = better

[4] also known as Dijkstra's Algorithm, especially when expanding *all* nodes

On weighted graph G, from node n_0 to (goal node) n_{goal}:

UniformCostSearch$(G, n_0, n_{\text{goal}})$:

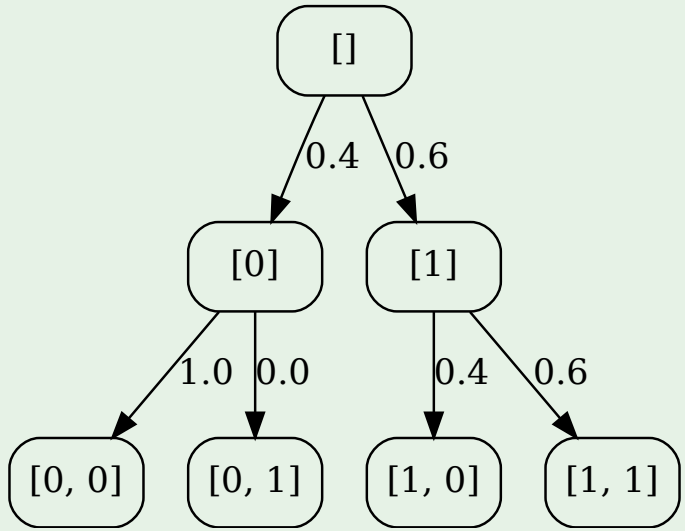
- $\mathcal{Q} = \{(n_0, g(n_0)\}$ ▷ The priority queue
- While $|\mathcal{Q}| > 0$:
  - $n \leftarrow \mathcal{Q}.\text{pop}()$ ▷ Pop[5] n with best $g(n)$
  - if $n = n_{\text{goal}}$, stop!
  - for $n' \in \text{Neighbours}(n)$:
    - $\mathcal{Q} \leftarrow \mathcal{Q} \cup \left(n', g(n')\right)$

(N.B. Usually also want to keep track of the path).

[5] where

$$n' \leftarrow \mathcal{Q}.\text{pop}() \Leftrightarrow \begin{cases} n' = \text{argmax}_{n \in \mathcal{Q}}\, g(n), \\ \mathcal{Q} \leftarrow \mathcal{Q} \setminus n' \end{cases}$$

Uniform Cost Search for Probability-Tree Search



(recall: *path* cost $g([y_1, \ldots, y_j]) = P(y_j|y_1, \ldots, y_j|\boldsymbol{x})$)

$\mathcal{Q} = \{([], 0)\}$

$\mathcal{Q} = \{([1], 0.6), ([0], 0.4)\}$ ▷ pop []

$\mathcal{Q} = \{([0], 0.4), ([1, 1], 0.36), ([1, 0], 0.24)\}$ ▷ pop [1]

$\mathcal{Q} = \{([0, 0], 0.4), ([1, 1], 0.36), ([1, 0], 0.24), ([0, 1], 0.0)\}$ ▷ pop [0]

Return: $\boldsymbol{y} = [0, 0]$

Beam Search

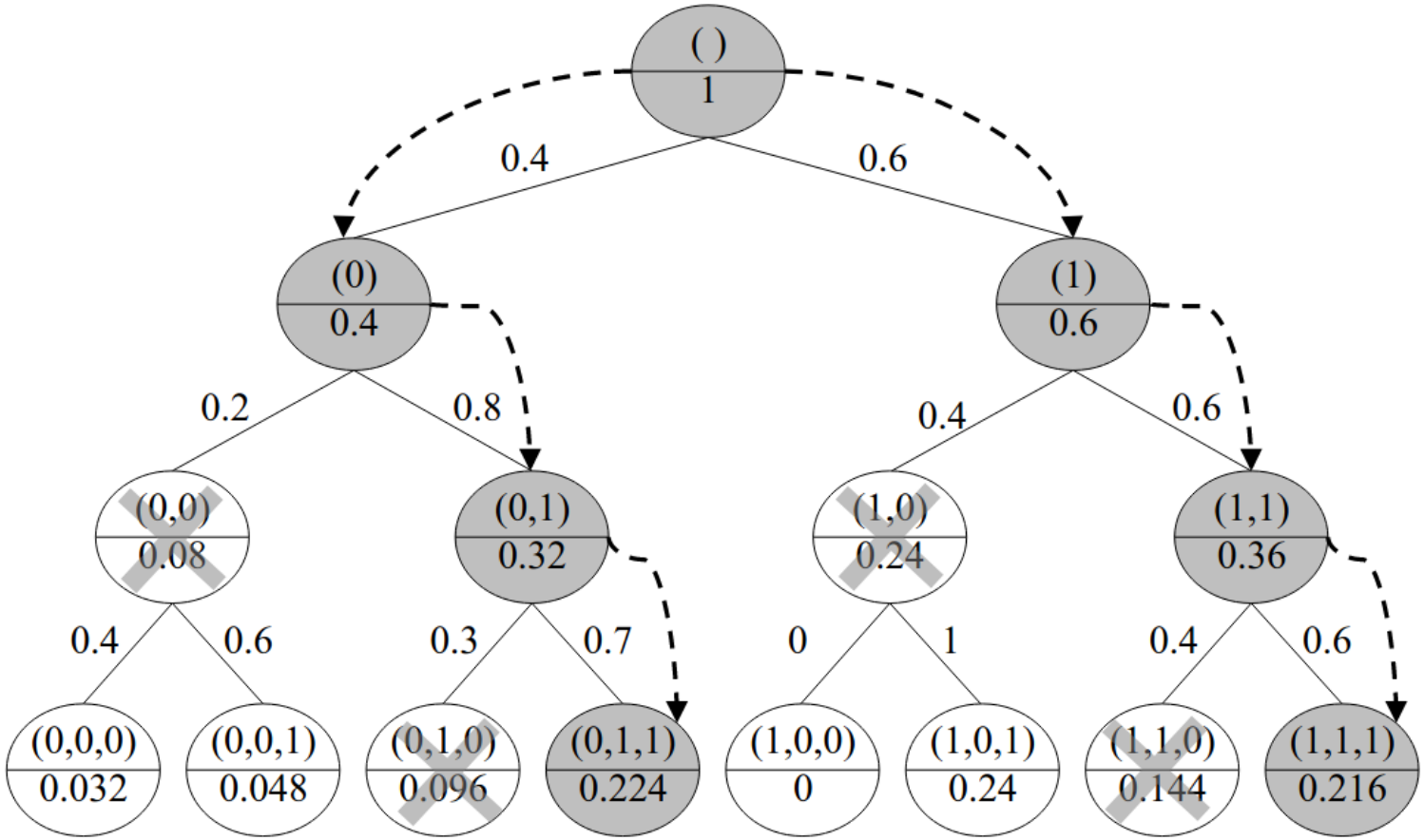
A variant of UCS with reduced memory requirements; only considers the top β nodes (the size of the **beam**) wrt path cost at each level.

- A variant of greedy search.
- Popular in neural networks, especially in language models

Where β is a hyperparameter;

- If $\beta = \infty$, then it is standard UCS

Beam Search, $\beta = 2$



From: Mena et al., *An overview of Inference Methods in Probabilistic Classifier Chains for Multi-label classification*

ϵ-Approximate Search

Proceed like UCS (node $n' = [y_1, \ldots, y_j]$ is added to the queue), but only if, at depth j,

$$\underbrace{P(y_1, \ldots, y_j | \boldsymbol{x})}_{g(n)} > \epsilon$$

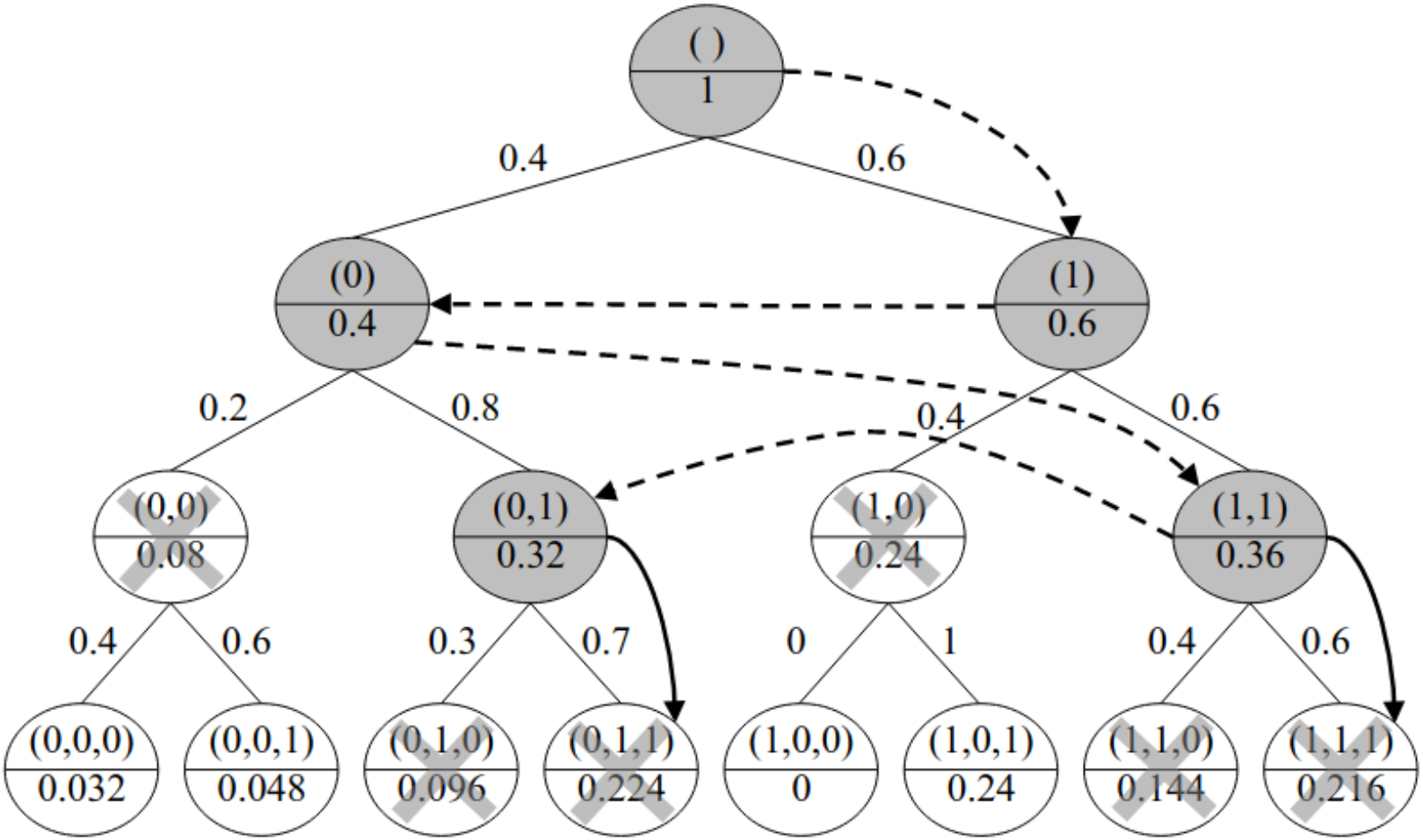
Continue (pop, goal-check, add children), until

- a leaf is found (done!); or
- queue is empty: apply greedy search to goal (from all childless [non-expanded] nodes).

Where ϵ is a hyper-parameter;

- if $\epsilon = 0.0$, then unit costs are ignored; defaults to UCS
- if $\epsilon = 0.5$, defaults to greedy search.

ϵ-Approximate Search, $\epsilon = 0.25$.



From: Mena et al., *An overview of Inference Methods in Probabilistic Classifier Chains for Multi-label classification*

Monte Carlo Search

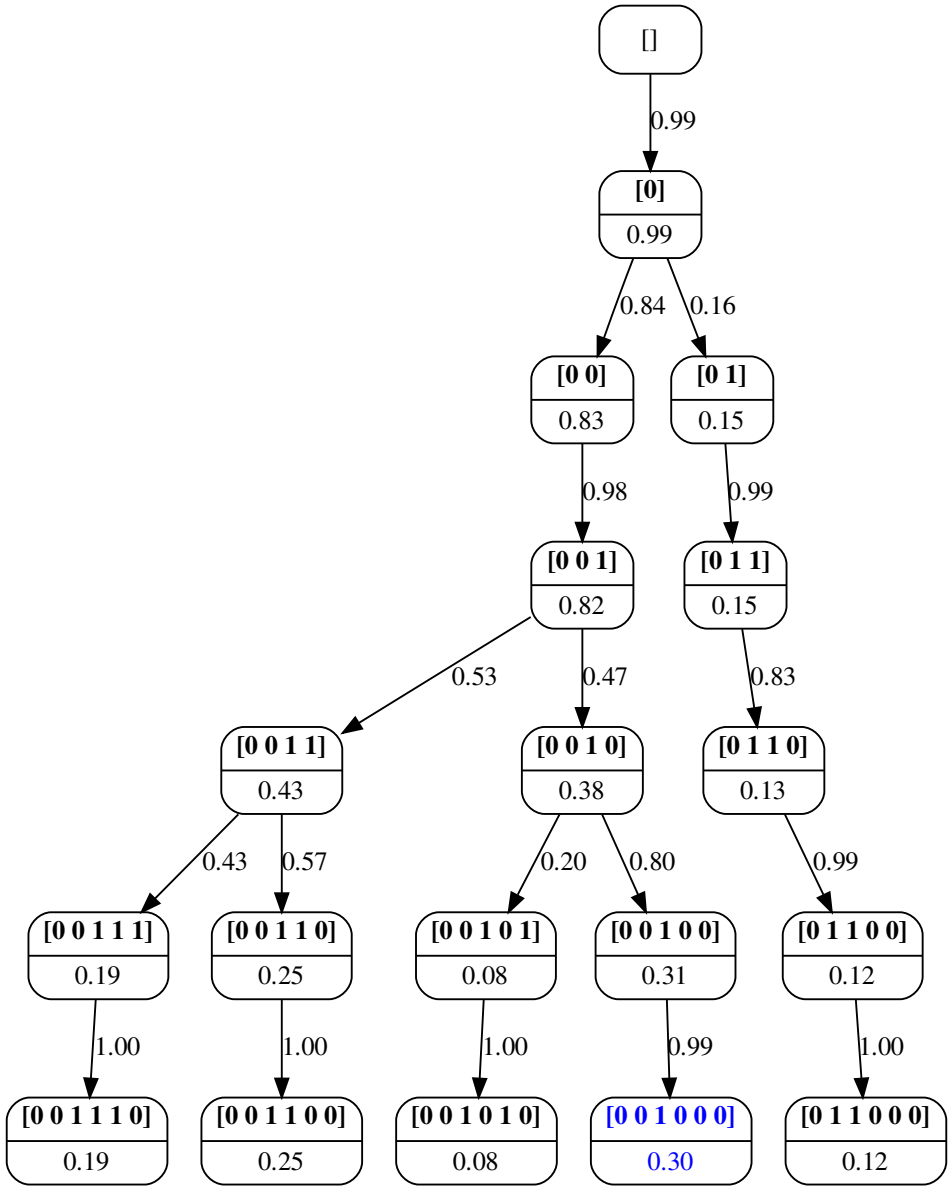
Using the Monte Carlo methodology.

1. Open branch $y_j^{(i)}$ with probability $P(y_j^{(i)} \mid \boldsymbol{x}, y_1^{(i)}, \ldots, y_{j-1}^{(i)})$
2. Repeat, and collect M samples $\{\boldsymbol{y}^{(i)}\}_{i=1}^{M}$
3. Return the i^*-th path; where $i^* = \text{argmax}_i \underbrace{P(\boldsymbol{y}^{(i)}|\boldsymbol{x})}_{g}$.

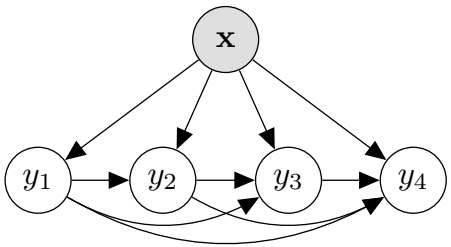
Where M is a hyper-parameter;

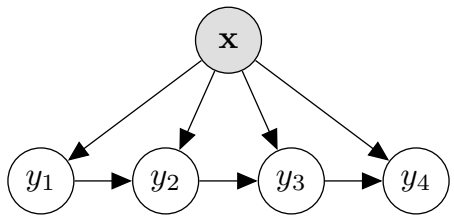
- if $M = \infty$, is optimal

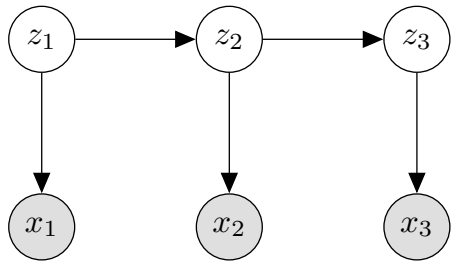
Monte Carlo Search (on the Emotions/Music data; $m = 6$, on a single test point $\boldsymbol{x}$), $T = 30$.



Suboptimal results







- Chance of finding Bayes optimal solution depends on $\beta/\epsilon/T$
- Other search methods are applicable, e.g., A* search, ...
- Monte Carlo search is the only one directly applicable to when $\boldsymbol{y} \in \mathbb{R}^m$, and from it we can derive Monte Carlo Tree Search.

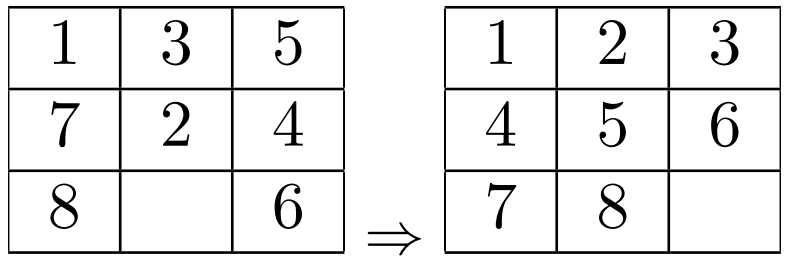
Can be seen as multi-step reasoning (ultra-simplistic version of multi-agent LLMs). Error propagation ('Hallucination') may occur when passing through zone of high entropy.

We're getting close to planning: plan a sequence of labels.

Planning as Search

Reinforcement Learning 0.5: The 8-Puzzle (Example)

Objective: move from



- State s (state of puzzle)
- Action a (slide a tile), changes the state
- Reward, e.g., $r(s, a) = 1$ iff s final state (else 0), so $g = \sum_t r_t$

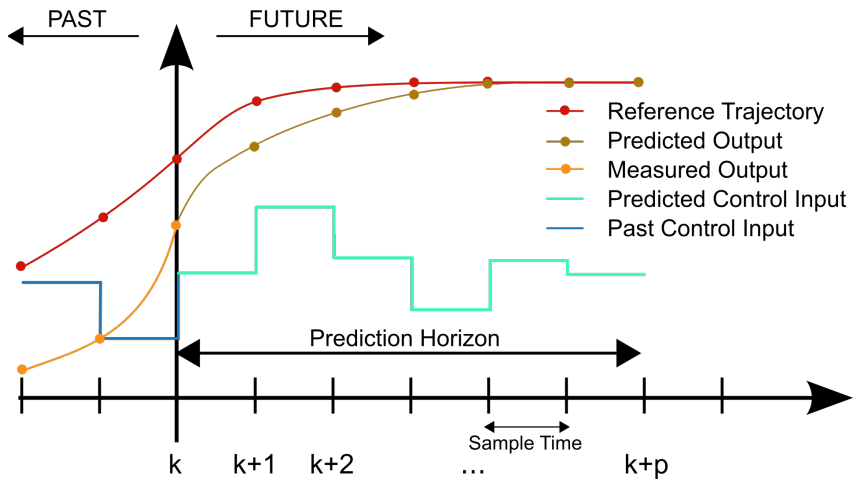
Solve via tree search! 'Planning'!

Applicable in special case:

- assumes knowledge of game rules
- game rules are deterministic
- search space is small

Model Predictive Control (MPC)

Solve an optimisation problem (search space, over a prediction horizon) as part of inference.



Source: Wikipedia. Where Reference is e.g., required output (related to reward); Measured is a measure of success; Control Input are actions (decisions).

MPC is typically used when we already have a model of system dynamics, often linear (the optimisation is easier). Well-used in industry.

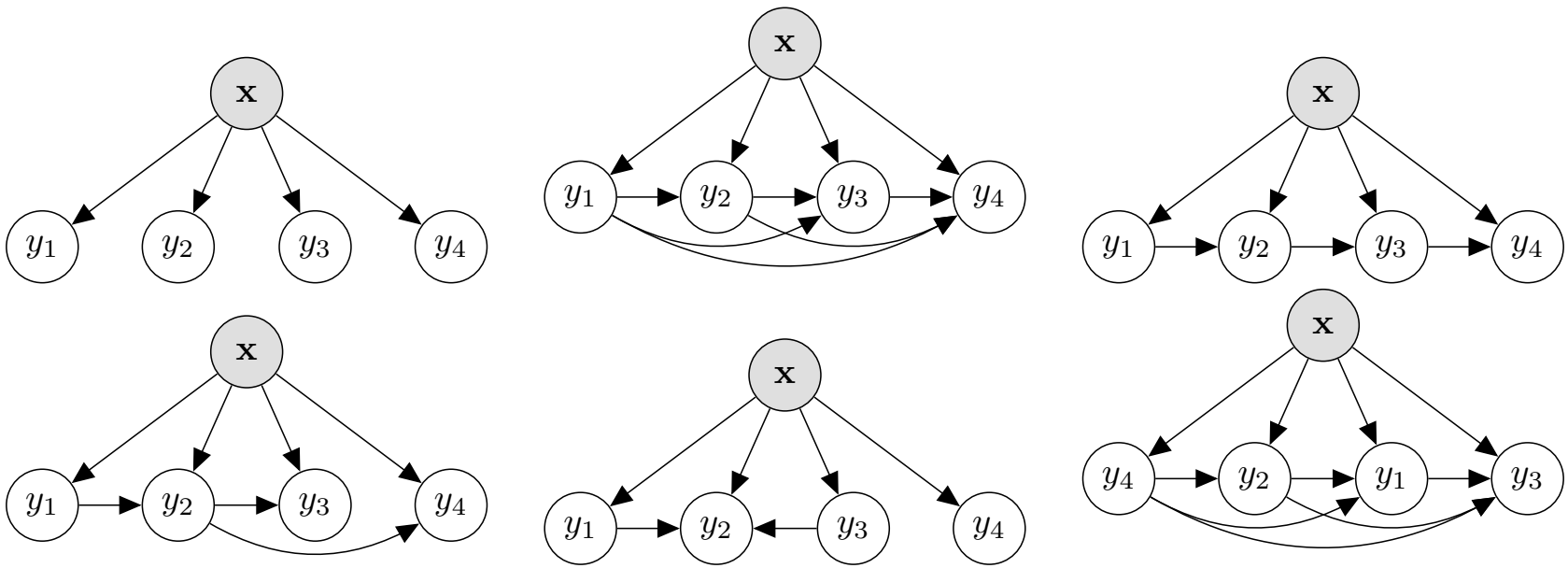
Learning as Search

Which Structure?

Learning can be seen as

$$\underset{\theta \in \Theta}{\text{argmax}}$$

i.e., a search. But θ can also define architecture/structure.



Also a consideration for, e.g., optimal Decision Trees.

Learning Structure from Data

Assume training data is generated from ground-truth graph (structure/Bayesian network) $G_* \in \mathcal{G}$:

$$\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n \sim P_{G_*;\theta_*}$$

How to find $G \approx G_*$?

Unfortunately:

- Intractable number of graphs $|\mathcal{G}|$ (super-exponential)
- And we cannot identify G_* anyway
  - Not enough data; instability (different G for same data)
  - Different G for same P (recall: $P(A|B)P(B) = P(B|A)P(A)$):

Fortunately:

- We can recover some equivalent G (equivalence class of G_*)
- We often don't need G_* (just want good expected accuracy)
- Many approaches are available to help us search for it

Search for Structure

- Node s is a graph, or tree, $G_n \in \mathcal{G}$
- Action a modify the graph (i.e., change the state)
- Reward r performance associated with the graph architecture

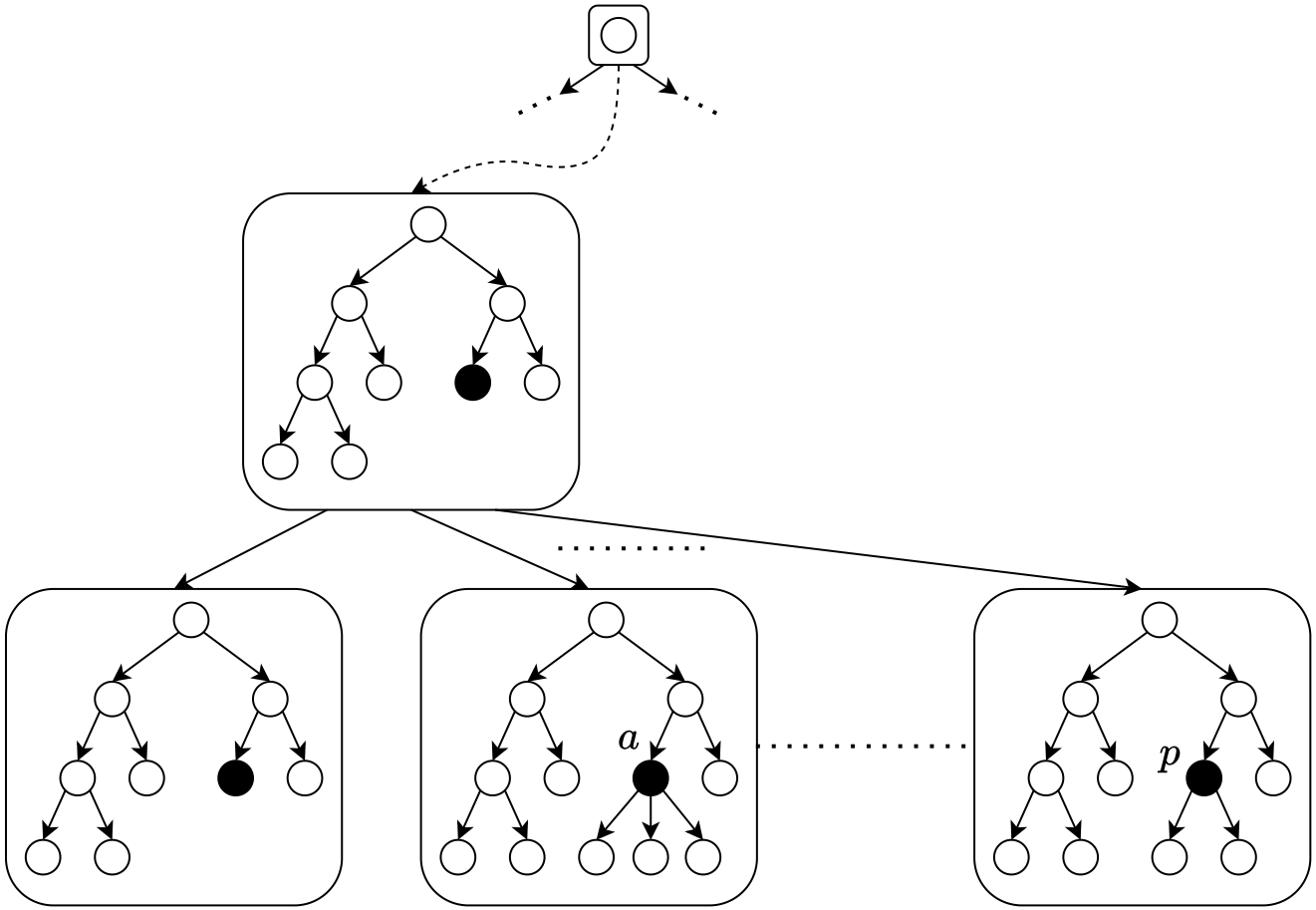


Image credits: *Online Learning of Decision Trees with Thompson Sampling*, Chaouki et al.

Dynamic Structure and Agents

So far:

1. Find $G \in \mathcal{G}$, train h_G
2. At inference time (test example $\tilde{\boldsymbol{x}}$),

$$\hat{\boldsymbol{y}} = h_G(\tilde{\boldsymbol{x}}) = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}} P_G(\boldsymbol{y} | \tilde{\boldsymbol{x}})$$

but is G best for this $\tilde{\boldsymbol{x}}$ in particular?

- Structure may be conditionally dependent on test instance!
- We can include structure in inference;

$$\hat{\boldsymbol{y}} = h(\tilde{\boldsymbol{x}}) = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}, G \in \mathcal{G}} P_G(\boldsymbol{y} | \boldsymbol{x})$$

approximations are necessary; many trade-offs involved!

Ideal setting: an agent looks at the data, provides the appropriate structure and parameters (architecture).

Neural Architecture Search

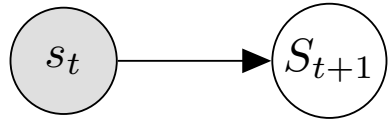
- Structure search is not specific to Bayesian networks
- Neural networks face the same problem: how to know what structure (architecture) and [hyper]-parameters to use, for a given problem/data set
- Bayesian networks can be neural networks (e.g., classifier chains); (Generally: Bayesian networks marginalize out $\boldsymbol{z}$, Neural networks propagate through $\boldsymbol{z}$)

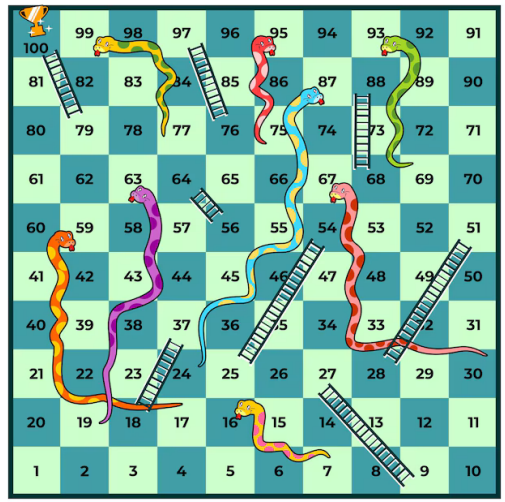
Some solutions/related concepts to be aware of:

- Use a recurrent structure (e.g., RNNs, LSTMs; output labels as a sequence)
- Transformers and attention mechanisms
- Hyperparameter optimization (structure as a hyperparameter)
- Meta learning: learning to learn (including structure)
- Automated ML (AutoML): structure choice is automated
- Neural Architecture Search: a policy to decide how build and traverse structure; reinforcement learning.

Learning and Inference Together

Markov Process (MP)



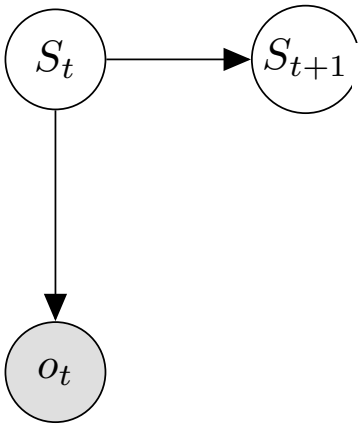


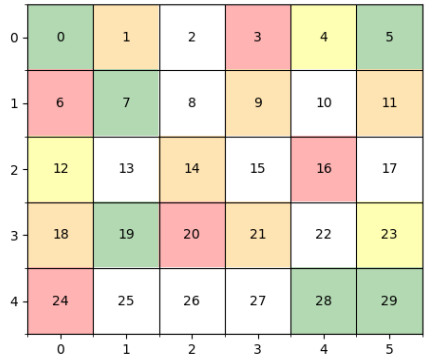
$$P(S_{t+1} \mid s_t)$$

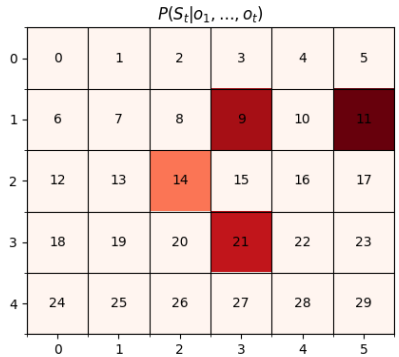
$$P(s_1, \ldots, s_t) = \prod_{i=1}^{t} P(s_i \mid s_{i-1})$$

If we don't have P, we can learn it (observation = state).

Partially Observed Markov Process (POMP)







$$o_1, \ldots, o_t = \begin{bmatrix} 01 \\ 00 \\ 11 \\ 00 \end{bmatrix}$$

But we never observe s!

Inference: maintain distribution over all possible states s_t:

$$P(S_t) = \sum_{s_1,\ldots,s_{t-1}} \prod_{i=1}^{t} p(s_i \mid s_{i-1}) p(o_i \mid s_i)$$

Can't just learn p from observation, because we don't observe s!
We need to make some assumptions about S_t and $P(S)$ is.

Learning the POMP

We can learn $P(Z \mid o)$, but this is an unsupervised task. Representation learning. For example, VAE,

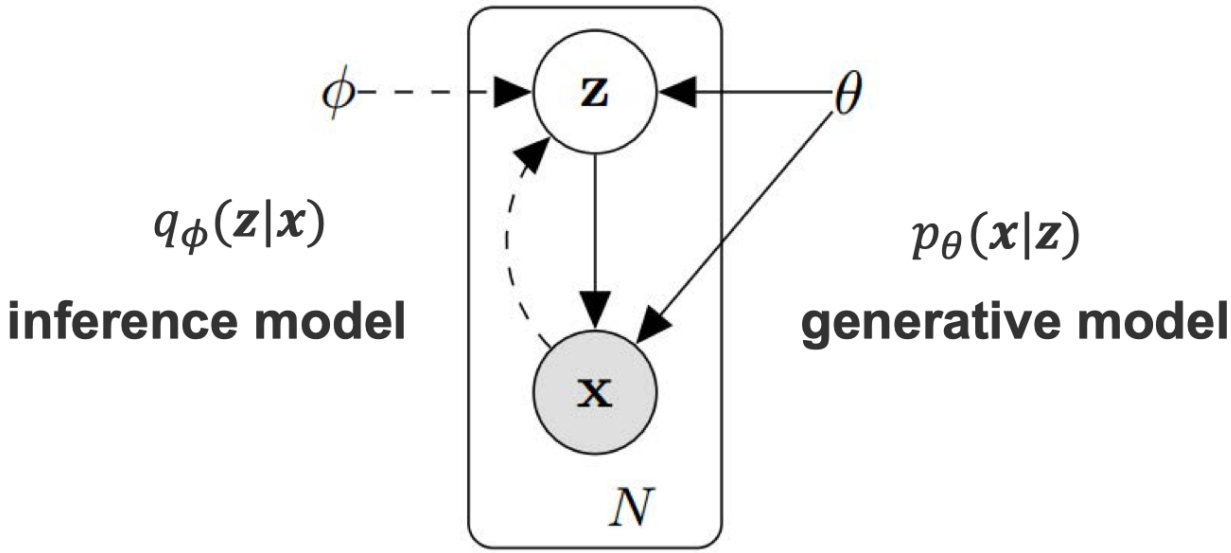
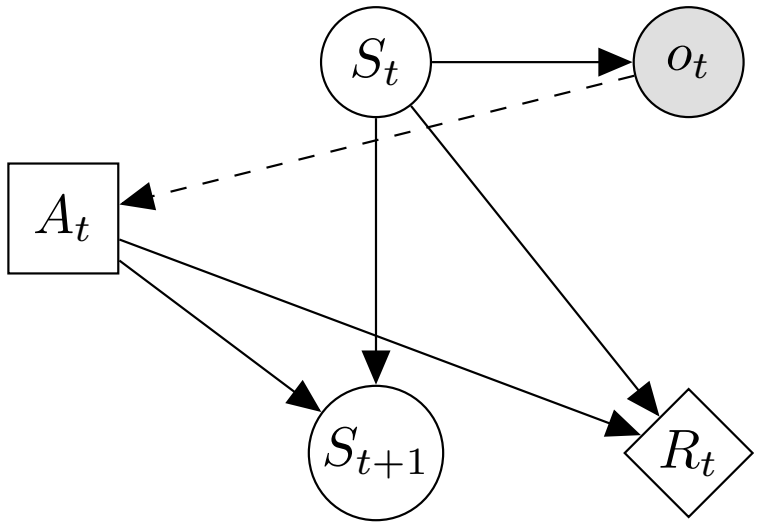


Figure courtesy: Kingma & Welling, 2014

This is difficult, we don't know about $\boldsymbol{s}$, develop $\boldsymbol{z}$ instead)

Learning is tractable in special cases, e.g., expectation maximization (EM), where $z_t \in \{1, \ldots, k\}$.

Partially Observed Markov Decision Process (POMDP)



We need to learn how to act!

Reinforcement learning

Major difference from POMPs: Causality! Our actions a_t change the future. Major implications.

But first, . . .

Imitation Learning

If we have data from an expert agent – we can just copy them.

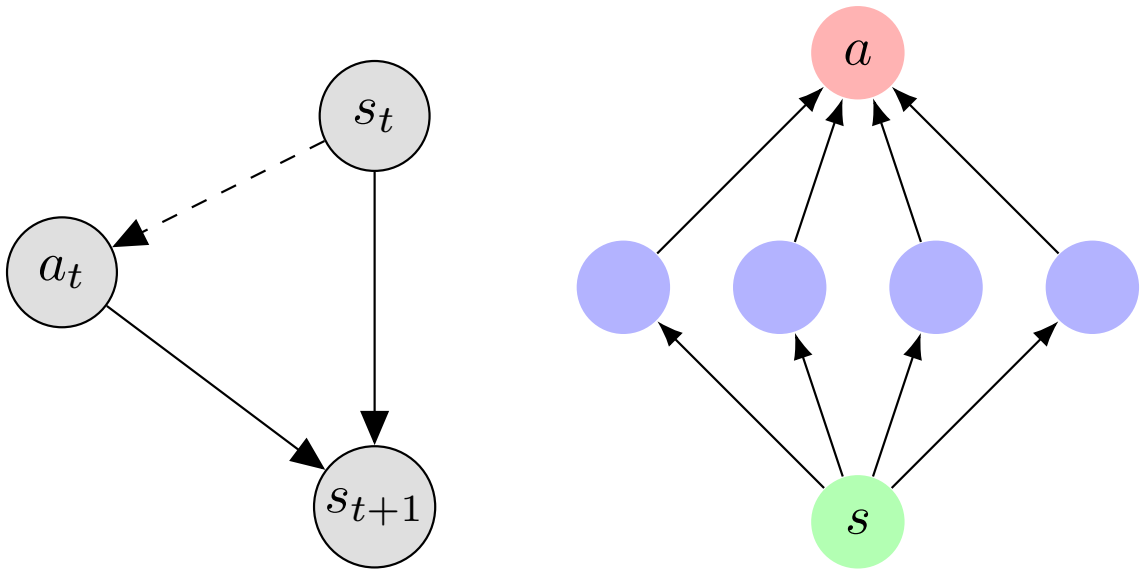


Image source (right): Sergey Levine

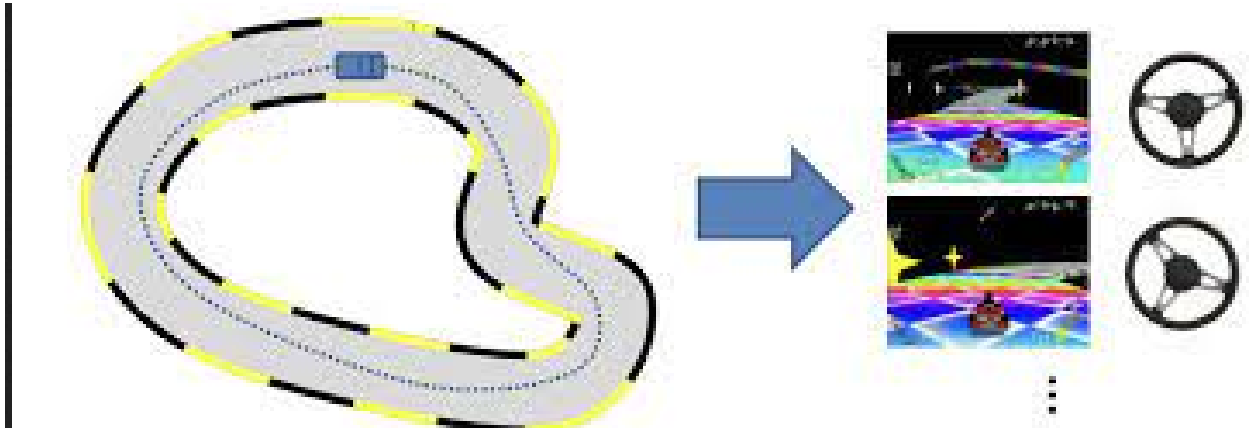
We target[6]

$$a \sim \pi(A \mid s)$$

or, directly $\pi_\theta : s \mapsto a$.

This is just supervised learning (complications may apply)!

[6] π for policy



1: **procedure** Behavioral Cloning (Imitation Learning)

2: Observe expert trajectories

$$(s_1, a_1), \ldots, (s_T, a_T) \sim \pi_*$$

3: Gather pairs from trajectories into dataset

$$\mathcal{D} = \{(s_t^{(i)}, a_t^{(i)})\}_{i=1}^{N}$$

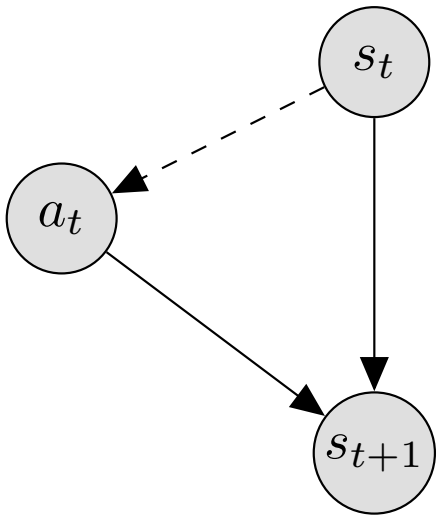
4: Train a supervised model (using dataset $\mathcal{D}$):

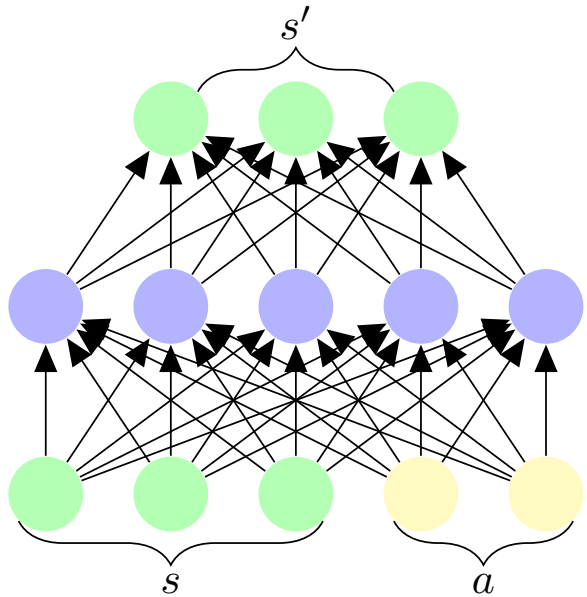
$$\pi_\theta : s \mapsto a$$

Leads to discussions of transfer learning and model learning.

Learning the Model

What if we shouldn't crash the car? Practice first in simulation. Build a simulator. Suppose data from an agent interacting in an environment.





We target

$$p(s' \mid s, a)$$

or, directly $s' = f(s, a)$.

This too is just supervised learning (complications may apply)!

1: **procedure** MODEL LEARNING

2: Observe some trajectories

$$(s_1, a_1), \ldots, (s_T, a_T) \sim \pi_0$$

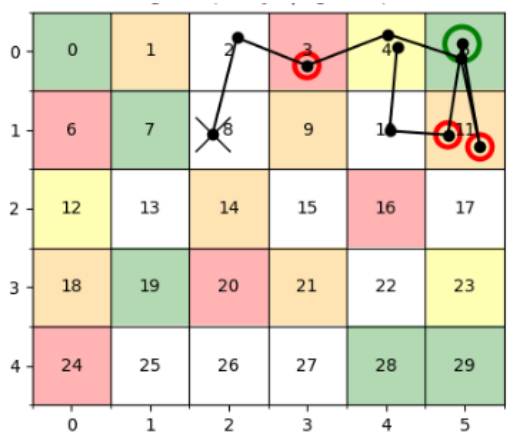
3: Gather pairs from trajectories into dataset

$$\mathcal{D} = \left\{ ((s_t^{(i)}, a_t^{(i)}), s_{t+1}^{(i)}) \right\}_{i=1}^{N}$$

4: Learn dynamics

$$f : (s, a) \mapsto s'$$

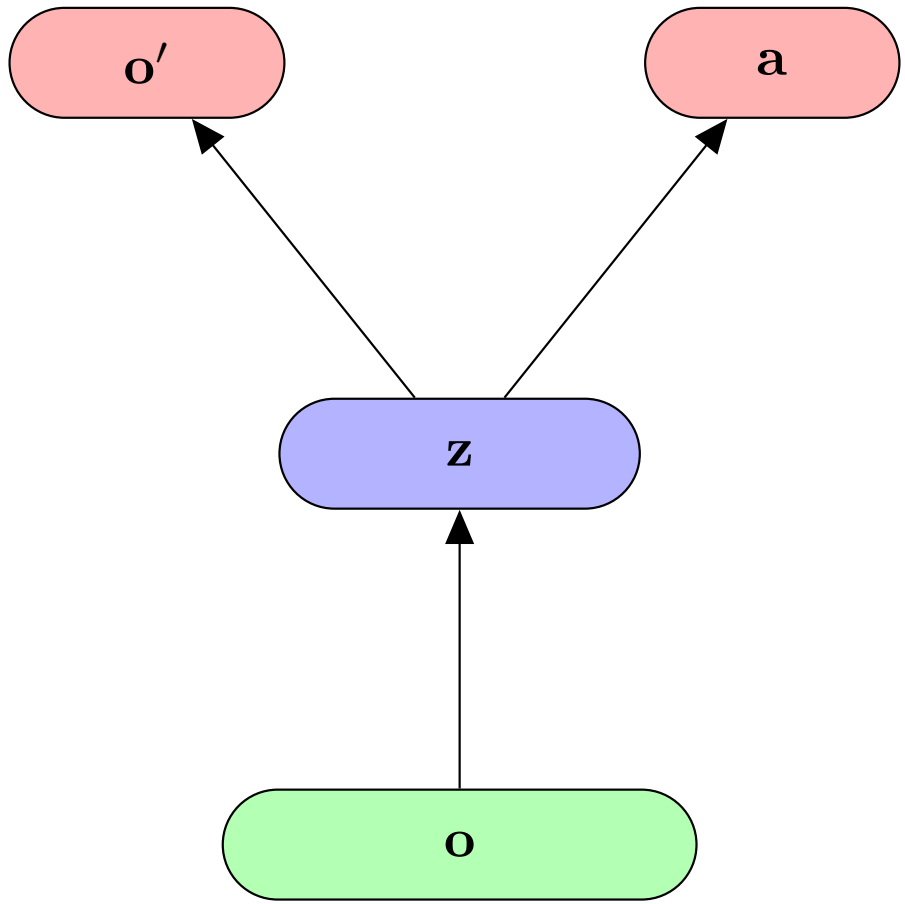
5: Do planning, e.g., build search tree from $f(s, a)$; solve



e.g., $3 = f(4, \leftarrow)$, so $\mathbf{a}_* = [\leftarrow, \leftarrow, \downarrow]$

Can replace s with observations (if we have no access to state).

Shared Representations; Multiple Heads



where $\boldsymbol{z} \approx \mathbf{s}$.

An Aside: Autonomous Agents and Data-Stream Learning

Typical instances of RL *are also* instances of learning from a data-stream:

$$o_1, a_1, o_2, a_2, o_3, a_3, \ldots$$

Learning (updating θ_t) must be carried out incrementally.

An area of literature 'data stream learning' already discovered (and proposed solutions for) many relevant questions, e.g.,

- starting from scratch ($t = 0$), what assumptions can we make?
- how to normalise across a stream?
- when/how to tune hyperparameters?
- online learning or batch-incremental updates?
- how to deal with delayed and sparse labels?
- how fast do we need to be?
- trade-offs in adapting (or not) to concept drift (dynamic environment)?

Lessons from the Data-Stream Learning Community

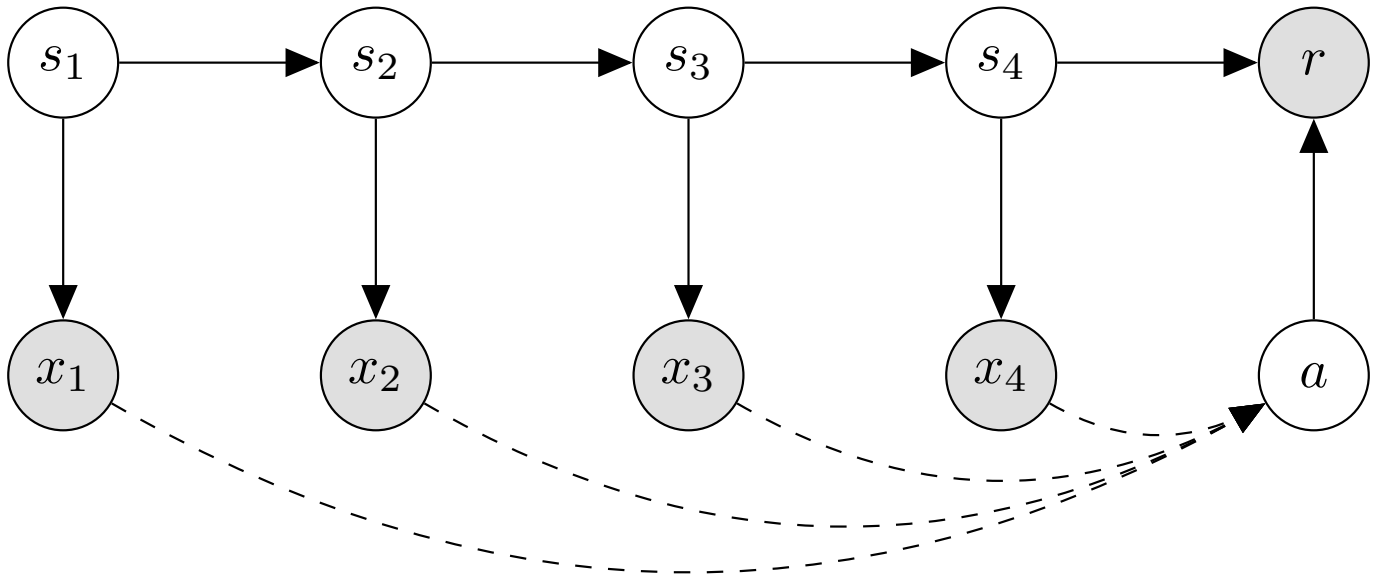
- start with synthetic data and toy problems
- validation on simple-but-standardised environments
- online learning is elegant, not necessarily most effective
- rigorous quantitative testing of hyper-parameters
- reproducible experiments, with statistical significance analysis
- develop and use extensively-tested software frameworks
- awareness of neighbouring areas (transfer learning, . . .)
- industry usually prefers simpler robust methods outperforming a baseline, rather than complex architectures with 'potential' for exceptional performance

Historical perspective on two communities: an early trend towards non-neural architectures (e.g., incremental decision trees) exhibiting high stability (no risk of forgetting) but low capacity, and thus a need to detect concept drift – vs – deep learning community: huge capacity, naturally incremental, but 'catastrophic forgetting'.

Autonomous Agent via Sequence Modelling

$$a \sim P(A_t \mid s_1, \ldots, s_t, a_1, \ldots, a_{t-1}, r)$$

where r the desired reward.



Decision Transformer: use a transformer model.

No model, no policy, just learn from dataset, e.g.,

$$\mathcal{D} = \{(x_1^{(i)}, x_2^{(i)}, \ldots, x_T^{(i)}, a_T^{(i)}, r_T^{(i)})\}_{i=1}^n$$

to produce policy $\pi_\theta(a \mid s)$ that obtains high return (value)

Summary

Summary

- Search and Learn
- Multi-label Classification (Bayesian Networks, with learning)
- Efficient Search on Probability Trees (Inference as Search)
- Planning as a Search Problem (more on this later)
- Architecture Search
- Imitation Learning (clone the policy)
- Model Learning (learn the environment)
- Practical challenges: efficiency, scaling up, uncertainty
- Practical challenges: learning from a data stream
- Probabilistic vs neural-network architectures:
  - Can be equivalent, can work together as part of the same model
  - Can constrain a Bayes net to act as a neural net; can 'force' probabilistic information from a neural net (dropout, etc.)
  - Different perceptions of "depth": propagation vs iteration

What's missing? (In the Context of Agents)

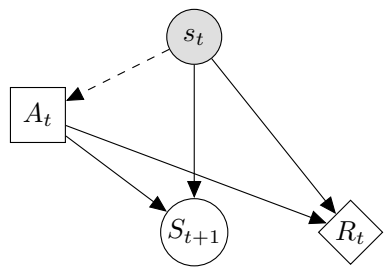
We have mechanisms to **learn** θ, structured around G; and **search** in complex $\mathcal{A}$:

$$\boldsymbol{a}_* = \pi(s \mid \boldsymbol{x}) = \underset{\boldsymbol{a},\theta,G}{\operatorname{argmax}} \underbrace{\mathbb{E}_{S \sim P_{G_\theta}(S|\boldsymbol{x})}[r(S, a)]}_{Q(\boldsymbol{a})}$$

Structure benefits agents and agents benefit structure [search]!

Almost a complete pipeline for an autonomous agent. What's missing (covered in upcoming lectures)?

Sequential decision making: when the action affects future states (and thus also future rewards).



Learning, Inference, and Planning

INF581 Advanced Machine Learning and Autonomous Agents

Jesse Read