

Неопределенные знания и рассуждения в условиях неопределенности

Тема 4

К.т.н., доцент

Беляев С.А.

Содержание

- Неопределенность, и проблема действий в условиях неопределённости
- Вероятностные рассуждения: байесовские сети, приближенные и точные алгоритмы поиска в байесовской сети
- Вероятностные рассуждения во времени: Марковский процесс, скрытые Марковские модели, фильтр Калмана, динамические байесовские сети
- Алгоритм фильтрации частиц
- Принятие простых решений: ожидаемая полезность, многоатрибутная полезность

Проблемы действий в условиях неопределенности

- Обработка всех возможных вариантов объяснений полученной информации
- Корректный условный план может стать бесконечно большим
- Иногда план не гарантирует достижение цели

Теория решений = теория вероятностей + теория полезности

- Maximum Expected Utility (MEU) – максимальная ожидаемая полезность

1

- Обновление доверительного состояния

2

- Вычисление результирующих вероятностей для каждого действия

3

- Выбор действия с наибольшей ожидаемой полезностью

Степень
уверенности

Теория
вероятностей

Нечеткая
логика

Теорема де Финетти

Если агент 1 руководствуется множеством степеней уверенности, которое нарушает аксиомы теории вероятностей, то существует такая комбинация ставок агента 2, которая гарантирует, что агент 1 будет терять деньги при каждой ставке

Утверждение	Убеждения №1	Ставки №2	Ставки №1	a, b	a, $\neg b$	$\neg a$, b	$\neg a$, $\neg b$
a	0.4	\$4 на a	\$6 на $\neg a$	-\$6	-\$6	\$4	\$4
b	0.3	\$3 на b	\$7 на $\neg b$	-\$7	\$3	-\$7	\$3
$a \vee b$	0.8	\$2 на $\neg(a \vee b)$	\$8 на $a \vee b$	\$2	\$2	\$2	-\$8
				-\$11	-\$1	-\$1	-\$1

Ставка не соответствует убеждениям – потери

Напоминаю аксиомы Колмогорова: $\sum P(\omega) = 1$

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

Полное совместное распределение вероятности

- **Сумма** всех вероятностей **равна 1**
 - обычно количество слагаемых слишком велико, чтобы использовать на практике
- **Независимые переменные** считаются отдельно
- **Правило Байеса** определяет изменение вероятности в зависимости от возникших событий

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

здесь
X – симптом (эффект, результат)
Y – причина

- Утверждения об **условной независимости** могут обеспечивать **масштабирование** вероятностных систем
 - Такие утверждения могут быть подкреплены данными намного проще по сравнению с утверждениями об абсолютной независимости
- **Наивная Байесовская модель** – модель применима при условии **независимости переменных**

Пример с зубной болью

	toothache		\neg toothache	
	catch	\neg catch	catch	\neg catch
cavity	0.108	0.012	0.072	0.008
\neg cavity	0.016	0.064	0.144	0.576

- $\Sigma P(x) = 1$
- $P(\text{cavity} \vee \text{toothache}) = 0.28$
 - $0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$
- $P(\text{cavity}) = 0.2$
 - $0.108 + 0.012 + 0.072 + 0.008 = 0.2$
- $P(\text{cavity} \mid \text{toothache}) = P(\text{cavity} \wedge \text{toothache}) / P(\text{toothache}) = 0.6$
 - $(0.108 + 0.012) / (0.108 + 0.012 + 0.016 + 0.064) = 0.6$
- $P(\neg \text{cavity} \mid \text{toothache}) = P(\neg \text{cavity} \wedge \text{toothache}) / P(\text{toothache}) = 0.4$
 - $(0.016 + 0.064) / (0.108 + 0.012 + 0.016 + 0.064) = 0.4$
- $P(\text{cavity} \mid \text{toothache}) = \alpha P(\text{cavity}, \text{toothache})$
 - $\alpha [<0.108, 0.016> + <0.012, 0.064>] = \alpha <0.12, 0.08> = <0.6, 0.4>$
- $P(\text{cause}, \text{effect}_1 \dots \text{effect}_n) = P(\text{cause}) \prod_i P(\text{effect}_i \mid \text{cause})$

catch – неприятные ощущения от захвата зуба клещами дантиста

Представление знаний в неопределенной проблемной области.⁷

Байесовская сеть

- Вершины – множество случайных переменных
- Вершины соединены ориентированными ребрами
- Каждая вершина характеризуется **распределением условных вероятностей** $P(X_i, \text{Parents}(X_i))$

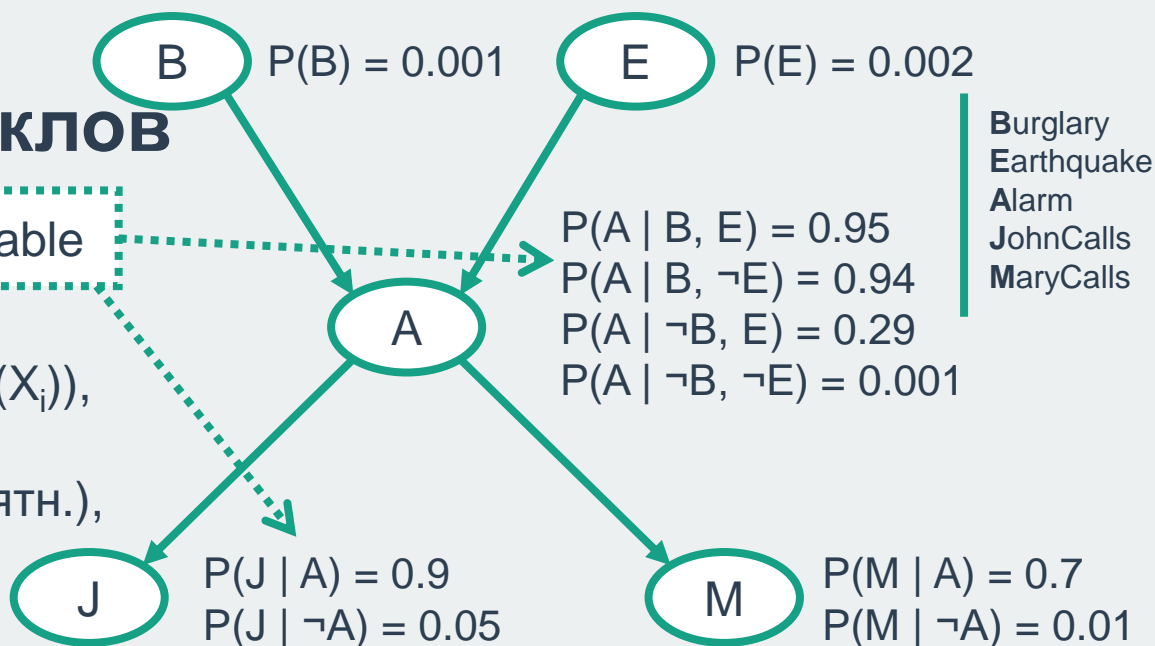
- **Граф не имеет циклов**

CPT – Conditional Probability Table

По цепному правилу:

$P(X_i | X_{i-1}, \dots, X_1) = P(X_i | \text{Parents}(X_i))$,
здесь $\text{Parents}(X_i) \subseteq \{X_{i-1}, \dots, X_1\}$.

$O(2^k)$ – количество чисел (вероятн.),
где k – количество родителей.



Алгоритм перебора для получения ответов на запросы в байесовских сетях (точное решение) (ENUMERATION-ASK)

функция Запрос_перебором(X, e, bn) : $P(X|e)$

// X – запрос, e – значения переменных свидетельств E ,
 bn – байесовская сеть с «переменные_ bn »

$Q(X)$ = пусто // распределение по X

для всех $x_i \in X$

$Q(x_i)$ = Перечислить_все(переменные_ bn , e_{x_i}), где $e_{x_i} = e \cup \{X = x_i\}$

вернуть Нормализовать($Q(X)$)

функция Перечислить_все(vars, e)

если vars == пусто, **то вернуть** 1.0

$Y \leftarrow$ Первая(vars)

если $Y == y$ в множестве e ,

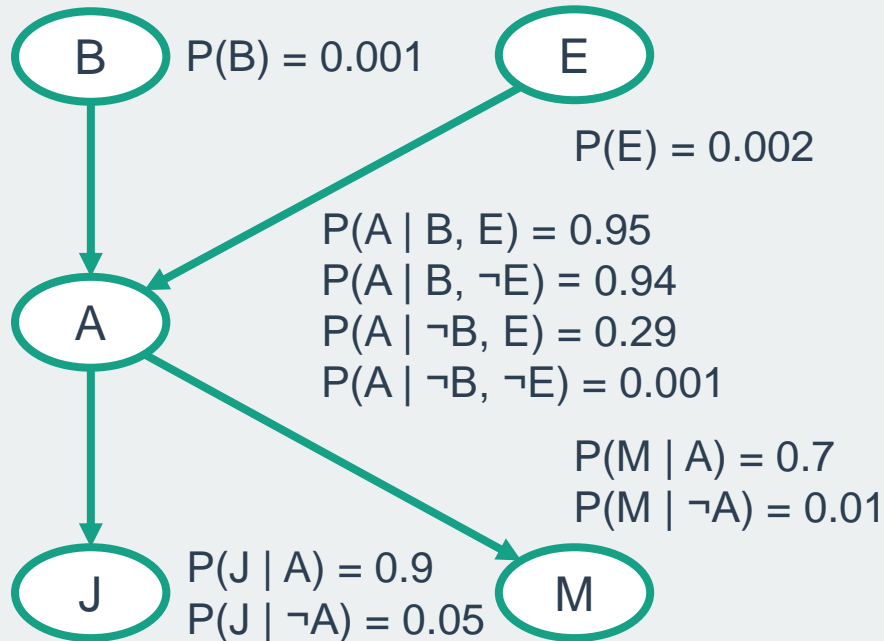
то вернуть $P(y|Parents(Y)) * \text{Перечислить_все}(\text{Остальные}(\text{vars}), e)$

иначе вернуть $\sum_y P(y|Parents(Y)) *$

$\text{Перечислить_все}(\text{Остальные}(\text{vars}), e_y)$, где $e_y = e \cup \{Y = y\}$

Сложность задачи: для n булевских переменных $O(2^n)$

Пример работы алгоритма



Классический поиск в глубину, обеспечивающий полный перебор всех вариантов, учитывающий специфику задачи.

В ситуации, когда указана Σ , то родитель может быть как истинным так и ложным (все возможные варианты).

- $bn = \{b, e, a, j, m\}$, $E = \{j\}$, $X = \{b, a\}$
- $Q(X) = \{0, 0\}$
- $x_i = b$
- $Q(b) = \text{Перечислить_все}(\{b, e, a, j, m\}, \{j, b\})?$
- $Y = \{b\} \in \{j, b\}?$
- вернуть $P(b|\text{родители})$ *
Перечислить_все ($\{e, a, j, m\}$, $\{j, b\}$)
- $Y = \{e\} \in \{j, b\}?$
- вернуть $\sum_e P(e|\text{родители})$ *
Перечислить_все ($\{a, j, m\}$, $\{j, b, e\}$)
- $Y = \{a\} \in \{j, b, e\}?$
- вернуть $\sum_a P(a|\text{родители})$ *
Перечислить_все ($\{j, m\}$, $\{j, b, e, a\}$)
- $Y = \{j\} \in \{j, b, e, a\}?$
- вернуть $P(j|\text{родители})$ *
Перечислить_все ($\{m\}$, $\{j, b, e, a\}$)
- $Y = \{m\} \in \{j, b, e, a\}?$
- вернуть $\sum_m P(m|\text{родители})$ *
Перечислить_все ($\{\}$, $\{j, b, e, a, m\}$)
- вернуть 1.0
- ...

Алгоритм оценки с учётом правдоподобия (LIKELIHOOD-WEIGHTING)

функция Оценка_правдоподобия(X , e , bn , N)

// N – количество выборок, которые будем формировать

// bn – сеть, задающая распределение $P(X_1, \dots, X_n)$

$W = \text{zeros}(n)$ // вектор взвешенных результатов подсчетов по X
для $j=1:N$

$\langle x, w \rangle \leftarrow \text{Взвешенный_экземпляр}(bn, e)$

$W[i] += w$ // x_i – значение переменной X в x

вернуть Нормализовать(W)

функция Взвешенный_экземпляр(bn , e)

$x \leftarrow$ событие с n элементами (с фиксацией значений из e)

$w=1$

для $i=1:n$

если X_i имеет значение x_{ij} в свидетельстве e

то $w \leftarrow w * P(X_i=x_{ij} \mid \text{Parents}(X_i))$

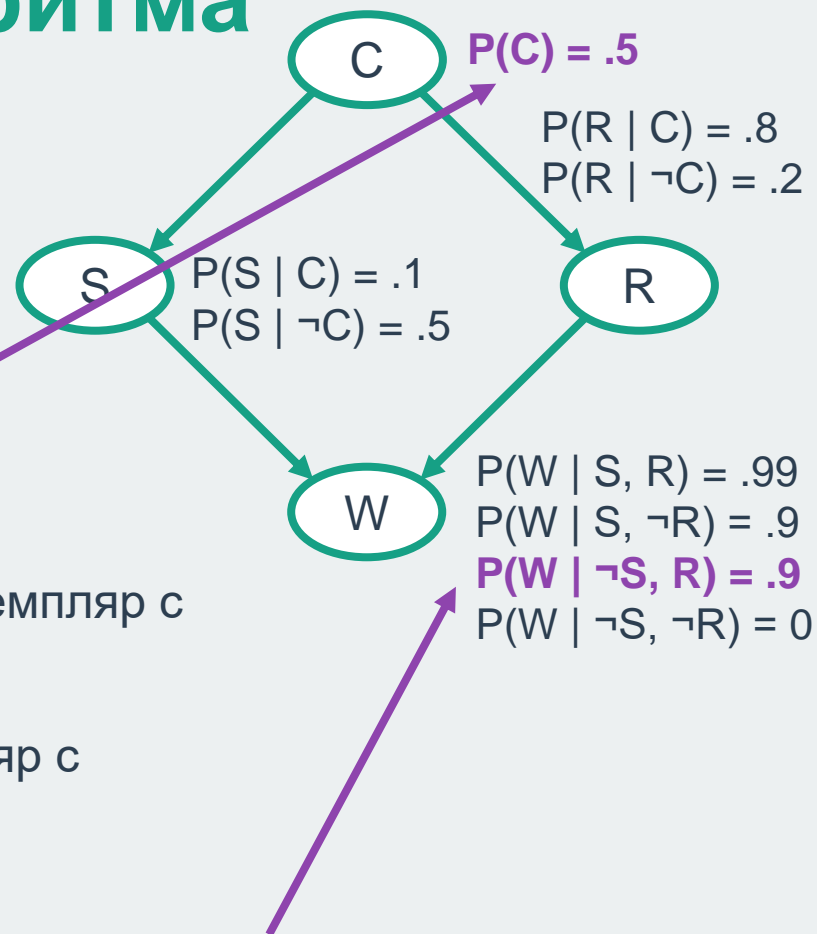
иначе $x_i \leftarrow$ случайная выборка из $P(X_i \mid \text{Parents}(X_i))$

вернуть $\langle x, w \rangle$

Идея: фиксировать значения из e и использовать случайную выборку для оценки w для переменных не из e

Пример работы алгоритма

- $P(\text{Rain} \mid \text{Cloudy}=\text{true}, \text{WetGrass}=\text{true})$
 - $e = \langle \text{Cloudy}=\text{true}, \text{WetGrass}=\text{true} \rangle$
- Порядок в X: Cloudy, Sprinkler, Rain, WetGrass
- Устанавливаем $w = 1$
- Cloudy есть в e, поэтому
 - $w \leftarrow w * P(\text{Cloudy}=\text{true}) = 0.5$
- **Sprinkler** нет в e, поэтому – случайный экземпляр с $P(\text{Sprinkler} \mid \text{Cloudy}=\text{true}) = \langle 0.1, 0.9 \rangle$
 - пусть получили Sprinkler=false
- **Rain** нет в e, поэтому – случайный экземпляр с $P(\text{Rain} \mid \text{Cloudy}=\text{true}) = \langle 0.8, 0.2 \rangle$
 - пусть получили Rain=true
- **WetGrass** есть в e, поэтому
 - $w \leftarrow w * P(\text{WetGrass}=\text{true} \mid \text{Sprinkler}=\text{false}, \text{Rain}=\text{true}) = 0.5 * 0.9 = 0.45$
- Возвращаемое значение $\langle \text{true}, \text{false}, \text{true}, \text{true} \rangle$ с весом 0.45



- Подход эффективнее перебора, но его эффективность резко снижается при большом количестве переменных свидетельства

Алгоритм формирования выборки на основе перебора (Монте-Карло) (PRIOR-SAMPLE)

функция Экземпляр(bn) : событие

// bn – сеть, задающая совместное распределение $P(X_1, \dots, X_n)$

$x \leftarrow$ событие с n элементами // массив из true/false

для $i=1:n$

$x_i \leftarrow$ случайная выборка из $P(x_i \mid \text{Parents}(X_i))$

вернуть x

- Формирует выборки на основе априорного совместного распределения, которое задано рассматриваемой сетью
- Создаётся большое количество экземпляров (много раз вызываем функцию), вычисляются значения, которые получили переменные (true/false)
 - оценка вероятности = отношение количества true к количеству экземпляров

Алгоритм формирования выборок с исключением (REJECTION-SAMPLING)

функция Экземпляр_с_исключением(X , e , bn , N) : $P(X|e)$

// X – запрос, e – событие, bn – сеть, N – количество выборок

C = пустой вектор // длины X

для $j=1:N$

$x \leftarrow$ Экземпляр(bn)

если x согласуется с e , **то**

$C_i ++$ // здесь i соответствует x_i – значению X в x

вернуть Нормализовать(C) // сумма значений равна 1

- Пусть $X = P(\text{Rain}|\text{Sprinkler}=\text{true})$, $e = \langle \text{Sprinkler}=\text{true} \rangle$, $N = 100$
- Пусть получилось $\langle \text{Sprinkler}=\text{false} \rangle$ 73 шт., а $\langle \text{Sprinkler}=\text{true} \rangle$ 27 шт.
- Пусть из 27 шт. $\langle \text{Rain}=\text{true} \rangle$ 8 шт., $\langle \text{Rain}=\text{false} \rangle$ 19 шт.
- Тогда $P(\text{Rain}|\text{Sprinkler}=\text{true}) \approx \text{Нормализовать}(\langle 8, 19 \rangle) = \langle 8/27, 19/27 \rangle = \langle 0.296, 0.704 \rangle$
- СКО ошибки $1/\sqrt{N}$

Вероятностный вывод по методу моделирования цепи Маркова (GIBBS-ASK) Markov chain Monte Carlo (MCMC)

функция MCMC_Запрос(X, e, bn, N) : $P(X|e)$

// X – запрос, e – событие, bn – сеть, N – количество выборок

C = пустой вектор // длины X

$Z \leftarrow$ переменные bn , отсутствующие в e

$x \leftarrow$ значения e , случайные значения из Z

для $k=1:N$

 случайный выбор z_i из Z

 случайный выбор значения z_i в x в соответствии с $P(z_i | mb(z_i))$

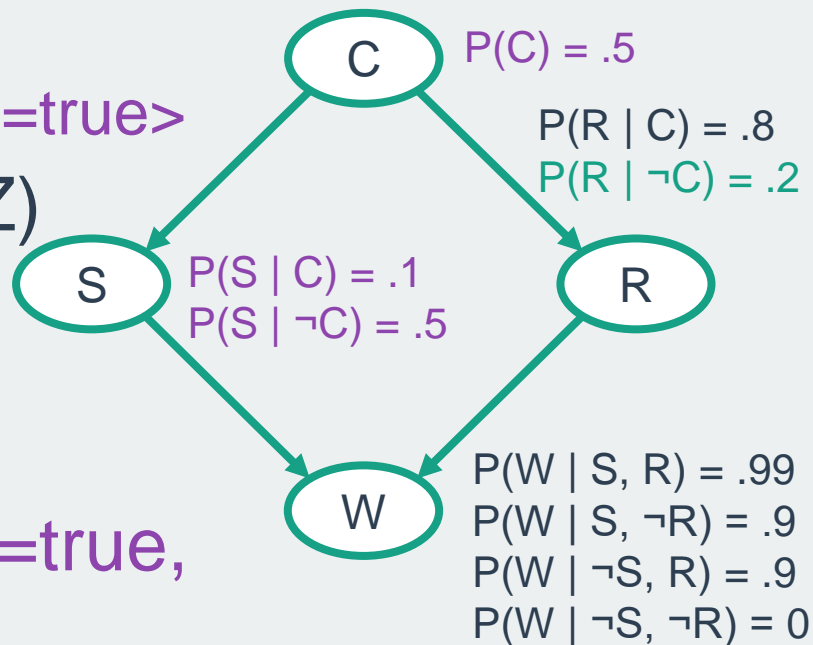
если x_i , **то** $C_i ++$ // здесь i соответствует x_i – значению X в x

вернуть Нормализовать(C) // сумма значений равна 1

- Здесь $P(x_i | mb(X_i)) = \alpha * P(x_i | Parents(X_i)) * \prod_{Y_j \in Children(X_i)} P(y_j | Parents(Y_j))$

Пример работы 1 шага алгоритма (выборка Гиббса)

- Запрос $P(\text{Rain} \mid \text{Sprinkler}=\text{true}, \text{WetGrass}=\text{true})$
 - $X = \text{Rain}$
 - $e = \langle \text{Sprinkler}=\text{true}, \text{WetGrass}=\text{true} \rangle$
- Выберем случайные (z_i из Z)
 $\text{Cloudy} = \text{true}, \text{Rain} = \text{false}$
- Вектор ($x = [c, s, r, w]$)
 $[\text{true}, \text{true}, \text{false}, \text{true}]$
- Расчет Cloudy для $\text{Sprinkler}=\text{true}, \text{Rain}=\text{false}$ ($P(z_i \mid \text{mb}(z_i))$)
 - $P(c \mid s, \neg r) = \alpha P(c) P(s|c) P(\neg r|c) = \alpha * 0.5 * 0.1 * 0.2$
 - $P(\neg c \mid s, \neg r) = \alpha P(\neg c) P(s|\neg c) P(\neg r|\neg c) = \alpha * 0.5 * 0.5 * 0.8$
 - $\alpha \langle 0.001, 0.020 \rangle \approx \langle 0.048, 0.952 \rangle$ - с этими вероятностями будет выбираться значение для Cloudy (x_i)



BUGS - байесовский вывод с использованием выборки по Гиббсу

The screenshot shows a web browser window with the URL www.mrc-bsu.cam.ac.uk. The page is the MRC Biostatistics Unit website. The header includes the University of Cambridge logo and navigation links: Study at Cambridge, About the University, Research at Cambridge, and a search bar. The main navigation bar includes links to Home, About Us, Research, Tackling COVID-19, People, Software (highlighted), and Training. Below this is a secondary navigation bar with Recruitment and News & Events. The main content area is titled 'The BUGS Project' and features a sidebar with a table of contents. The table of contents includes 'MRC Biostatistics Unit', 'Software', and 'The BUGS Project' with sub-links for WinBUGS, OpenBUGS, Support and contact, and New WinBUGS examples. The main text area is titled 'Background to BUGS' and describes the project's history and goals. It mentions that the project began in 1989 and led to the development of WinBUGS and OpenBUGS. The text also mentions that the site hosts the stand-alone WinBUGS 1.4.3 package.

MRC Biostatistics Unit

The BUGS Project

Background to BUGS

The BUGS (Bayesian inference Using Gibbs Sampling) project is concerned with flexible software for the Bayesian analysis of complex statistical models using Markov chain Monte Carlo (MCMC) methods. The project began in 1989 in the MRC Biostatistics Unit, Cambridge, and led initially to the 'Classic' BUGS program, and then onto the WinBUGS software developed jointly with the Imperial College School of Medicine at St Mary's, London. Developments were later focused on OpenBUGS, an open source equivalent of WinBUGS.

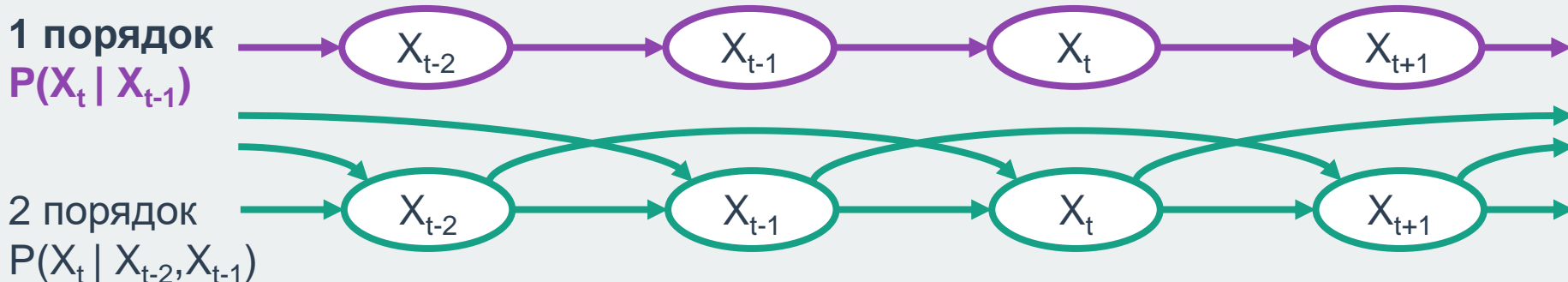
WinBUGS 1.4.3

This site at the MRC Biostatistics Unit hosts the stand-alone WinBUGS 1.4.3 package.

<https://www.mrc-bsu.cam.ac.uk/software/bugs/>

Марковский процесс

• Модель перехода



• Модель восприятия (переменные свидетельства)

- $P(E_t | X_{0:t}, E_{0:t-1}) = P(E_t | X_t)$
 - Как фактическое состояние мира влияет на результаты восприятия (свидетельства)
- ## • Распределение априорных вероятностей при $t=0$
- $P(X_0)$
- ## • Тогда с предположением независимости получаем
- $P(X_0, X_1, \dots, X_t, E_1, \dots, E_t) = P(X_0) \prod_{i=1..t} P(X_i | X_{i-1}) P(E_i | X_i)$
- ## • Если точность низкая, то рассмотреть возможность
1. повышения порядка модели
 2. расширения переменных состояния

Задачи вероятностного вывода во временных моделях

• Фильтрация

- Вычисление доверительного состояния распределения апостериорных вероятностей переменных текущего состояния

- вычисление $P(X_t | e_{1:t})$ // доверительное состояние

- $P(X_{t+1} | e_{1:t+1}) = \alpha P(e_{t+1} | X_{t+1}) \sum_{x_t} P(X_{t+1} | x_t) P(x_t | e_{1:t})$



пример на
следующем
слайде

• Предсказание

- Вычисление распределения апостериорных вероятностей значений переменных в будущем состоянии

- вычисление $P(X_{t+k} | e_{1:t})$ для заданного $k > 0$

- $P(X_{t+k+1} | e_{1:t}) = \sum P(X_{t+k+1} | x_{t+k}) P(x_{t+k} | e_{1:t})$

• Сглаживание

- Вычисление распределения апостериорных вероятностей значений переменных прошлого состояния

- вычисление $P(X_k | e_{1:t})$ для заданного $0 < k < t$

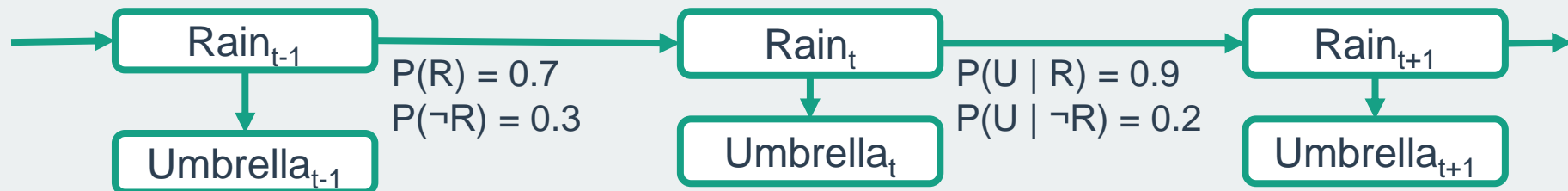
• Наиболее правдоподобное объяснение

- Определение последовательности состояний, которые с наибольшей вероятностью стали причиной получения результатов наблюдений

• Обучение

- Вычисление структуры моделей перехода и восприятия

Пример фильтрации



- Пусть $P(r_0) = \langle 0.5, 0.5 \rangle$
- День №1, наблюдаем $U_1 = \text{true}$
 - $P(R_1) = \sum P(R_1 | r_0) P(r_0) = \langle 0.7, 0.3 \rangle * 0.5 + \langle 0.3, 0.7 \rangle * 0.5 = \langle 0.5, 0.5 \rangle$
 - $P(R_1 | u_1) \propto P(u_1 | R_1) P(R_1) = \propto \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle = \propto \langle 0.45, 0.1 \rangle \approx \langle 0.818, 0.182 \rangle$
- День №2, наблюдаем $U_2 = \text{true}$
 - $P(R_2 | u_1) = \sum P(R_2 | r_1) P(r_1 | u_1) = \langle 0.7, 0.3 \rangle * 0.818 + \langle 0.3, 0.7 \rangle * 0.182 \approx \langle 0.627, 0.373 \rangle$
 - $P(R_2 | u_1, u_2) \propto P(u_2 | R_2) P(R_2 | u_1) = \propto \langle 0.9, 0.2 \rangle \langle 0.627, 0.373 \rangle = \propto \langle 0.565, 0.075 \rangle \approx \langle 0.883, 0.117 \rangle$
- ...

Скрытая марковская модель (Hidden Markov Model – HMM)

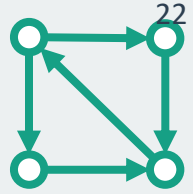
- **HMM** – это временная вероятностная модель, в которой состояние процесса описано с помощью **единственной дискретной случайной переменной**
 - значения этой переменной – **возможные состояния мира**
- Пусть X_t – переменная, имеющая S состояний
 - тогда **модель перехода** из состояния i в j описывается матрицей $S \times S$
 - $T_{ij} = P(X_t=j \mid X_{t-1}=i)$
- Для задачи с зонтиком $S=2$, $T = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$
- **Свидетельства** – диагональная матрица O , в которой на диагонали стоят $P(e_t \mid X_t=i)$
- Для задачи с зонтиком
 - $U_i=\text{true} \rightarrow O_i = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$, $U_i=\text{false} \rightarrow O_i = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix}$
- **Прямое преобразование:** $f_{1:t+1} = \alpha O_{t+1} T^T f_{1:t}$
- **Обратное преобразование:** $b_{k+1:t} = T O_{k+1} b_{k+2:t}$
- Пример: $f_{1:1} = \alpha \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} = \alpha \begin{pmatrix} 0.45 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.818 \\ 0.182 \end{pmatrix}$

См.
предыдущий
слайд

Пример снайпера с использованием Марковского процесса

- Предположим, что вектор состояний описывает безопасность 4 позиций снайпера
 - $V = \begin{pmatrix} 1.0 \\ 0.5 \\ 1.0 \\ 1.5 \end{pmatrix}$, sum = 4.0
- Выстрел из позиции 1 привлечёт внимание противника к этой позиции
 - $M = \begin{pmatrix} 0.1 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.8 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.8 \end{pmatrix}$
- Применяя её к вектору состояний, получим новый вектор
 - $V = \begin{pmatrix} 0.1 \\ 0.7 \\ 1.1 \\ 1.5 \end{pmatrix}$, sum = 3.4
 - Через некоторое время нигде не будет безопасно
- Не смотря на то, что мы работаем с распределением вероятностей, значения в матрице переходов задаются вручную
- Подстройка значений для получения нужного эффекта обычно затруднительна

Марковский конечный автомат (не распространённое название)



- Используя Марковские процессы, мы можем создавать системы принятия решений на основе численных значений для состояний
 - **Конечный автомат будет реагировать** на условия или события, **применяя** соответствующие **переходы** к вектору состояний
 - **Если** условий или **событий нет**, то выполняется **переход по умолчанию**
- **Переходы по умолчанию**
 - Выполняется на каждом «тике» времени
 - Конечный автомат **имеет внутренний таймер и матрицу перехода** по умолчанию
 - Если какой-то переход срабатывает, то таймер сбрасывается
 - Если переходов нет, то таймер считает время
- **Действия**
 - **Переходы возвращают действия, а состояния нет**

Фильтр Калмана (на простом примере)

Kalman filter network (KFN)

- **Предсказание** местоположения объекта **в следующий момент**, когда есть корреляция между скоростью и положением. Нормальный закон распределения ошибки.

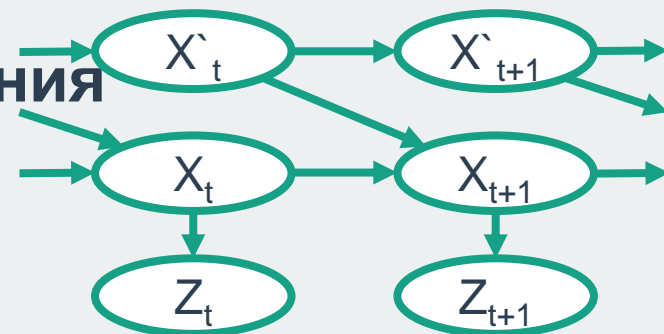
- $x_k = \begin{pmatrix} position \\ velocity \end{pmatrix}, P_k = \begin{pmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{vp} & \Sigma_{vv} \end{pmatrix}, \mu$ – среднее,
 x_0 – **начальное значение**

- $\begin{cases} p_k = p_{k-1} + \Delta t v_{k-1} \\ v_k = v_{k-1} \end{cases} \Rightarrow x_k = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} x_{k-1} = F_k x_{k-1}$

- $F_k = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$ – **матрица проецирования**

- $\begin{cases} x_k = F_k x_{k-1} \\ P_k = F_k P_{k-1} F_k^T \end{cases}$

- Будет работать **при отсутствии внешних воздействий**



<https://habr.com/ru/post/594249/>

Фильтр Калмана

Внешнее воздействие

- a – ожидаемое ускорение объекта

- $$\begin{cases} p_k = p_{k-1} + \Delta t v_{k-1} + \frac{1}{2} a \Delta t^2 \\ v_k = v_{k-1} + a \Delta t \end{cases}$$

- Назовём **внешние воздействия** u_k

$$x_k = F_k x_{k-1} + \begin{pmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{pmatrix} a = F_k x_{k-1} + B_k \overrightarrow{u_k}$$

- $$\begin{cases} x_k = F_k x_{k-1} + B_k \overrightarrow{u_k} \\ P_k = F_k P_{k-1} F_k^T \end{cases}$$

- B_k – матрица управления

- u_k – вектор управления

- **Новое** лучшее возможное **значение** является **прогнозом**, сделанным из предыдущего лучшего возможного значения плюс поправка на известные внешние воздействия

Фильтр Калмана

Внешняя неопределенность

- Мы рассматриваем **не отслеживаемые воздействия** как **шум** с ковариацией Q_k
- $$\begin{cases} x_k = F_k x_{k-1} + B_k \overrightarrow{u_k} \\ P_k = F_k P_{k-1} F_k^T + Q_k \end{cases}$$
- **Новое** лучшее возможное **значение** является **прогнозом**, сделанным **из предыдущего** лучшего возможного **значения** плюс поправка на известные внешние воздействия
- **Новая неопределённость** прогнозируется из **старой** неопределённости с дополнительной неопределённостью, вызванной окружением

Фильтр Калмана

Уточнение по результатам измерений

- Датчики измеряют состояние не точно.
Допустим, один считывает положение, другой – скорость
- Датчики моделируются матрицей H_k – прогноз измерения относительно реального положения
- Распределение показаний датчиков:
$$\begin{cases} \mu_{expected} = H_k x_k \\ \Sigma_{expected} = H_k P_k H_k^T \end{cases}$$
- Результаты наблюдения распределены по нормальному закону
 - z_k – среднее наблюдаемое значение показателей датчиков
 - R_k – ковариация неопределенности наблюдения

Перемножение нормальных распределений

- Нормальное распределение **предсказываемого местоположения** $N(x, \mu_0, \sigma_0)$
- Нормальное распределение **измеренного местоположения** $N(x, \mu_1, \sigma_1)$
- **Результат перемножения** нормальных распределений $N(x, \mu', \sigma') = N(x, \mu_0, \sigma_0) * N(x, \mu_1, \sigma_1)$, **одномерный случай**:
 - $$\begin{cases} k = \frac{\sigma_0^2}{\sigma_0^2 + \sigma_1^2} \\ \mu' = \mu_0 + k(\mu_1 - \mu_0) \\ \sigma'^2 = \sigma_0^2 - k\sigma_0^2 \end{cases}$$
- **В многомерном случае**:
 - $$\begin{cases} K = \Sigma_0(\Sigma_0 + \Sigma_1)^{-1} \\ \vec{\mu}' = \vec{\mu}_0 + K(\vec{\mu}_1 - \vec{\mu}_0) \\ \Sigma' = \Sigma_0 - K\Sigma_0 \end{cases}$$
- **K – коэффициент усиления Калмана**

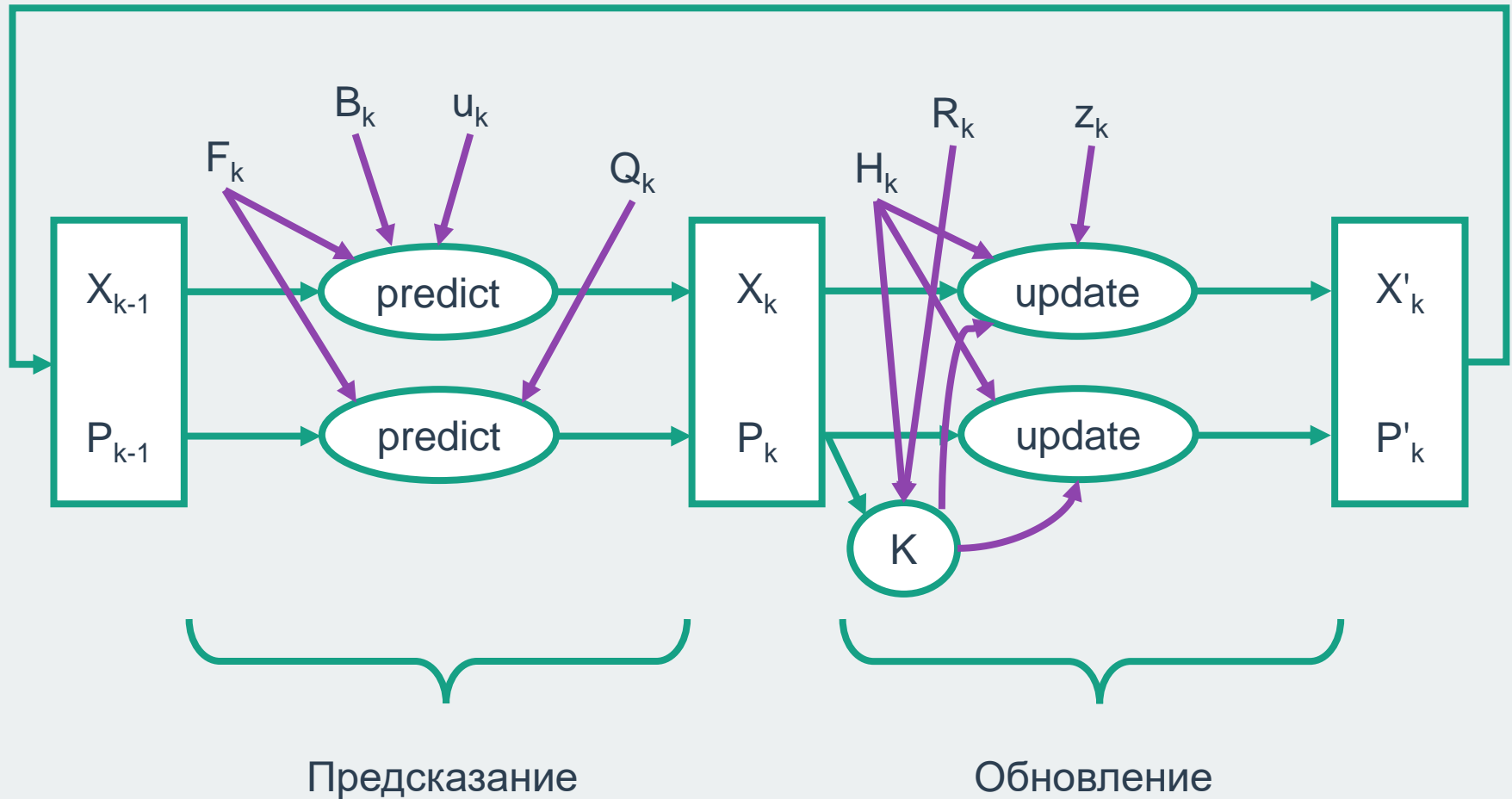
Фильтр Калмана

Обновление на основе наблюдения

- Распределение прогноза измерения
 - $(\mu_0, \Sigma_0) = (H_k x_k, H_k P_k H_k^T)$
- Распределение наблюдаемого измерения
 - $(\mu_1, \Sigma_1) = (\bar{z}_k, R_k)$
- Полные уравнения шага обновления
 - $$\begin{cases} K' = P_k H_k^T (H_k P_k H_k^T + R_k)^{-1} \\ x'_k = x_k + K' (\bar{z}_k - H_k x_k) \\ P'_k = P_k - K' H_k P_k \end{cases}$$
- Здесь x' и P' - наилучшие возможные значения с учетом результатов наблюдения
- Описанное решение работает для линейных систем, в случае нелинейных систем нужен расширенный фильтр Калмана

Kalman filter information flow

29



Динамическая байесовская сеть

Dynamic Bayesian networks (DBN)

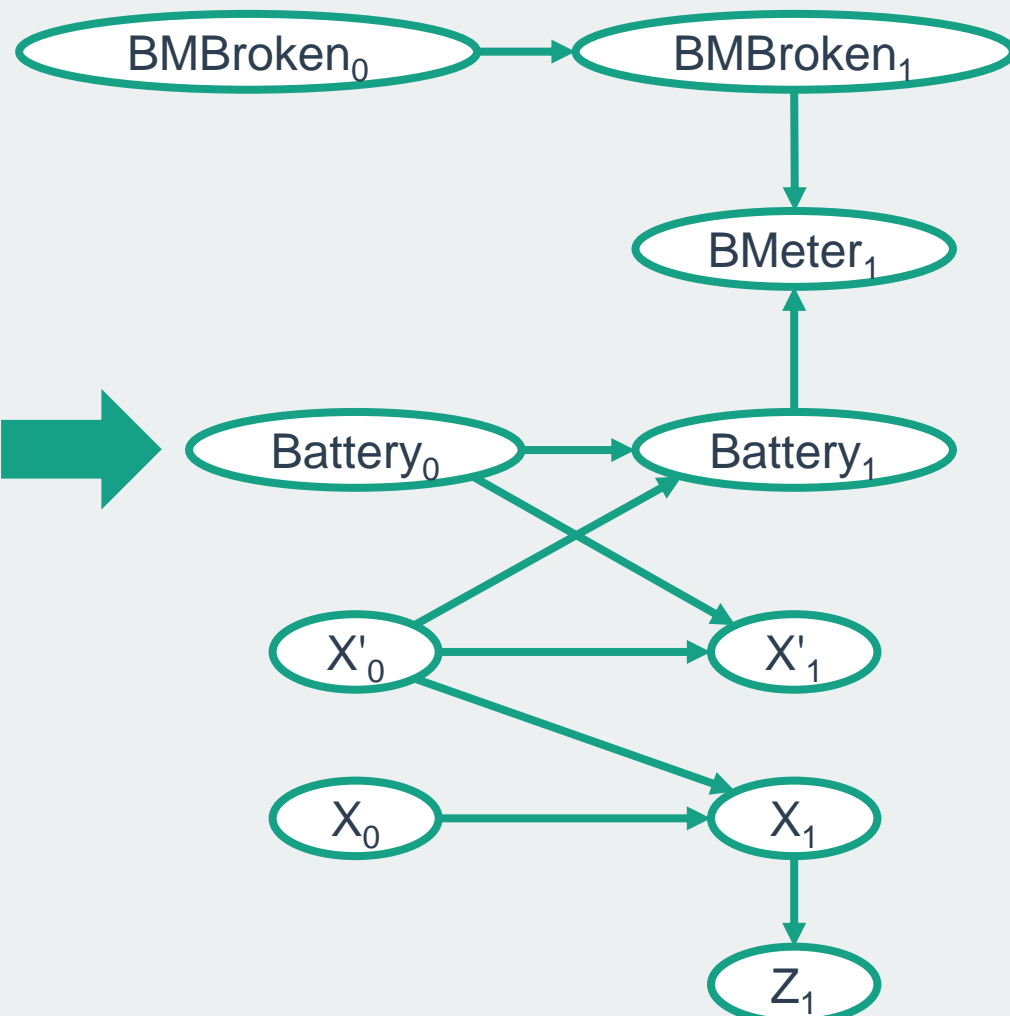
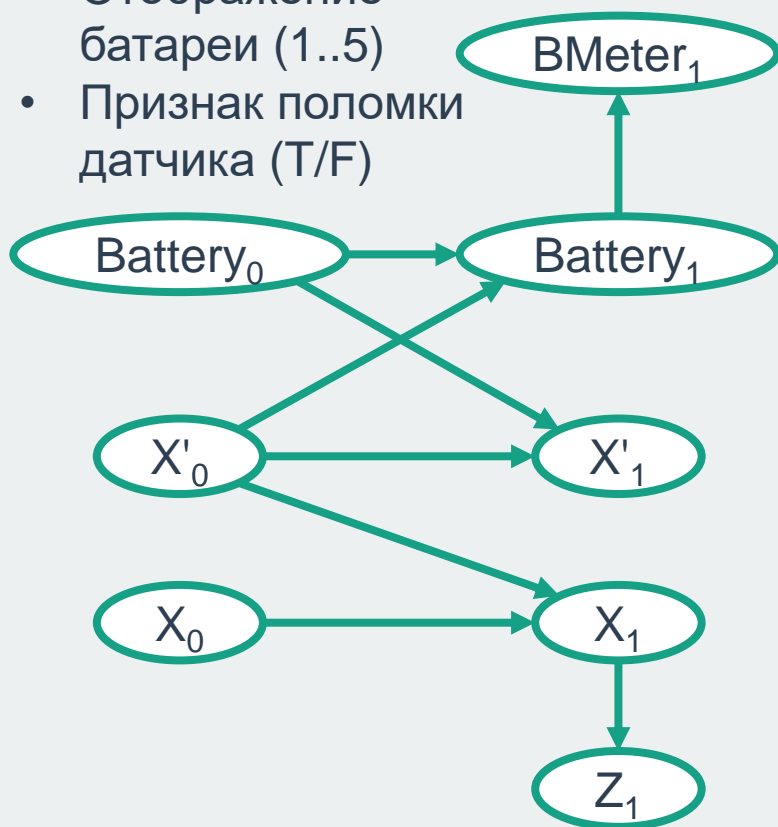
- DBN – временная версия байесовской сети
- Каждый временной срез DBN может иметь **любое количество переменных** состояния X_t и переменных свидетельства E_t
- DBN задается
 - $P(X_0)$ – начальное состояние
 - $P(X_{t+1} | X_t)$ – модель перехода
 - $P(E_t | X_t)$ – модель восприятия
 - Для **стационарной сети** модели идентичны для любого t , поэтому задаются для 1 шага (для первого среза)
- **Примеры DBN:**
 - HMM, KFN
- Отличия HMM и DBN
 - Пусть даны n дискретных переменных с d значениями
 - **Размер матрицы** переходов HMM = $O(d^{2n})$
 - Размер матрицы переходов DBN = $O(nd^k)$
 - k – верхняя граница количества «родителей» в сети
- Отличия KFN и DBN
 - KFN – единственное многомерное гауссово распределение
 - DBN – **произвольное распределение**

Пример DBN

Модель робота

31

- Координаты (X, Y)
- Скорости (X', Y')
- Батарея (1..5)
- Отображение батареи (1..5)
- Признак поломки датчика (T/F)



1. DBN могут использоваться для **представления** очень **сложных временных процессов** с многочисленными переменными, характеризующимися разрозненными связями между ними
2. **Не существует** возможности эффективно и **точно** формировать **вероятностные рассуждения** об этих процессах

Алгоритм фильтрации частиц (общий вид)

функция Фильтрация_частиц(e , N , dbn)

// dbn содержит $P(x_0)$, $P(X_1|X_0)$, $P(E_1|X_1)$

$S \leftarrow$ генерация_выборок(N , $dbn.P_{x0}$) // N случайных выборок

$W = \text{zeros}(N)$

для $i=1:N$

$S[i] \leftarrow$ выборка из $P(X_1 | X_0=S[i])$

$W[i] \leftarrow P(e | X_1=S[i])$

// С малым весом – вычеркиваем, с большим весом можем несколько раз взять

$S \leftarrow$ взвешенный_экземпляр_с_заменой(N , S , W)


вернуть S // множество выборок для следующего временного интервала

Далее – на примере
оценки местоположения
робота



Алгоритм фильтрации частиц

Инициализация

	?	?
?	?	?
?	?	?

- Задача: **определить местоположение робота**
- Дано:
 - развороты с точностью до ± 5 градусов (**погрешность разворота**)
 - перемещение с точностью до ± 20 метров (**погрешность движения**)
 - дальность до ориентиров с точностью до ± 15 метров (**погрешность оценки**)
- Алгоритм фильтра частиц делим на две части:
 1. инициализация
 2. основной цикл фильтрации
- Инициализация
 - Основной параметр фильтра частиц — **число** этих самых **частиц** — N
 - Если известно, что робот находится в клетке с координатами $(0, 0)$, то все частицы должны быть **случайным образом распределены** внутри этой клетки
 - Если априорной информации нет, тогда частицы распределяются по всей карте случайным образом
 - **Ориентация** робота внутри этой клетки нам неизвестна, поэтому — **случайна** (от -180 до 180 градусов)
 - Основной параметр каждой частицы — её **вес**, начальное значение — $w[i] = 1/N$
 - Ограничение: если частица **выходит за пределы карты**, её **вес становится равным нулю**
 - **Ориентиры**, если есть, **нужно добавить** в фильтр частиц

<https://habr.com/ru/post/276801/>

Алгоритм фильтрации частиц

1. Движение

- Робот **совершает движение** и из-за погрешностей **теряет информацию** о своем местоположении
- **Каждая частица** представляет собой **модель робота** — двигаться она должна как робот
 - Если дать роботу команду «проехать вперед 200 метров», то все частицы должны получить эту же команду, при этом «вперед» у них разное
- **Формулы**
 1. ориентация = ориентация + угол
 2. $\text{новый_X} = \text{старый_X} - \text{дистанция} + \sin(\text{ориентация})$
 3. $\text{новый_Y} = \text{старый_Y} + \text{дистанция} + \cos(\text{ориентация})$
- Здесь
 - нулевой угол вниз, угол 90° - направление влево
- **Результат**
 - робот мог проехать **с погрешностью** как **по углу**, так и **по расстоянию**
 - **частицы повторяют в т.ч. погрешность** робота

Алгоритм фильтрации частиц

2. Измерение

- Робот и частицы **вычисляют расстояние до ориентиров**
 - $\text{расстояние}_i = \sqrt{(\text{ориентир}_X - \text{робот}_X)^2 + (\text{ориентир}_Y - \text{робот}_Y)^2}$
 - здесь i – номер частицы
- **Сравнение с использованием f , идентичному нормальному закону распределения**
 - $f(\mu) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{\mu-\mu_0}{\sigma}\right)^2}$
 - σ – СКО измерений
 - μ_0 – МО (измерение робота)
 - μ – измерение частицы
 - $f(\mu)$ – вероятность получения μ при заданных μ_0 и σ
- Пусть дано 4 ориентира, тогда **вес каждой частицы**
 - $w[i] = f(\text{расстояние}_i(1) * \text{расстояние}_i(2) * \text{расстояние}_i(3) * \text{расстояние}_i(4))$
- **Нормализация веса частиц (сумма должна равняться 1)**
 - $S = \sum w[i]$
 - $w[i] = w[i] / S$

Алгоритм фильтрации частиц

3. Отсев

- Необходимо **отсеять частицы с низким весом**
- Каждая частица из исходного массива переходит в новый (**переживает отсев**) с **вероятностью равной её весу**
- Есть много алгоритмов, например, **колесо отсева** (resampling wheel)

функция колесо_отсева(список_частиц, w, N)

index = случайное_целое(0,N-1) // Начальное положение в списке

beta = 0, новый_список = [], новый_w = []

для i=1:N

beta += случайное_uniform(0, 2 * максимальный_вес)

пока beta > w[index]

beta -= w[index]

index = (index + 1) % N

новый_список.добавить(список_частиц[index])

новый_w.добавить(w[index])

вернуть <новый_список, новый_w>

- **Нормализация веса** частиц (сумма должна равняться 1) – выполняется **после отсева**

$$S = \sum w[i]$$

$$w[i] = w[i] / S$$

Алгоритм фильтрации частиц

Оценка состояния

- Получить **оценку состояния** можно **на любом этапе** работы алгоритма

функция оценка(список_частиц, w, N)

$X = 0$

$Y = 0$

для $i=1:N$

$X += \text{список_частиц}[i].X * w[i]$

$Y += \text{список_частиц}[i].Y * w[i]$

вернуть $\langle X, Y \rangle$

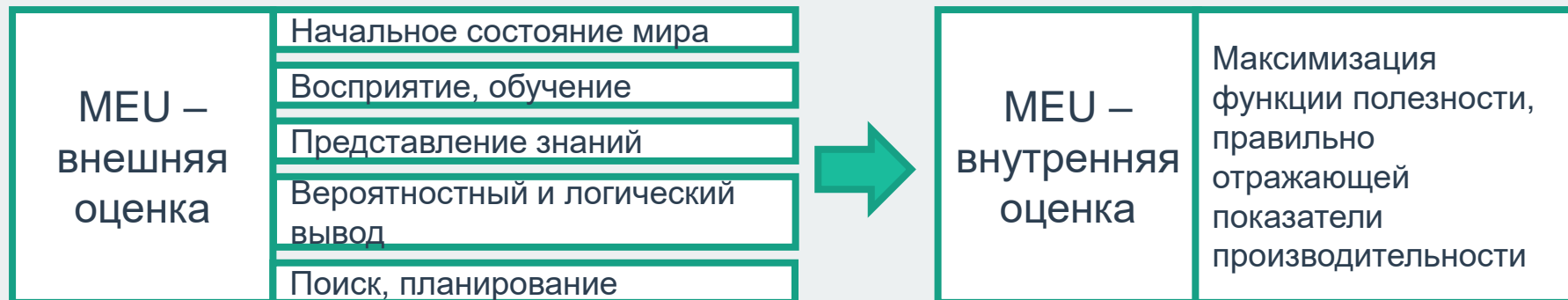
- Путем применения фильтра частиц в приведенном примере, удалось всего за 8 итераций основного цикла получить координаты робота с точностью 12.9 метров, при погрешности движения 20 метров и погрешности измерения 15 метров

<https://habr.com/ru/post/276801/>

Выбор действий в условиях неопределенности

- $P(\text{RESULT}(a)=s') = \sum P(s) * P(s' | s, a)$ – вероятность перейти в состояние s' , выполняя действие a
 - a – доступное действие
 - $P(s)$ – вероятность того, что находимся в состоянии s
 - $P(s' | s, a)$ – вероятность того, что перейдём в s' при выполнении действия a в s
- $EU(a) = \sum P(\text{RESULT}(a)=s') * U(s')$ – **ожидаемая полезность действия a**
 - U – функция полезности (utility function)
- Принцип **максимальной ожидаемой полезности (MEU)**, которым пользуется рациональный агент:
 - $\text{action} = \text{argmax } EU(a)$
- В случае последовательности действий данный подход **потребуется перебрать все возможные последовательности**, чтобы максимизировать ожидаемую полезность

Принцип MEU – лучший рецепт интеллектуального поведения



Рациональные предпочтения

39

- Варианты: A, B, C
- $A > B$
 - A предпочтительней
- $A \sim B$
 - Выбор безразличен
- $A \geq B$
 - A предпочтительней или безразлично
- Если действия **детерминированные**, то A, B, C – **конкретные состояния**
- иначе – **недетерминированный случай**, тогда это **лотерея**
 - у каждого состояния появляется **вероятность p**
 - $L = [p_1, C_1; \dots; p_n, C_n]$

- **Аксиомы теории полезности**
 - Упорядочиваемость
 $(A > B) \vee (B > A) \vee (A \sim B)$
 - Транзитивность
 $(A > B) \wedge (B > C) \Rightarrow (A > C)$
 - Непрерывность
 $A > B > C \Rightarrow \exists p [p, A; 1-p, C] \sim B$
 - Заменяемость
 $A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$
 - Монотонность
 $A > B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] > [q, A; 1-q, B])$
 - Декомпозируемость
 $[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$
- **Принцип полезности**
 - $U(A) > U(B) \Leftrightarrow A > B$
 - $U(A) = U(B) \Leftrightarrow A \sim B$
- **Принцип MEU**
 - $U([p_1, S_1; \dots; p_n, S_n]) = \sum p_i * U(S_i)$
- **При линейном изменении U ничего не изменится:**
 - $U'(S) = a * U(S) + b$
 - $a > 0$

Принятие решений в группе

- **Парадокс Кондорсе** (Condorcet paradox)
 - Alice: $X > Y > Z$
 - Bob: $Y > Z > X$
 - Cory: $Z > X > Y$
 - Вы не можете выбрать один вариант
 - Неясно, каким должен быть групповой результат в этом случае, потому что он симметричен по исходам
- **Функция социального предпочтения** дает отношение предпочтения для группы
 - Мы хотели бы, чтобы функция социальных предпочтений зависела от предпочтений индивидов в группе
- При наличии трех или более исходов **следующие свойства не могут одновременно выполняться** для любой функции социальных предпочтений
 - **Функция социальных предпочтений** является полной и транзитивной
 - Допускается любое индивидуальное предпочтение, которое является полным и переходным
 - Если каждый индивидум предпочитает результат o_1 результату o_2 , то группа предпочитает o_1 результату o_2
 - **Групповое предпочтение** между исходами o_1 и o_2 **зависит только от индивидуальных предпочтений** по o_1 и o_2 , а не от индивидуальных предпочтений по другим исходам
 - Ни один человек не может единолично решить исход (никакой диктатуры)

Это же распространяется и на многоатрибутную полезность

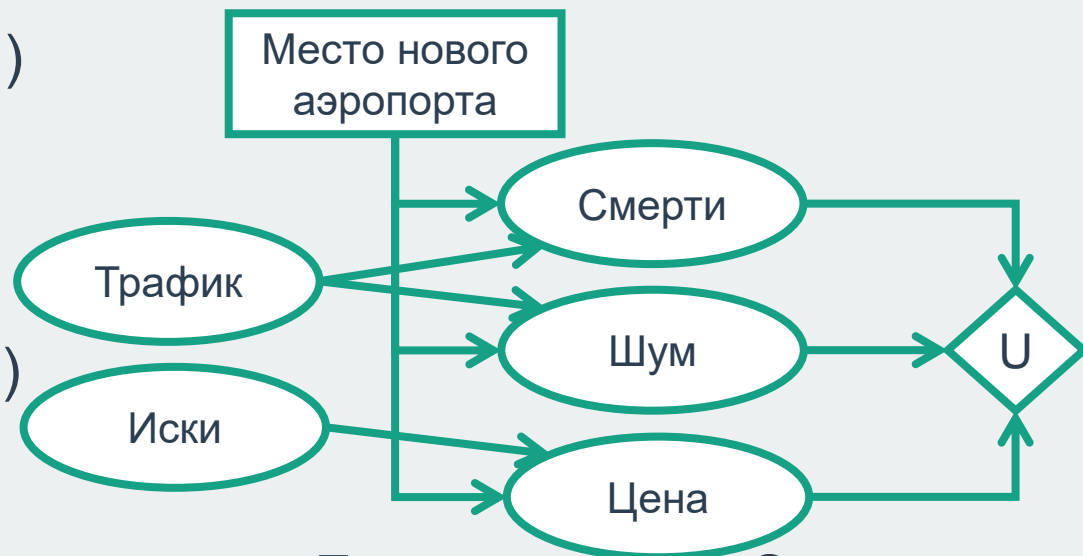
Многоатрибутная полезность

41

- Пусть (x_1, \dots, x_n) – предпочтения агента, каждый имеет d значений,
- тогда $U(x_1, \dots, x_n)$ имеет d^n значений – наихудший случай
- Необходимо выявить **регулярную структуру**, такую что $U(x_1, \dots, x_n) = f(f_1(x_1), \dots, f_n(x_n))$
- Предпочтения **без неопределенности**
 - **Детерминированный** случай, предполагаем независимость предпочтений
 - тогда **функция предпочтений**:
 $V(x_1, \dots, x_n) = \sum V_i(x_i)$
- Предпочтения **с неопределенностью** (предпочтения между лотереями)
 - Предполагаем **независимость функций полезностей**
 - иначе структура может быть очень сложной
 - Обозначим $U_i = k_i * U_i(x_i)$, где k_i – константа
 - Тогда **для трех переменных**
 - $U = U_1 + U_2 + U_3 + U_1 * U_2 + U_1 * U_3 + U_2 * U_3 + U_1 * U_2 * U_3$

Сети принятия решений

- **Узлы жеребьёвки** (овалы)
 - случайные переменные
- **Узлы принятия решений** (прямоугольники)
 - выбор ЛПР
- **Узлы полезности** (ромбы)
 - функция полезности



Алгоритм

1. **Определить значения** переменных E для текущего S
2. Для каждого возможного значения **узла принятия решений**:
 - а) **ввести это значение** в узел принятия решений
 - б) **вычислить апостериорные вероятности** для родительских узлов узла полезности, используя стандартный алгоритм вероятностного вывода
 - в) **вычислить результирующее значение полезности** для данного действия
3. **Возвратить действие** с самым **высоким** значением полезности

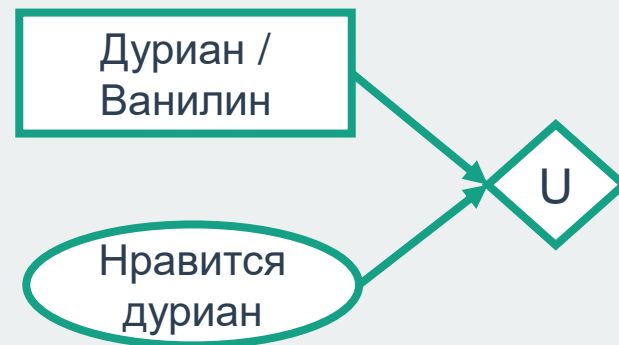
Пример сети принятия решений

- Выбираем наполнитель для мороженого
- Стоимость мороженого \$2
- Готов заплатить \$3
- Нравится ли дуриан? 50%/50%
- Если дуриан
 - нравится, то удовольствие оценим в \$100
 - не нравится – -\$80
- Что выбрать?
- $0.5 * \$100 - 0.5 * \$80 - \$2 = \8



Д/В	U
Дуриан	+\$8
Ванилин	+\$1

- Варианты без переменной «Нравится дуриан» – **не детерминистические**



Д/В	Нравится дуриан?	U
Дуриан	True	+\$98
Дуриан	False	-\$82
Ванилин	True	+\$1
Ванилин	False	+\$1

Возможные варианты неопределённости:

- относительно своей U
- относительно U человека, ради которого выполняется действие

Goal-oriented behavior (GOB). Цели

- Нет единой техники для **GOB**
- **Цели**

Упрощённая реализация сети
принятия решений

- Существуют **тысячи возможных целей** (мотивов), и у персонажа активных мотивов может быть произвольное количество
 - Уровень важности представляет собой число
 - Уровень 0 говорит о полном удовлетворении цели
- Возможные цели: **есть, восстановить здоровье, убить противника**
- Можно управлять целями без использования уровня важности, но при этом **сложно определить на какой цели сосредоточиться**
- В игре The Sims физические и эмоциональные параметры персонажа могут быть интерпретированы как значения целей
 - Персонаж должен иметь мотив «голод»
 - Чем выше мотив голода, тем более довлеющей становится цель «поесть»

- В дополнение к целям (мотивам) необходим **допустимый набор действий**, из которых можно выбрать
 - Могут создаваться централизованно
 - Обычно **создаются объектами в окружающем мире**
 - В The Sims
 - Чайник добавляет действие «Вскипятить чайник»
 - Пустая печь добавляет действие «Поместить сырую еду» в список доступных действий
- Доступное действие зависит от текущего состояния игры
 - После добавления действия в список возможностей все они упорядочиваются в соответствии с целями персонажа
 - Упорядочивание показывает эффект действия на каждую из целей
- **Реализация цели может быть в нескольких шагах**
 - Сырая еда не удовлетворит голод
 - Если персонаж её поднимет
 - он не станет менее голодным
 - но пустая микроволновка предложит действие «Поместить сырую еду», что обещает в последствии удовлетворить голод персонажа

ГОВ. Простой выбор

- Есть множество возможных действий и множество целей
- Действия обещают удовлетворить различные цели
- Предположим пример
 - Цель: Eat=4
 - Цель : Sleep=3
 - Действие: Get-Raw-Food (Eat-3)
 - Действие : Get-Snack (Eat-2)
 - Действие : Sleep-In-Bed (Sleep-4)
 - Действие : Sleep-On-Sofa (Sleep-2)
- Простой выбор
 - выбрать наиболее актуальную цель (с наибольшим значением)
 - найти действие, которое или полностью удовлетворяет цель, или предоставляет максимальное доступное удовлетворение
 - Лучшее: **get-raw-food action** (приведёт к **cooking** и **eating** еды)
- Персонаж естественно **выберет действие с наибольшим значением функции полезности**

Общая функция полезности против простого выбора

- Иногда полезнее измерить важность достижения цели (для **удовлетворения дискомфорта**)
- Пример
 - Цель: $Eat=4$
 - Цель : $Bathroom=3$
 - Действие: Drink-Soda ($Eat-2$; $Bathroom+2$)
 - afterwards: $Eat=2$, $Bathroom=5$: **Discontentment=29**
 - Действие : Visit-Bathroom ($Bathroom-4$)
 - afterwards: $Eat=4$, $Bathroom=0$: **Discontentment=16**
- Выбирается действие, которое приводит к минимальному дискомфорту
 - **Дискомфорт – число, которое мы пытаемся минимизировать**

Дискомфорт – один из способов задания функции полезности

Учёт времени выполнения действий 48

- Действия требуют **время на их выполнение**, что необходимо учитывать при принятии решений
- Действия в нескольких шагах от цели позволяют **оценить общее время** достижения цели
- Оценка времени делится на
 1. **Действия** требуют **время** на выполнение
 2. Иногда требуется **время** для **достижения нужного места** для начала выполнения действия
 - Например, может потребоваться 20 минут, чтобы пройти с одной части карты в другую
 - Это длинное путешествие для 5-минутного действия
- Необходимо учитывать также **время на поиск способа** решения задачи
 - В играх с сотнями объектов и тысячами возможных действий поиск вариантов решения для каждого действия – нецелесообразен
 - Обычно используется эвристика

Функция полезности с учётом времени

- Есть разные способы учитывать время
 - Можно включить время в «неудобство» или при вычислении функции полезности
 - Можно при прочих равных предпочитать действия, которые требуют меньше времени
- Пример, в котором персонаж хочет есть
 - Цель: $\text{Eat}=4$ **changing at+4 per hour**
 - Цель : $\text{Bathroom}=3$ **changing at+2 per hour**
 - Действие: $\text{Eat-Snack} (\text{Eat}-2)$ 15 minutes
 - afterwards: $\text{Eat}=2$, $\text{Bathroom}=3.5$: $\text{Discontentment}=21.25$
 - Действие : $\text{Eat-Main-Meal} (\text{Eat}-4)$ 1 hour
 - afterwards: $\text{Eat}=0$, $\text{Bathroom}=5$: $\text{Discontentment}=25$
 - Действие : $\text{Visit-Bathroom} (\text{Bathroom}-4)$ 15 minutes
 - afterwards: $\text{Eat}=5$, $\text{Bathroom}=0$: $\text{Discontentment}=25$
- Персонаж сначала будет **искать еду**, а уже потом **беспокоиться о ванной**

Вычисление цели, изменяющейся во времени⁵⁰

- В некоторых играх **скорость изменения целей** во времени фиксирована и задаётся дизайнером
 - The Sims имеет базовую скорость, с которой изменяются цели
- Простейший и наиболее эффективный способ **оценить скорость** изменения – регулярно **фиксировать изменения** каждой цели
 - Можно на каждом цикле запускать GOB и быстро проверять как изменились цели
- Возможное решение

$$\text{rateSinceLastTime} = \text{changeSinceLastTime} / \text{timeSinceLast}$$
$$\text{basicRate} = 0.95 * \text{basicRate} + 0.05 * \text{rateSinceLastTime}$$

- здесь 0.95 и 0.05 могут быть любыми, чтобы сумма была равна 1

Необходимость планирования⁵¹

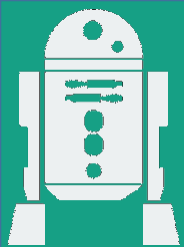
- Не важно, какой механизм выбора используется
 - Предполагаем, что **действия доступны** только если персонаж **может их выполнить**
 - Предполагаем, что персонаж не будет выбирать действия, которые не может выполнить
- **Действия могут зависеть от ситуации**
 - Это нормально, когда одно действие запрещает или разрешает другие!
- Предположим RPG-фэнтези, в которой магический персонаж использует **5 пунктов энергии** волшебной палочкой
 - Персонажу нужно лечение, но он также хочет победить великана
 - Цель: Heal=4
 - Цель : Kill-Ogre=3
 - Действие: **Fireball (Kill-Ogre-2) 3 energy-slots**
 - Действие : **Lesser-Healing (Heal-2) 2 energy-slots**
 - Действие : **Greater-Healing (Heal-4) 3 energy-slots**
 - Наиболее эффективная комбинация: “Lesser-Healing” + “Fireball”
 - Но “Lesser-Healing” оставляет персонаж в худшем состоянии, чем “Greater-healing”!
 - Простой алгоритм выбора предложит “Greater-healing”
- Чтобы персонаж учитывал **последствия нескольких действий**, необходимо **планирование действий**

Системы вероятностного вывода⁵²

- Stan
 - <https://mc-stan.org/>
- FACTORIE
 - <http://factorie.cs.umass.edu/>
- ProbLog
 - <https://dtai.cs.kuleuven.be/problog/>
- BLOG
 - <https://bayesianlogic.github.io/>
- WebPPL
 - <http://webppl.org/>
- ANGLICAN
 - <https://probprog.github.io/anglican/>
- Venture
 - <http://probcomp.csail.mit.edu/software/venture/>

Вопросы для самопроверки

- Какие есть проблемы действий в условиях неопределенности?
- В чем смысл теоремы де Финетти?
- Из чего состоит байесовская сеть?
- Какие есть точные/приближенные алгоритмы поиска в байесовской сети?
- Опишите как осуществляется "взвешивание" в алгоритме оценки с учётом правдоподобия?
- Что такое Марковский процесс? Опишите на примере с "зонтиком"
- Дайте определение скрытой марковской модели (НММ). Приведите примеры применения
- Опишите фильтр Калмана
- Что такое динамическая байесовская сеть?
- Опишите работу алгоритма фильтрации частиц
- Что такое ожидаемая полезность? Как её считать?
- Что такое многоатрибутная полезность? Какое отношение это имеет к принятию решений в группе?
- Что такое сети принятия решений? Как ими пользоваться?



Принятие сложных решений в условиях неопределенности

Тема 5

К.т.н., доцент

Беляев С.А.

Содержание

- Принятие сложных решений: проблема последовательного принятия решений в условиях неопределенности
- Действие на основе полезности, оптимальная стратегия
- Алгоритмы итерации по значениям в «Markov decision processes» и в «Partially observable Markov decision processes»
- Динамическая сеть принятия решений
- Дилемма заключенного, равновесие Нэша, турнир Аксельрода
- Мультиагентное принятие решений и кооперация
- Вероятностное программирование

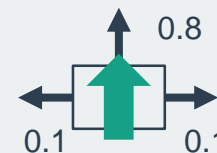
Проблема последовательного принятия решений

- **Полезность состояния** – ожидаемое общее вознаграждение от этого состояния и каждая попытка достижения финального состояния суммируется в каждом узле
- **В конце** каждой последовательности шагов вычисляется **наблюдаемое вознаграждение** для каждого состояния и обновляется оцененная полезность для каждого состояния
- **В пределе бесконечного** количества испытаний выборочное среднее будет сходиться к **истинному математическому ожиданию** в функции полезности
 - **Прямая оценка полезности** – пример контролируемого обучения, где каждый пример имеет состояние в качестве входных данных и **наблюдаемое вознаграждение** в качестве выходных данных
 - К сожалению, при этом упускается из виду, что **значения функции полезности не являются независимыми!**
 - **Полезность** каждого состояния равна его **собственной награде** плюс **ожидаемая полезность следующих состояний**

Стратегия $\pi(s)$ – что делать дальше в состоянии s ?
 π^* – обозначение оптимальной стратегии

Постановка задачи

3				+1
2				-1
1	start			
	1	2	3	4



Оптимальная стратегия

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

Полезности состояний (при $\gamma=1$)

3	0.85	0.91	0.96	+1
2	0.80		0.70	-1
1	0.74	0.70	0.65	0.43
	1	2	3	4

$$R(4,3)=1$$

$$R(4,2)=-1$$

$$R(x,y)=-0.04$$

За 10 шагов

от (1,1) до (4,3)

$$R=9*(-0.04)+1=0.64$$

Марковский процесс принятия решений (MDP)

1. Начальное состояние S_0
2. Модель перехода $P(s' | a, s)$
3. Функция вознаграждения $R(s, a, s')$

В MDP конечные $|S|$ и $|A(s)|$!!!

Стационарное отношение предпочтения

Вспоминаем
предыдущую
лекцию

- Дана последовательность действий и состояний
 - $[s_0, a_0, s_1, a_1, \dots, s_n]$
- Многоатрибутивная теория полезности → для получения простого выражения функции полезности для $[s_0, a_0, s_1, a_1, \dots, s_n]$
 - предположение о **независимости предпочтений**
 - **стационарность** $[s_0, a_0, s_1, a_1, \dots, s_n]$
 - нет ограничения по времени
 - если **#0** – сегодня и я предпочёл $[s_0, a_0, s_1, a_1, \dots, s_n]$, то если **#0** будет завтра, то я тоже предпочту $[s_0, a_0, s_1, a_1, \dots, s_n]$
- Следствие стационарности – ровно два вида функции предпочтений
 - **Аддитивные вознаграждения**
 - $U([s_0, a_0, s_1, a_1, \dots, s_n]) = R(s_0, a_0, s_1) + R(s_1, a_1, s_2) + \dots + R(s_{n-1}, a_{n-1}, s_n)$
 - **Обесцениваемые вознаграждения** (в большинстве случаев)
 - $U([s_0, a_0, s_1, a_1, \dots, s_n]) = R(s_0, a_0, s_1) + \gamma * R(s_1, a_1, s_2) + \dots + \gamma^n * R(s_{n-1}, a_{n-1}, s_n)$
 - γ – коэффициент обесценивания (от 0 до 1)

Выбор действия на основе полезности

- **Оптимальная стратегия** (возвращает действие)
 - $\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * U(s')]$
- **Полезность** s' равна сумме **непосредственного вознаграждения** за пребывание в s' и ожидаемой **обесцениваемой полезности следующего состояния**
 - $U(s) = \max_{a \in A(s)} \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * U(s')]$
 - **Уравнение Беллмана (Bellman equation)** – будет дальше многократно использоваться!
- **Применение к миру 4×3** (Up, Left, Down, Right)

$$U(1,1) = \max \{ [0.8(-0.04 + \gamma U(1,2)) + 0.1(-0.04 + \gamma U(2,1)) + 0.1(-0.04 + \gamma U(1,1))], \\ [0.9(-0.04 + \gamma U(1,1)) + 0.1(-0.04 + \gamma U(1,2))], \\ [0.9(-0.04 + \gamma U(1,1)) + 0.1(-0.04 + \gamma U(2,1))], \\ [0.8(-0.04 + \gamma U(2,1)) + 0.1(-0.04 + \gamma U(1,2)) + 0.1(-0.04 + \gamma U(1,1))] \}$$
 - Лучшее действие (при $\gamma=1$) – Up
- **Линейное масштабирование** функции вознаграждения R **не влияет** на результат
- **Q-функция** (функция **действие-полезность**, далее - **Q-value**)
 - $U(s) = \max_a Q(s, a)$
 - $\pi^*(s) = \operatorname{argmax}_a Q(s, a)$
 - $Q(s, a) = \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * U(s')] = \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * \max_{a'} Q(s', a')]$
- **Хранение в памяти**
 - простейшее решение – трёхмерные таблицы $|S|^2 \times |A|$. Для «мира 4×3»: $11^2 \times 4 = 484$
 - MDP расширяют узлами решения, вознаграждения и полезности – получают Dynamic Decision Networks (DDN)

Решение MDP (VALUE-ITERATION)

Итерация по значению

функция Итерация_по_значению(mdp, ϵ)

// mdp содержит состояния S , действия $A(s)$, модели $P(s' | s, a)$, $R(s, a, s')$, коэффициент γ

// ϵ – допустимая ошибка в функции полезности

$U = U' = [0, \dots, 0]$ // полезность – вектор длины $|S|$

повторять

$U \leftarrow U'$

$\delta \leftarrow 0$ // максимальная разница между функциями

для каждого s в S

$U'[s] \leftarrow \max \mathbf{Q-VALUE}(\text{mdp}, s, a, U)$ // по всем $a \in A(s)$

если $|U'[s] - U[s]| > \delta$ **тогда** $\delta \leftarrow |U'[s] - U[s]|$

пока $\delta \leq \epsilon * (1 - \gamma) / \gamma$

вернуть U

1. Начинаем с неизвестного состояния
2. Применяем обновление Беллмана к функции полезности
 - $U_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * U_i(s')]$
3. При стремлении к бесконечности достигаем равновесия

Итерация по значению

На примере «мира 3×4»

0	0	0	1
0		0	-1
0	0	0	0

Инициализация U
нулями, $|S| == 11$

-0.04	-0.04	0.79	1
-0.04		-0.04	-1
-0.04	-0.04	-0.04	-0.04

П	П	П	
П		Л	
П	П	П	Н

$\delta=0.79$

-0.08	0.52	0.86	1
-0.08		0.43	-1
-0.08	-0.08	-0.08	-0.08

П	П	П	
П		В	
П	П	П	Н

$\delta=0.56$

0.32	0.67	0.91	1
-0.11		0.52	-1
-0.11	-0.11	0.26	-0.11

П	П	П	
П		В	
П	П	В	Н

$\delta=0.40$

0.46	0.74	0.92	1
0.17		0.56	-1
-0.14	0.13	0.32	0.04

П	П	П	
В		В	
П	П	В	Л

$\delta=0.28$

0.55	0.76	0.93	1
0.33		0.58	-1
0.08	0.21	0.38	0.10

П	П	П	
В		В	
В	П	В	Л

$\delta=0.22$

$U(1,1) = \max\{ [0.8(-0.04+\gamma U(1,2))+0.1(-0.04+\gamma U(2,1))+0.1(-0.04+\gamma U(1,1))],$
 $[0.9(-0.04+\gamma U(1,1))+0.1(-0.04+\gamma U(1,2))],$
 $[0.9(-0.04+\gamma U(1,1))+0.1(-0.04+\gamma U(2,1))],$
 $[0.8(-0.04+\gamma U(2,1))+0.1(-0.04+\gamma U(1,2))+0.1(-0.04+\gamma U(1,1))] \}$

вверх
влево
вниз
вправо

$\gamma=0.9$
 $\epsilon=0.1$
 $\epsilon^*(1-\gamma)/\gamma=0.011$
 $R(x,y)=-0.04$
 $R(4,3)=1$
 $R(4,2)=-1$

Здесь:

П – вПраво;

Л – вЛево;

В – вВверх;

Н – вНиз;

Белые цифры –
значение U .

Получили оптимальную
стратегию на 5 шаге
алгоритма;
условие выхода ещё не
выполнено, т.к.

U ещё не сформирована
окончательно
(δ слишком велика)

Решение MDP (POLICY-ITERATION)

Итерация по стратегиям

функция Итерация_по_стратегиям(mdp)

// mdp содержит состояния S , действия $A(s)$, модели $P(s' | s, a)$,

// U – функция полезности

π = случайная_стратегия(mdp)

повторять

$U \leftarrow \text{Оценка_стратегии}(\pi, U, \text{mdp})$

без_изменений = true

для каждого s в S

$a^* \leftarrow \operatorname{argmax}_{a \in A(s)} \mathbf{Q-VALUE}(\text{mdp}, s, a, U)$

если $\mathbf{Q-VALUE}(\text{mdp}, s, a^*, U) > \mathbf{Q-VALUE}(\text{mdp}, s, \pi[s], U)$, **то**

$\pi[s] \leftarrow a^*$

без_изменений = false

пока без_изменений

вернуть π

функция Оценка_стратегии(π, U, mdp)

$U_{i+1}(s) = \sum_{s'} P(s' | s, \pi_i(s)) * [R(s, \pi_i(s), s') + \gamma * U_i(s')]$

вернуть U // Упрощенное обновление Беллмана

- За счёт отсутствия функции $\max()$ получаем набор линейных уравнений
- Для замкнутых систем решение всегда есть, сложность $O(n^3)$

Другие подходы к решению MDP

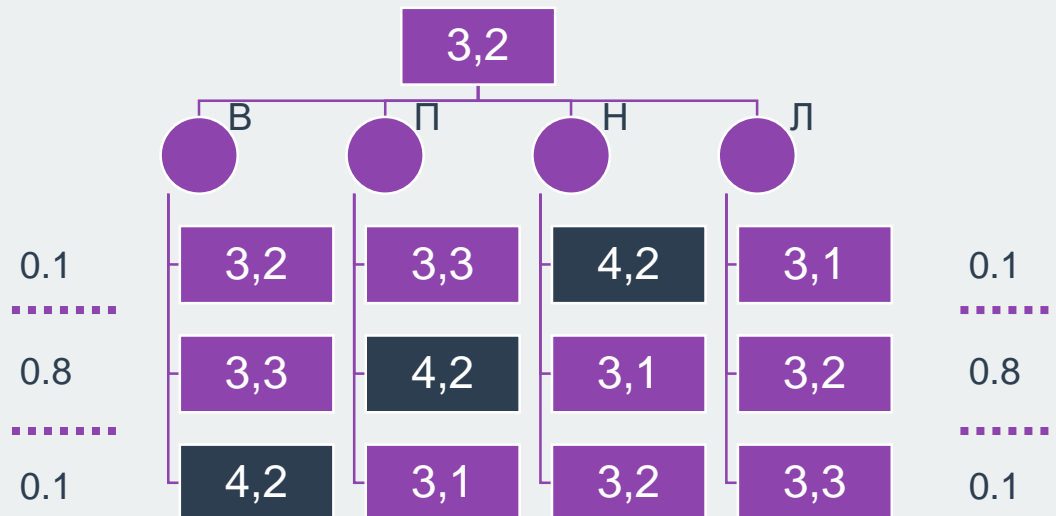
9

- **Линейное программирование**

- $U(s) \geq \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * U(s')]$

- **Online-алгоритмы**

- Для больших задач **offline** решение невозможно
 - Например, в **Тетрис** (10×20) 10^{62} состояний
 - Реализация **EXPECTIMINIMAX** (**модификация минимакса**), в который вводятся **случайные узлы**, функция оценки применяется к не терминальным узлам
 - Для «мира 3×4» решение находится в среднем за 50 ходов



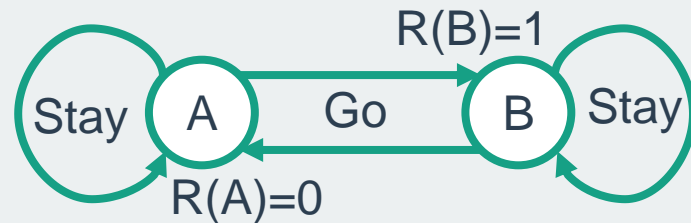
Узлы «4,2»
заканчиваются $R=-1$,
остальные узлы
раскрываются по
анalogии с «3,2»

MDP в частично наблюдаемых вариантах среды – POMDP

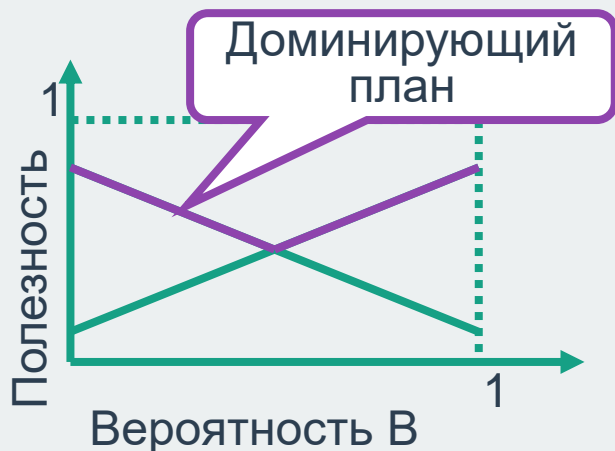
- «Появляются» наблюдение e_t (и модель наблюдения $P(E_t | X_t)$) и доверительное состояние b_t , «уходит» знание реального состояния x_t
- **Вычисление следующего доверительного состояния**
 - $b'(s') = \alpha * P(e|s') * \sum_s P(s' | s, a) * b(s)$
 - α – константа нормализации
 - Часто обозначают $b' = \alpha * \text{FORWARD}(b, a, e)$
- Например, для «мира 3×4»
 - $b_0 = \langle 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 0, 0 \rangle$
- В POMDP оптимальное действие зависит только от текущего доверительного состояния
 - Оптимальная стратегия $\pi^*(b)$ «маппит» доверительное состояние на действие
- **Основной цикл POMDP**
 1. Выполнить действие $a = \pi^*(b)$
 2. Получить результаты наблюдения e
 3. Установить новое $b' = \alpha * \text{FORWARD}(b, a, e)$, повторить с п.1
- **Вероятность наблюдать e при заданных a и b :**
 - $P(e|a, b) = \sum_{s'} P(e|s') * \sum_s P(s' | s, a) * b(s)$
- **Вероятность достичь b' при заданных a и b :**
 - $P(b' | b, a) = \sum_e P(b'|e, a, b) * \sum_{s'} P(e|s') * \sum_s P(s' | s, a) b(s)$
- **Решение POMDP в физическом пространстве сводится к решению MDP в пространстве доверительных состояний**

POMDP. Итерация по значениям

- Пусть дан мир из **двух состояний A и B**
- Есть **действия Stay и Go**, **точность выполнения = 0.9**
- **Вознаграждения** $R(A) = 0$, $R(B) = 1$
- **Дисконтирование** $\gamma = 1$
- **Точность определения состояния = 0.6**
- Необходимо разработать **условный план**
 - Вероятность нахождения в состоянии s обозначим $\mathbf{b}(s)$. $\mathbf{b}(A) + \mathbf{b}(B) = 1$
 - Фиксированный **план p** с **началом в состоянии s** назовём $\alpha_p(s)$
 - **Функция оценки ожидаемой полезности** для доверительного состояния \mathbf{b} обозначим $\sum_s \mathbf{b}(s)\alpha_p(s)$, или $\mathbf{b} \alpha_p$
 - Тогда $\mathbf{U}(\mathbf{b}) = \mathbf{U}^{\pi^*}(\mathbf{b}) = \max_p \mathbf{b} * \alpha_p$ – оптимальный план максимизирует функцию ожидаемой полезности для любого доверительного состояния
- Формулы **оценки плана α_p** (глубины $d=1$):
 - $\alpha_{[\text{Stay}]}(A) = 0.9 \cdot R(A, \text{Stay}, A) + 0.1 \cdot R(A, \text{Stay}, B) = 0.1$
 - $\alpha_{[\text{Stay}]}(B) = 0.1 \cdot R(B, \text{Stay}, A) + 0.9 \cdot R(B, \text{Stay}, B) = 0.9$
 - $\alpha_{[\text{Go}]}(A) = 0.1 \cdot R(A, \text{Go}, A) + 0.9 \cdot R(A, \text{Go}, B) = 0.9$
 - $\alpha_{[\text{Go}]}(B) = 0.9 \cdot R(B, \text{Go}, A) + 0.1 \cdot R(B, \text{Go}, B) = 0.1$



С вероятностью 0.9
останемся, с вероятностью
0.1 перейдём



Соответственно, можно построить 8 планов (глубины $d=2$):

[Stay; if Percept=A then Stay else Stay]

[Stay; if Percept=A then Stay else Go]

[Go; if Percept=A then Stay else Stay]

...

Если нарисовать их на графике, то большинство будет применимо только на ограниченном участке вероятности B.

Формула расчета плана p глубиной $(d-1)$ для подплана $p.e$ на основе восприятия e :

$$\alpha_p(s) = \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * \sum_e P(e | s') * \alpha_{p.e}(s')]$$

POMDP. Алгоритм итерации по значениям

функция POMDP-VALUE-ITERATION(pomdp, ϵ)

// pomdp включает $S, A(s), P(s' | s, a), P(e|s), R(s, a, s'), \gamma$

// ϵ – допустимая ошибка

// U, U' – наборы планов p , связанных с вектором a_p

$U' \leftarrow$ все 1-шаговые планы $[a]$ с $\alpha_{[a]}(s) = \sum_{s'} P(s' | s, a) * R(s, a, s')$

повторять

$U \leftarrow U'$

$U' \leftarrow$ все наборы планов для всех возможных действий и следующего восприятия в соответствии с

$$\alpha_p(s) = \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * \sum_e P(e|s') * \alpha_{p.e}(s')]$$

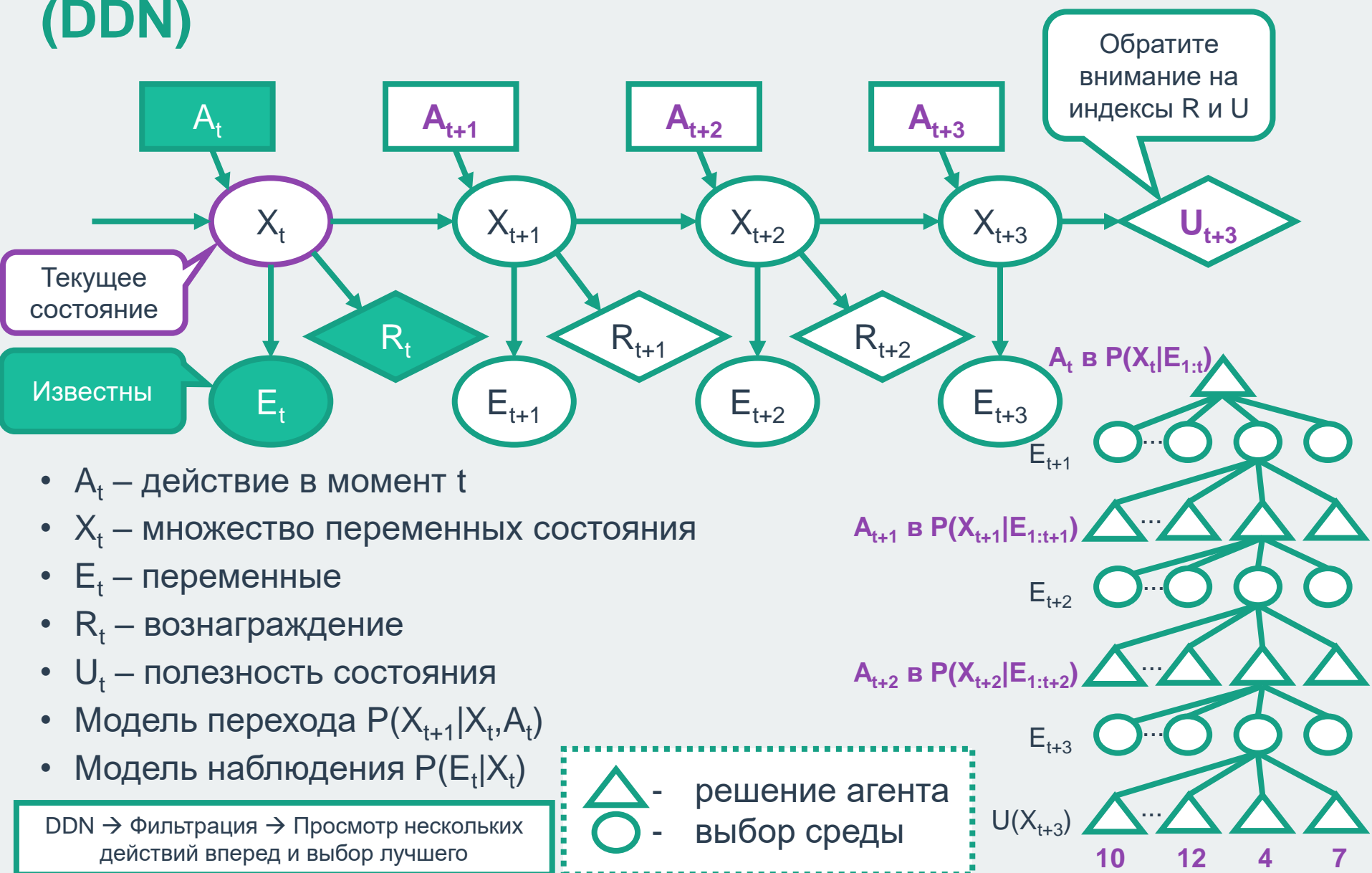
$U' \leftarrow \text{REMOVE-DOMINATED-PLANS}(U')$

пока MAX-DIFFERENCE(U, U') $\leq \epsilon(1-\gamma)/\gamma$

вернуть U

- Сложность алгоритма **очень велика** и на практике его в таком виде не используют

Универсальная структура динамической сети принятия решений – Dynamic Decision Network (DDN)



Мультиагентные системы

- **Применение теории игр**

- проектирование агентов
- проектирование механизмов

- **Подходы**

- кооперативная игра
 - общие цели
 - проблема координации
- не кооперативная игра

- **Решатель**

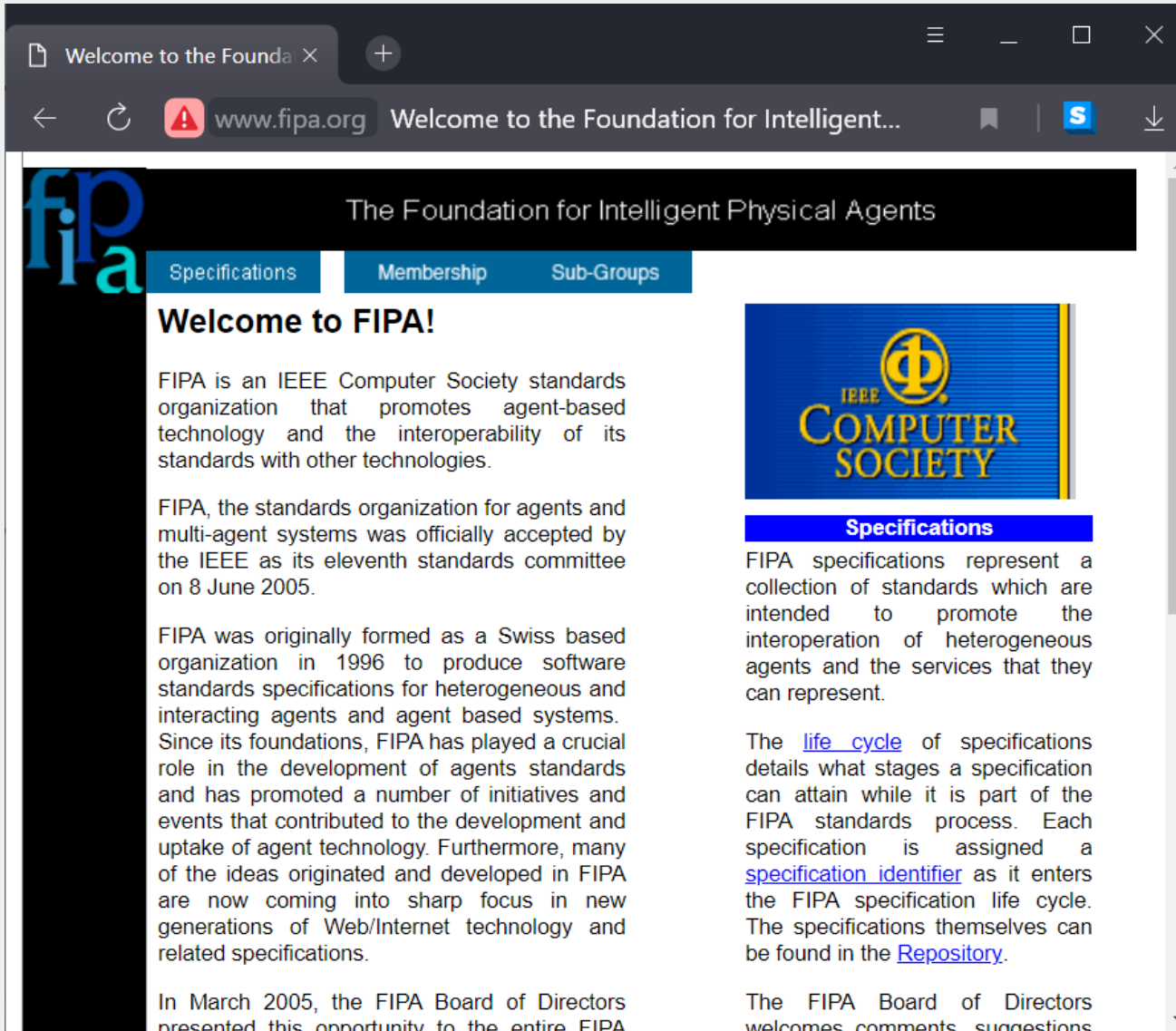
- общий
 - один на несколько агентов / тел
 - один на несколько эффекторов (говорить и ходить)
- распределенный
 - один на одного агента

- **Параллельность исполнения**

- последовательное исполнение
- реальная параллельность – соответствует реальному миру
- синхронизованное выполнение – существенное упрощение
 - каждая задача занимает фиксированное время и агенты выполняют их параллельно

Кооперативная игра	
Соглашение	Социальные законы
Коммуникации	Распознавание плана

Foundation for Intelligent Physical Agents (FIPA)



The screenshot shows the homepage of the Foundation for Intelligent Physical Agents (FIPA). The browser address bar displays 'www.fipa.org'. The website header features the FIPA logo and the text 'The Foundation for Intelligent Physical Agents'. Navigation tabs include 'Specifications', 'Membership', and 'Sub-Groups'. The main content area is titled 'Welcome to FIPA!' and contains three paragraphs of text. The first paragraph describes FIPA as an IEEE Computer Society standards organization. The second paragraph mentions FIPA's acceptance by the IEEE in 2005. The third paragraph details FIPA's history and mission. To the right, there is a blue box with the IEEE Computer Society logo and a section titled 'Specifications' which describes the purpose of FIPA specifications and mentions a 'life cycle' and 'specification identifier'. At the bottom, there is a section for 'The FIPA Board of Directors'.

Welcome to the Founda ×

← → www.fipa.org Welcome to the Foundation for Intelligent...

fipa The Foundation for Intelligent Physical Agents

Specifications Membership Sub-Groups


Welcome to FIPA!

FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.

FIPA, the standards organization for agents and multi-agent systems was officially accepted by the IEEE as its eleventh standards committee on 8 June 2005.

FIPA was originally formed as a Swiss based organization in 1996 to produce software standards specifications for heterogeneous and interacting agents and agent based systems. Since its foundations, FIPA has played a crucial role in the development of agents standards and has promoted a number of initiatives and events that contributed to the development and uptake of agent technology. Furthermore, many of the ideas originated and developed in FIPA are now coming into sharp focus in new generations of Web/Internet technology and related specifications.

In March 2005, the FIPA Board of Directors presented this opportunity to the entire FIPA



Specifications

FIPA specifications represent a collection of standards which are intended to promote the interoperation of heterogeneous agents and the services that they can represent.

The [life cycle](#) of specifications details what stages a specification can attain while it is part of the FIPA standards process. Each specification is assigned a [specification identifier](#) as it enters the FIPA specification life cycle. The specifications themselves can be found in the [Repository](#).

The FIPA Board of Directors welcomes comments suggestions

<http://www.fipa.org/>

- **JADE**
<https://jade.tilab.com/>
- **JIAC**
<https://www.jiac.de/>
- **GAMA**
<https://gama-platform.org/>
- **Jack**
<http://www.agent-software.com.au/>
- **Jadex**
<https://www.activecomponents.org/>
- **Spade**
<https://github.com/javipalanca/spade>

Нормальная форма игры

Стратегическая форма игры (нормальная форма игры) состоит из

- **Конечное множество I агентов**, обычно отождествляемых с целыми числами $I = \{1, \dots, n\}$
- **Набор действий A для каждого агента $i \in I$**
 - Назначение действия в A_i каждому агенту $i \in I$ является профилем действия
 - Мы можем рассматривать профиль действия как кортеж $\langle a_1, \dots, a_n \rangle$, который указывает, что агент i выполняет действие a_i
- **Функция полезности u_i для каждого агента $i \in I$** с учётом профиля действия возвращает ожидаемую полезность данного действия для агента i

Дилемма заключенного

Постановка задачи

- Двое предполагаемых грабителей, Алиса и Боб, были пойманы с поличным недалеко от места ограбления и допрошены отдельно
- Прокурор предлагает каждому сделку:
 - если вы дадите показания против вашего партнера как главаря, вы выйдете на свободу за сотрудничество, а ваш главарь будет отбывать 10 лет в тюрьме
 - если вы оба дадите показания друг против друга, вы оба получите 5 лет
- Алиса и Боб тоже это знают.
 - если оба откажутся давать показания, они будут отбывать только по 1 году каждый за меньшее обвинение в хранении украденного имущества
- Теперь Алиса и Боб сталкиваются с так называемой дилеммой заключенного:
 - должны ли они дать показания или отказаться?

Дилемма заключенного

Доминирующая стратегия обоих

Алиса анализирует матрицу выплат следующим образом:

- Допустим, Боб **даст показания**, тогда я **получу 5 лет**,
- если я **дам показания**, и **10 лет**, если **нет**,
- так что в таком случае **лучше давать показания**.

С другой стороны, если **Боб откажется**,

- тогда я **получу 0 лет**, если буду **давать показания**, и **1 год**, если **откажусь**,
- так что и в этом случае **лучше давать показания**.

Так что в любом случае мне **лучше дать показания**, так что я должна это сделать

	Alice: testify	Alice: refuse
Bob: testify	A = -5 B = -5	A = -10 B = 0
Bob: refuse	A = 0 B = -10	A = -1 B = -1

Формально в виде матрицы выигрыша:

j \ i	defect	coop
defect	2 2	1 4
coop	4 1	3 3

Давать показания – доминирующая стратегия игры

Равновесие Нэша (Nash Equilibrium)

19

В общем случае мы будем говорить, что две стратегии s_1 и s_2 находятся в **равновесии Нэша**, если:

1. в предположении, что агент i играет s_1 , агент j может играть не лучше, чем s_2 ; и
2. в предположении, что агент j играет s_2 , агент i не может сделать ничего лучше, чем играть s_1

Ни у одного из агентов **нет стимула отклоняться от равновесия Нэша**

К сожалению:

1. Не каждый сценарий взаимодействия имеет равновесие Нэша
2. Некоторые сценарии взаимодействия имеют более одного равновесия Нэша

Соревнования и взаимодействия с **нулевой суммой**

- Там, где предпочтения агентов диаметрально противоположны, мы имеем строго конкурентные сценарии
- Встречи с нулевой суммой – это те, где функции полезности суммируются до нуля:

$$u_i(\omega) + u_j(\omega) = 0 \\ \text{for all } \omega \in \Omega$$

- Нулевая сумма подразумевает **острую конкуренцию**
- Встречи с нулевой суммой в реальной жизни очень редки ... но люди склонны действовать во многих сценариях, как если бы они были нулевой суммой

Теорема Нэша

- Нэш доказал, что каждая **конечная** игра имеет равновесие Нэша в **смешанных стратегиях**
 - В отличие от случая с чистыми стратегиями
- Таким образом, этот результат преодолевает недостаток решений
 - но равновесие Нэша все еще может быть не одним...

Оптимальность Парето

- Результат считается **оптимальным по Парето** (или эффективным по Парето), если **нет** другого результата, который делает **одному агенту лучше, не делая другому агенту хуже**
- Если результат является оптимальным по Парето, то по крайней мере один агент будет неохотно **отходить от него** (потому что этому агенту будет хуже)

Сотрудничество в дилемме заключенного

- **Индивидуальное рациональное действие** — это признание, которое гарантирует **выигрыш не хуже 2**, тогда как **сотрудничество гарантирует выигрыш не более 1**
- Таким образом, признание является лучшим ответом на все возможные стратегии
 - Оба агента признаются, и получают выигрыш 2
- Но интуиция подсказывает, что это не лучший исход
 - Конечно, они **оба должны сотрудничать**, и каждый получит **вознаграждение** в размере 3!

j \ i	defect	coop
defect	2 2	1 4
coop	4 1	3 3

- Этот кажущийся парадокс является **фундаментальной проблемой мультиагентных взаимодействий**
- Это, по-видимому, подразумевает, что **сотрудничество не будет встречаться в обществах эгоистичных агентов**
- **Примеры из реального мира:**
 - сокращение ядерных вооружений ("почему бы мне не сохранить свое. . .")
 - бесплатные системы велосипедист — общественный транспорт
 - в Великобритании — телевизионные лицензии
- Дилемма заключенного распространена повсеместно
- **Можем ли мы восстановить сотрудничество?**

Аргументы в пользу восстановления сотрудничества²²

- Результаты анализа:
 - в теории игр понятие рационального действия неверно!
 - почему-то дилемма формулируется неправильно
- Аргументы для восстановления сотрудничества:
 - Мы не все Макиавелли!
 - Другой заключенный — мой близнец!
 - Тень будущего...
- Стратегия, которую вы действительно хотите сыграть в дилемме заключенного, такова:
 - Я буду сотрудничать, если он тоже будет
- Подходы к равновесию обеспечивают один из способов обеспечения этого
- Каждый агент представляет стратегию программы посреднику, который совместно выполняет стратегии
 - Принципиально важно, что стратегии могут быть обусловлены стратегиями других участников

Повторяющаяся Дилемма заключенного

- Один ответ:
 - играйте в эту игру не один раз
- Если вы знаете, что **снова встретитесь** с вашим противником, то стимул к признанию, кажется, испаряется
- **Сотрудничество – это рациональный выбор** в **бесконечно** повторяющейся дилемме заключенного (Ура!)

Обратная индукция

- Но ... предположим, вы оба знаете, что будете играть **ровно n раз** в раунде **$n - 1$** , у вас есть стимул к признанию, чтобы получить этот дополнительный бит выигрыша... но это делает раунд **$n - 2$ последним «реальным»**, и поэтому у вас есть стимул к признанию там тоже
Это проблема обратной индукции
- Играя дилемму заключенного с фиксированным, конечным, заранее определенным, общеизвестным количеством раундов, **признание** является лучшей стратегией

- Предположим, вы играете в **повторяющуюся дилемму заключенного против целого ряда противников...** какую стратегию вы должны выбрать, чтобы максимизировать общий выигрыш?
- Аксельрод (1984) исследовал эту проблему с помощью компьютерного турнира для программ, играющих дилемму заключенного

Стратегии

- **ALLD:**
 - «Всегда признаваться» - стратегия ястреба
- **TIT-FOR-TAT:**
 1. В раунде $u = 0$, сотрудничать
 2. В раунде $u > 0$ делайте то, что ваш противник сделал в раунде $u-1$
- **TESTER:**
 - В 1-м раунде – признание. Если соперник отомстил, то играй TIT-FOR-TAT. Иначе перемежаются сотрудничество и признание.
- **JOSS:**
 - Как TIT-FOR-TAT, за исключением периодически дефекта

Рецепты успеха в турнире Аксельрода

- Аксельрод предлагает следующие правила для успеха в своем турнире
 - **Не завидуйте:**
не играйте так, как будто это **нулевая сумма!**
 - **Будьте милы:**
начните с сотрудничества и ответьте взаимностью на сотрудничество
 - **Мстите соответствующим образом:**
всегда наказывайте признавшегося немедленно, но используйте "измеренную" силу — не переусердствуйте
 - **Не держите обид:**
всегда отвечайте взаимностью на сотрудничество немедленно

Игра в монету

- Игроки i и j **одновременно** выбирают «орел» либо «решка»
- Если они **выбирают одинаковую** сторону, то **выигрывает i** , а если они **выбирают разные** стороны, то **выигрывает j**

j \ i	heads	tails
heads	1 -1	-1 1
tails	-1 1	1 -1

Смешанные стратегии

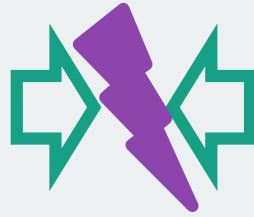
- В этой игре **нет (чистой стратегии) равновесия Нэша**
- Решение состоит в том, чтобы разрешить смешанные стратегии:
 - играть "орел" с вероятностью 0.5
 - играть "решка" с вероятностью 0.5

Теория кооперативных игр

27

- $G = (N, v)$ – игра
- $N = \{1, \dots, n\}$ – игроки
- v – характеристическая функция
 - $v(\{\}) = 0$ – всегда
 - $v(\{i\}) = 0$ – в некоторых играх
- $C \subseteq N$ – коалиции
 - $v(C) \geq 0$ для всех C
 - Например, для $N = \{1, 2, 3\}$
 - коалиции: $\{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{3, 1\}$ и $\{1, 2, 3\}$
 - структуры коалиции: $\{\{1\}, \{2\}, \{3\}\}, \{\{1\}, \{2, 3\}\}, \{\{2\}, \{1, 3\}\}, \{\{3\}, \{1, 2\}\}$ и $\{\{1, 2, 3\}\}$
- Если x_i – выплата i -му игроку
 - $\sum_i x_i = v(C)$ – всё распределяется между игроками
- **Супераддитивность v**
 - $v(C \cup D) \geq v(C) + v(D)$ для всех $C, D \subseteq N$
- **«Вменение» кооперации**
 - $\sum_i x_i = v(N)$ – все средства распределены
 - $x_i \geq v(\{i\})$ для всех $i \in N$ – можно заработать не хуже, чем одному

Работая вместе



- Доброжелательные (benevolent) агенты

- Если мы «владеем» всей системой, мы можем **создавать агентов, которые будут помогать друг другу**, когда нас попросят
- В этом случае мы можем предположить, что агенты доброжелательны: наши интересы - это их интересы
- Решение проблем в благожелательных системах – это **совместное распределенное решение проблем** (cooperative distributed problem solving – CDPS)
- Доброжелательность чрезвычайно упрощает задачу проектирования системы!

- Корыстные Агенты

- Если агенты представляют индивидов или организации (более общий случай), то мы не можем сделать предположение о доброжелательности
- Предполагается, что **агенты будут действовать в своих собственных интересах**, возможно, за счет других
- Конфликтный потенциал
- Может чрезвычайно усложнить задачу проектирования

Совместное использование задач и результатов

- Два основных способа совместного решения проблем:
 - **совместное использование задач** (task sharing): компоненты задачи распределяются между агентами
 - **обмен результатами** (result sharing): информация (частичные результаты и т.п.) распространяется
- Хорошо известный протокол совместного использования задач для распределения задач — это **сеть контрактов** (contract net)
 - Распознавание
 - Recognition
 - Объявление
 - Announcement
 - Торги
 - Bidding
 - Награждение
 - Awarding
 - Ускорение
 - Expediting

Сеть контрактов

Распознавание, Объявление

Распознавание

- На этой стадии агент осознает, что у него **есть проблема**, с которой нужна помощь.
У агента есть цель или...
 - осознает, что **не может достичь цели один** — не имеет возможности
 - понимает, что **предпочел бы не достигать цели один** (как правило, из-за качества решения, крайнего срока и т.п.)

Объявление

- На этом этапе агент с заданием отправляет **объявление задачи**, которое включает в себя спецификацию задачи, которая должна быть решена
- **Спецификация** должна кодировать:
 - **описание** самой задачи (возможно, исполняемой)
 - любые **ограничения** (например, сроки, ограничения качества)
 - информация о **мета-задаче** (например, «заявки должны быть представлены...»)
- Затем объявление распространяется

Сеть контрактов

Торги, Награждение & Ускорение

Торги

- Агенты, получившие объявление, сами решают, **хотят ли они участвовать** в конкурсе на выполнение задания
- Факторы:
 - агент должен решить, **способен ли он ускорить выполнение задачи**
 - агент должен определить **ограничения качества** и информацию о ценах (если это необходимо)
- Если они решат принять участие в торгах, то они подтверждают участие в тендере

Награждение & Ускорение

- Агент, отправивший объявление о задании, должен выбрать между предложениями и решить, **кому «отдать контракт»**
- Результат этого процесса **доводится до сведения** агентов, подавших заявку
- Затем успешный **подрядчик ускоряет выполнение** задачи
- Может включать в себя формирование дальнейших отношений между менеджером и подрядчиком: **субподряд**

Сеть контрактов

Особенности

- Как...

- ... конкретизировать задачи?
- ... определить качество обслуживания?
- ... выбрать между конкурирующими предложениями?
- ... дифференцировать предложения по нескольким критериям?

- Подход к распределенному решению задач, основывается на **распределении задач**
- Распределение задач рассматривается как своего рода **переговоры по контракту**
- «Протокол» определяет **содержание коммуникации**, а не только ее форму
- Двусторонняя передача информации является естественным продолжением механизмов передачи управления

Сеть контрактов

Подход к переговорам агентов

- **Совокупность связанных узлов** – это «сеть контрактов»
- **Каждый узел** сети (агент) может в разное время или для разных задач быть **менеджером** или **подрядчиком**
- Когда узел получает **составную задачу** (или по какой-либо причине не может решить свою текущую задачу), он **разбивает** ее **на подзадачи** (если это возможно) и **объявляет тендер** (действуя в качестве менеджера), **получает предложения** от потенциальных подрядчиков, а затем **присуждает задание** (пример домена: управление сетевыми ресурсами, принтеры, ...)

Типы сообщений

1. Объявление задачи
2. Заявка
3. Награда
4. Промежуточный доклад
 - о ходе
5. Заключительный отчет
 - в том числе описание результата
6. Сообщение о прекращении контракта
 - если менеджер хочет расторгнуть договор

Общение – speech act (1)

- **Остин** заметил, что некоторые высказывания скорее похожи на «физические действия», которые, по-видимому, изменяют состояние мира
- Примеры парадигмы:
 - объявление войны
 - крещение
 - «Объявляю вас мужем и женой»
- Но в более общем плане все, что мы произносим, произносится с намерением удовлетворить какую-то цель или намерение
- Теория того, как высказывания используются для достижения намерений, – это теория общения (speech act theory)
- **Searle** (1969) выделил различные типы речевого акта:
 - **representatives:** такие как информирование, например, "идет дождь"
 - **directives:** попытки заставить слушателя сделать что-то, например: "пожалуйста, приготовьте чай"
 - **commissives:** которые заставляют говорящего что-то делать, например: "я обещаю..."
 - **expressives:** посредством которых говорящий выражает свое ментальное состояние, например: "Спасибо!"
 - **declarations:** например, объявление войны или Крещение

Общение – speech act (2)

- Есть некоторые споры о том, корректна (или нет!) **типология речевых актов**
- В общем, речевой акт можно рассматривать как имеющий две составляющие
 - **перформативный глагол**: (например, просить, информировать, обещать, ...)
 - **пропозициональное содержание**: (например, "дверь закрыта")
- Предположим:
 - **performative** = request
content = "the door is closed"
speech act = "please close the door"
 - **performative** = inform
content = "the door is closed"
speech act = "the door is closed!"
 - **performative** = inquire
content = "the door is closed"
speech act = "is the door closed?"

Семантика на основе плана

- Пример **семантики запроса**:

request(s, h, ϕ)

pre:

- s believe h can do ϕ
(**you don't ask someone to do something unless you think they can do it**)
- s believe h believe h can do ϕ
(**you don't ask someone unless they believe they can do it**)
- s believe s want ϕ
(you don't ask someone **unless you want it!**)

post:

- h believe s believe s want ϕ
(the effect is to make them **aware of your desire**)

- Как определить семантику речевых актов? **Когда можно сказать, что кто-то произнес, например, просьбу или сообщение?**
- Cohen & Perrault (1979) определили семантику речевых актов с помощью **формализации** списка предварительного **условия-удаления-добавления**
- Заметьте, что **говорящий не может** (как правило) **заставить** слушателя **принять** какое-то желаемое **психическое состояние**

- Теперь рассмотрим языки коммуникации агентов (ACL) — **стандартные форматы обмена сообщениями**
- Наиболее известным ACL является **KQML**, разработанный инициативой ARPA по обмену знаниями
- KQML состоит из двух частей:
 - язык запросов и манипуляций знаниями (**KQML**)
 - формат обмена знаниями (**KIF**)
- KQML — это «внешний» язык, который **определяет** различные приемлемые «коммуникативные глаголы», или **перформативы**, примеры перформативов:
 - спросите-если («это правда, что. . .»)
 - выполните («пожалуйста, выполните следующее действие. . .»)
 - скажите («это правда. . .»)
 - ответьте («Ответ есть . . .»)
- KIF-это язык для выражения содержания сообщения

Используется для
указания

- **Свойства вещей** в домене (например, «Иван-председатель»)
- **Отношения между вещами** в домене (например, «Иван босс Сергея»)
- **Общие свойства** домена (например, «Все студенты зарегистрированы по крайней мере для первого семестра»)

- “The temperature of m1 is 83 Celsius”:
(= (temperature m1) (scalar 83 Celsius))
- “An object is a bachelor if the object is a man and is not married”:
(defrelation bachelor (?x) :=
 (and (man ?x) (not (married ?x))))
- “Any individual with the property of being a person also has the property of being a mammal”:
(defrelation person (?x) :=> (mammal ?x))

- Если дано **утверждение** ϕ , а ω – **возможные миры**, тогда
 - $P(\phi) = \sum_{\omega: \phi \text{ is true in } \omega} P(\omega)$
 - $P(\phi|e)$ – условная вероятность определяется также
- Вероятностное программирование на Прологе (ProbLog)
 - <https://dtai.cs.kuleuven.be/problog/>
- Для каждого факта и правила определяется **его вероятность** (в данном примере 0.3, 0.4 и 0.5)

0.5::c1.

0.5::c2.

0.3::e :- c1.

0.4::e :- c2.

evidence(c1,true).

evidence(c2,true).

query(e).

- Результат

$p(e) = 0.58$

- Объяснение

$p(e) = 1 - (1 - 0.3) * (1 - 0.4) = 0.58$

0.5::c1.

0.5::c2.

0.3::e :- c1.

0.4::e :- c2.

evidence(c1,true).

query(e).

- Результат

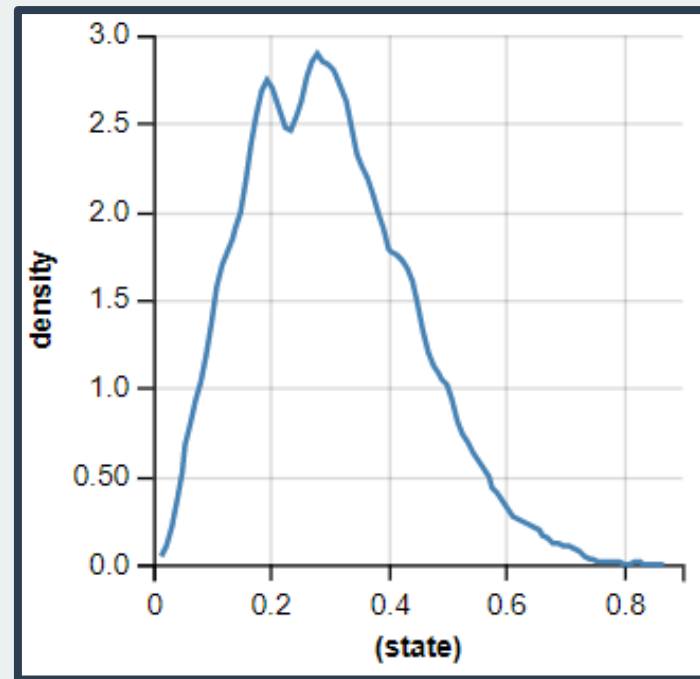
$p(e) = 0.44$

- Объяснение

$p(e) = 1 - (1 - 0.3) * (1 - 0.4 * 0.5) = 0.44$

Использование разных законов распределения

- Язык WebPPL на JavaScript
 - <http://webppl.org/> <http://dippl.org/> <https://agentmodels.org/>
- Случайные переменные
 - Бернулли (`flip(p)` или `Bernoulli({p: p})`)
 - `sample(Bernoulli({ p: 0.5 }))`
 - Результат true или false
 - Бета (`Beta({a: ..., b: ...})`)
 - `Beta({a: 3, b: 7})` →
 - `sample(Beta({a: 3, b: 7}))`
 - Результат: (1) график, (2) число
 - `Binomial({p: 0.3, n: 17})`
 - `Discrete({ps: [2,4,6]})`
 - `Gaussian({mu: 2, sigma: 1})`
 - ...
- <https://webppl.readthedocs.io/en/master/distributions.html>



Вычисление вероятности

```
var twoHeads = Infer({  
  model() { // Модель из трех монет  
    var a = flip(0.5);  
    var b = flip(0.5);  
    var c = flip(0.5);  
    condition(a + b + c === 2); // Две - аверс  
    // condition(a + b + c >= 2); // Две или три - аверс  
    return a; // Какова вероятность, что первая - аверс?  
  }  
});  
print('Вероятность, что первая аверс (при условии, что две аверс) : ');  
print(Math.exp(twoHeads.score(true)));
```

- Результат

Вероятность, что первая аверс (при условии, что две аверс) :

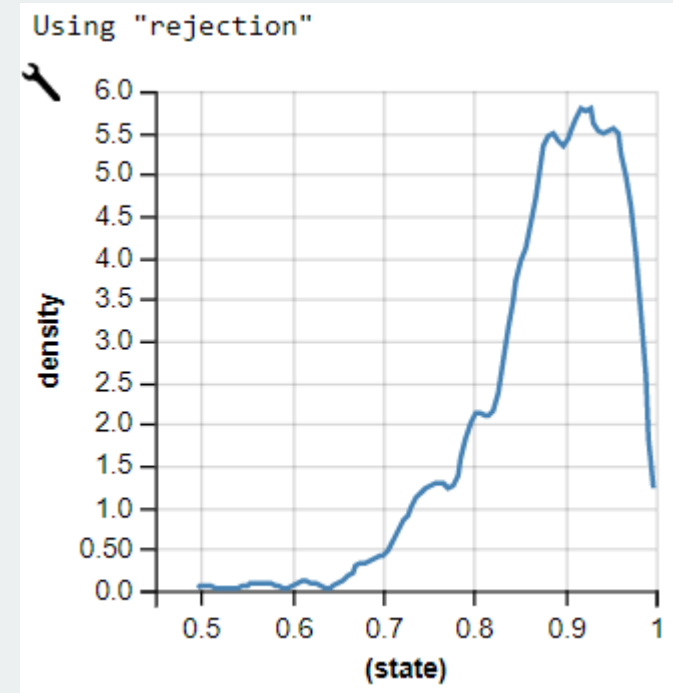
0.6666666666666666

Задание результатов наблюдения

42

```
var model = function(){  
  var coin_weight = uniform(0, 1)  
  observe(Binomial({n: 15, p: coin_weight}), 14)  
  return coin_weight  
}  
Infer(model)
```

Монету подбросили 15 раз и 14 раз выпал орел.
Вопрос: какой баланс у монеты?



Максимизация функции полезности

43

```
var argMax = function(f, ar){ // Вспомогательная
  return maxWith(f, ar)[0]
};
var actions = ['italian', 'french']; // Выбор кухни
var transition = function(state, action) { // Переход
  if (action === 'italian') return 'pizza';
  return 'steak frites';
};
var utility = function(state) { // Функция полезности
  if (state === 'pizza') return 10;
  return 0;
};
var maxAgent = function(state) { // Максимизация
  return argMax(
    function(action) { // Вызов функции
      return utility(transition(state, action));
    }, // Все действия
    actions);
};
print('Choice : ' + maxAgent('initialState'));
```

- Максимизируется

$$\operatorname{argmax}_{a \in A} U(T(s, a))$$

- Результат

Choice : italian

- Если изменить в utility 0 на 20, то будет результат

Choice : french

Максимизация ожидаемой полезности

```

var argMax = function(f, ar){ // Вспомогательная
  return maxWith(f, ar)[0]
};
var actions = ['italian', 'french']; // Действия
var transition = function(state, action) { // Ожидаемый переход
  var nextStates = ['bad', 'good', 'spectacular'];
  var nextProbs = (action === 'italian') ? [0.2, 0.6, 0.2] : [0.05, 0.9, 0.05];
  return categorical(nextProbs, nextStates);
};
var utility = function(state) { // Полезность перехода
  var table = {
    bad: -10,
    good: 6,
    spectacular: 8
  };
  return table[state];
};
var maxEUAgent = function(state) { // Ожидаемая полезность
  var expectedUtility = function(action) {
    return expectation(Infer({
      model() {
        return utility(transition(state, action));
      }
    }));
  };
  // Максимизация ожидаемой полезности
  return argMax(expectedUtility, actions);
};
print(maxEUAgent('initialState')); // Посчитать переход

```

- Максимизируется

$$\max_{a \in A} E(U(T(s, a)))$$

- Результат

french

- Если изменить [0.2, 0.6, 0.2] на [0.02, 0.6, 0.38], то результат будет

italian

Реализация HMM на WebPPL

45

```
var transition = function(s) { // Модель перехода
  return s ? flip(0.7) : flip(0.3)
}

var observeState = function(s) { // Модель наблюдения
  return s ? flip(0.9) : flip(0.1)
}

var hmm = function(n) { // HMM на n шагов (S[0] = true)
  var prev = (n==1) ? {states: [true], observations: []} : hmm(n-1)
  var newState = transition(prev.states[prev.states.length-1])
  var newObs = observeState(newState)
  return { // Добавляем к предыдущим
    states: prev.states.concat([newState]),
    observations: prev.observations.concat([newObs])
  }
}

var trueObs = [false, false, false] // Реальные наблюдения
var model = function(){
  var r = hmm(3) // Задаем три шага HMM
  factor(_.isEqual(r.observations, trueObs) ? 0 : -Infinity)
  return r.states
};

viz.table(Infer({ model }))) // Визуализируем модель
```

0	1	2	3	probability
true	false	false	false	0.8296918550634864
true	true	false	false	0.092187983895943
true	false	false	true	0.03950913595540412
true	false	true	false	0.01693248683803033
true	true	true	false	0.010243109321771443
true	false	true	true	0.004389903995044901
true	true	false	true	0.004389903995044901
true	true	true	true	0.0026556209352740735

Вероятностные модели «открытого» мира – Open-Universe Probability Models

- **Неопределенность**

- существования объекта
 - книги/покупателя может не быть
- идентификации
 - покупатель может создать несколько фейковых аккаунтов

- **Распространение феномена неопределённости**

- неизвестно, что за углом?
- неизвестно, а объект тот же, что наблюдался недавно?
- неизвестно, «Мэри», «Миссис Смит», «она», «их кардиолог», «его мать» – это одно и тоже лицо?

- Для описания часто используют **семантику логики первого порядка**

- **Примеры**

- `#Customer ~ UniformInt(1,3).`
- `#Book ~ UniformInt(2,4).`
- `#LoginID(Owner=c) ~ if Honest(c) then Exactly(1) else UniformInt(2,5).`

- В примере использовано равномерное (uniform) распределение, обычно для положительных чисел используется **распределение Пуассона**

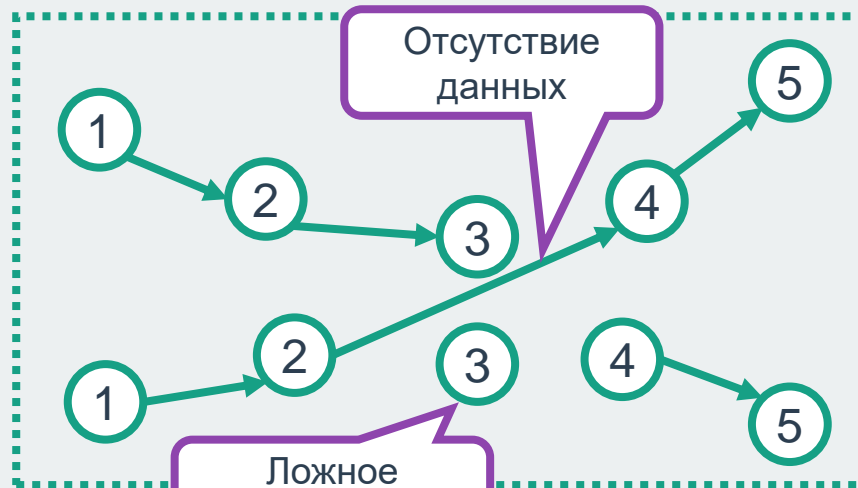
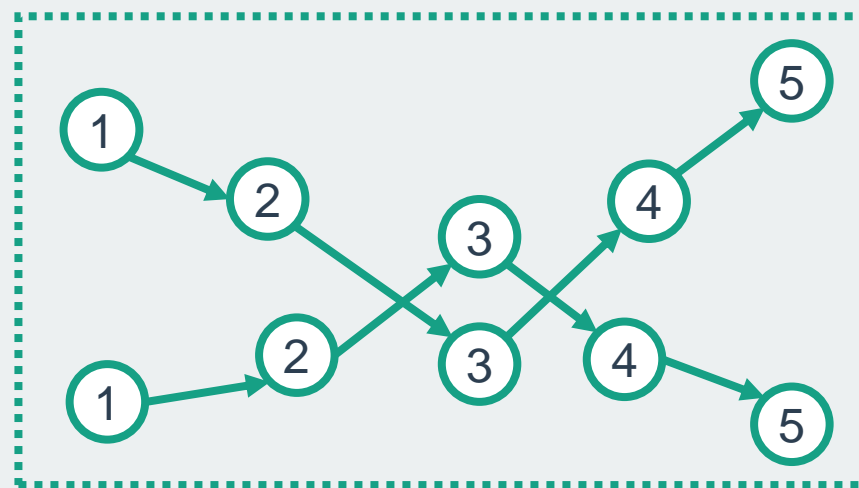
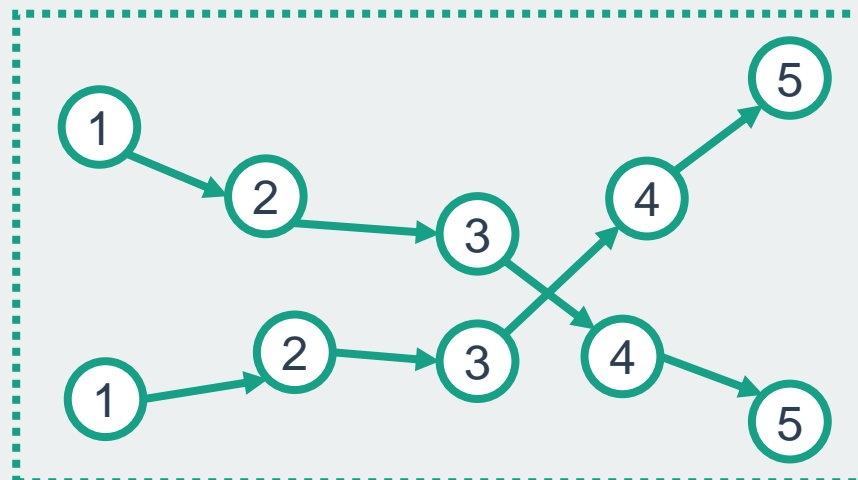
- $P(X = k) = \lambda^k * e^{-\lambda} / k!$, здесь λ – дисперсия

- В качестве алгоритма вероятностного вывода часто используется **МСМС**

- Markov chain Monte Carlo, алгоритм Монте-Карло по схеме марковской цепи

Отслеживание сложного мира

Пример измерений от 2 объектов



guaranteed Aircraft A_1, A_2

$X(a, t) \sim \text{if } t = 0 \text{ then InitX() else } N(F X(a, t-1), \Sigma_x)$

$\#Blip(\text{Source}=a, \text{Time}=t) = 1$

$Z(b) \sim N(H X(\text{Source}(b), \text{Time}(b)), \Sigma_z)$

F – матрица перехода, Σ_x – её шум

H – матрица наблюдения, Σ_z – её шум

здесь

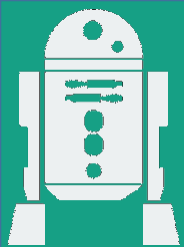
Ложное
срабатывание

Для n объектов и T моментов времени есть $(n!)^T$ вариантов назначения

Вопросы для самопроверки

48

- В чём заключается проблема последовательного принятия решений в условиях неопределенности?
- Как выбирается действие на основе полезности? Что такое оптимальная стратегия?
- Что такое MDP? Как реализуется алгоритм итерации по значениям в MDP?
- Чем отличается итерация по стратегиям? Это точный алгоритм?
- Что такое POMDP? Как реализуется алгоритм итерации по значениям в POMDP?
- Что такое динамическая сеть принятия решений?
- Что такое равновесие Нэша? Как оно проявляется в дилемме заключенного? Какие стратегии применимы в турнире Аксельрода?
- На чем основывается кооперация в мультиагентных системах? Как и зачем реализуется общение агентов?
- Что такое вероятностное программирование?
- Что такое модели "открытого" мира? В чём могут возникать проблемы в этом мире?



Обучение

Тема 6

К.т.н., доцент

Беляев С.А.

Содержание

- Обучение на основе наблюдений, контролируемое обучение
- Регрессионные деревья, деревья классификации, деревья принятия решений
- Статистические методы обучения, байесовское обучение
- Обучение по методу наибольшего правдоподобия
- Нейронные сети, сверточные нейронные сети, рекуррентные нейронные сети
- Пассивное и активное обучение с подкреплением

Виды обучения

- **Контролируемое обучение**
 - Обучение на примере входных-выходных данных
- **Неконтролируемое обучение**
 - Выявление закономерностей во входных данных
- **Обучение с подкреплением**
 - За действие (или последовательность) даётся вознаграждение

Контролируемое обучение

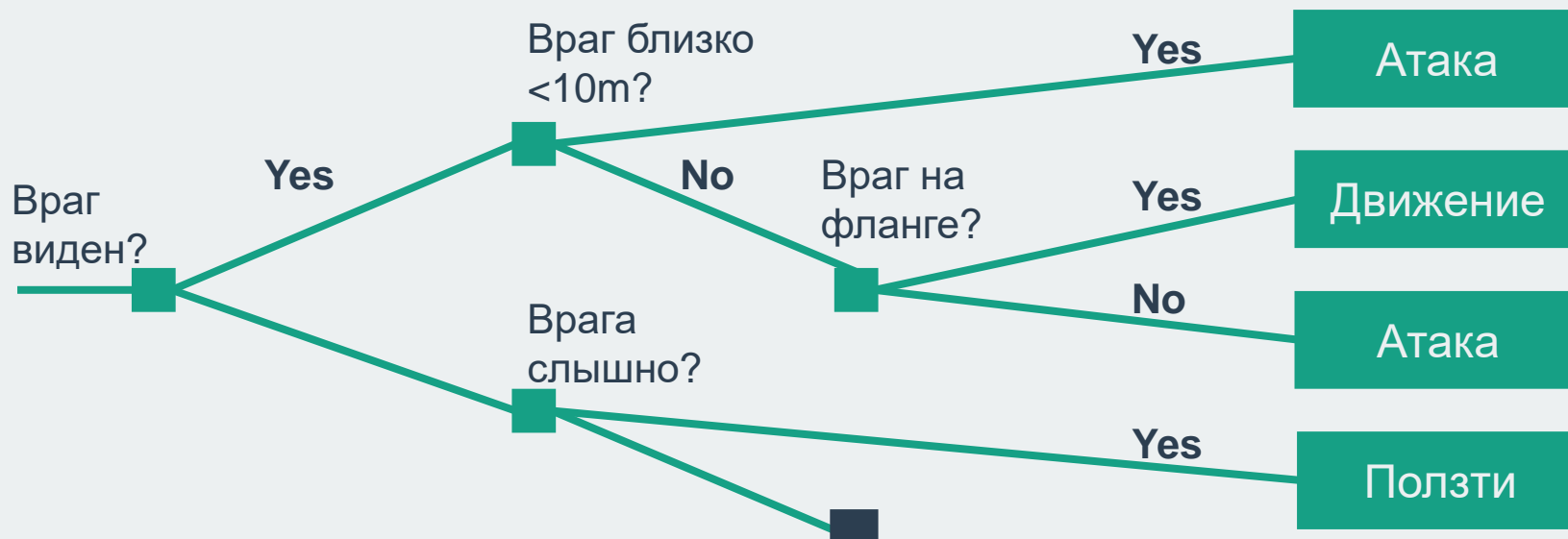
- **Дан обучающий набор**
 - $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$
 - Пары сгенерированы неизвестной функцией $y = f(x)$
- **Необходимо найти гипотезу h , аппроксимирующую f**
 - Поиск гипотезы, согласованной с наблюдаемыми примерами
- **Цель**
 - $h^* = \operatorname{argmax}_{h \in H} P(h|\text{data})$
- **Обучение** принятию решений **не может заменить базовые инструменты принятия решений**
- **Структура подхода** к обучению принятию решений
 - Персонаж имеет **набор опций поведения**, из которых он может сделать выбор
 - Дополнительно, у него есть **наблюдаемые параметры игры**
 - Необходимо **обучить связи** между решениями (в форме поведения – выбранной опции) с результатами наблюдения

Обучение принятию решений

- Чему следует учить?
 - Количество наблюдаемых элементов данных будет огромным, **количество действий – ограничено**
 - Качественное обучение требует, чтобы персонаж в результате обучения **выполнял действия «компетентно»**
 - Сложнее всего будет **обучить нюансам** действий в конкретной ситуации
 - Проще всего сразу добавить **базовые правила**, чтобы в целом корректное поведение было получено сразу
- **В играх** обычно используют следующие техники
 - **Наивная Байесовская классификация**
 - Просто реализуется и даёт хороший базис для более сложных техник
 - **Обучение деревьев решений**
 - Важное удобство деревьев в том, что они могут анализироваться человеком
 - **Обучение с подкреплением**
 - Будет рассмотрено в следующей теме
 - **Нейронные сети**

Decision trees (DT)

- DT – простейшая техника принятия решений
- Используется для **управления персонажем** или для других задач **принятия решений**, таких как управление анимацией
 - Используется от анимации до сложных тактических или стратегических задач ИИ
- Очень **модульные и легко создаются**
- Деревья решений могут обучаться
 - Обучение очень быстрое, если сравнивать с нейронными сетями или генетическими алгоритмами

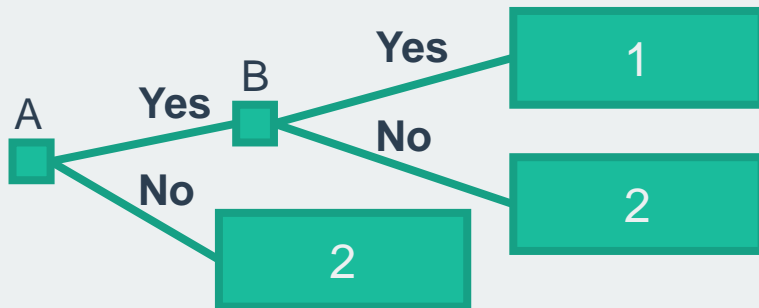


Деревья решений

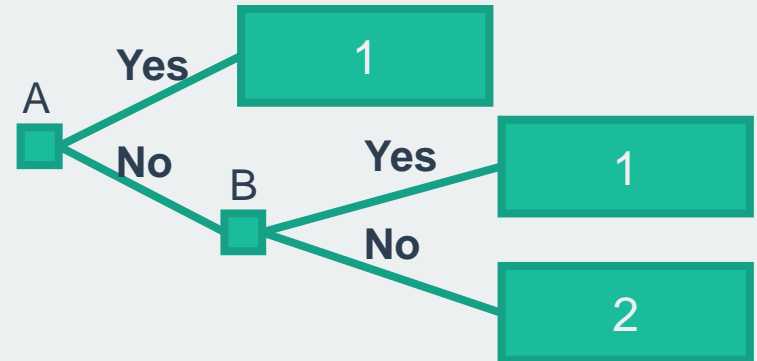
- Деревья состоят из **узлов принятия решений**
- У дерева **есть корень**
- Для каждого решения, начиная с корня, **выбирается только одна ветвь**
- Алгоритм **проходит по дереву**, делая выбор, пока не доберётся **до листа**
- Каждый **лист** дерева содержит указание на **действие**, которое необходимо выполнить
- Как только добрались до **действия**, действие необходимо **выполнить немедленно**

Комбинации решений

- Деревья решений эффективны, потому что решения – **очень простые**
 - Каждое **решение** – одна проверка
- Если требуются **логические комбинации**
 - Для организации **И** – условия ставятся последовательно
 - Для организации **ИЛИ** – условия ставятся последовательно, но в отрицании



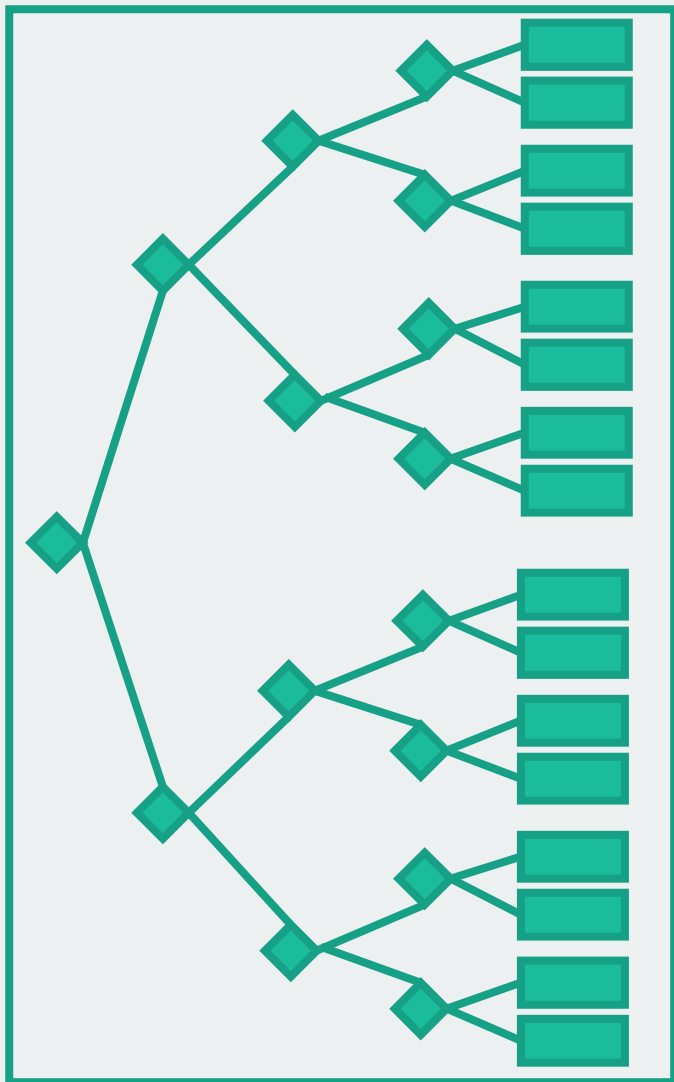
If **A AND B** then action 1, otherwise action 2



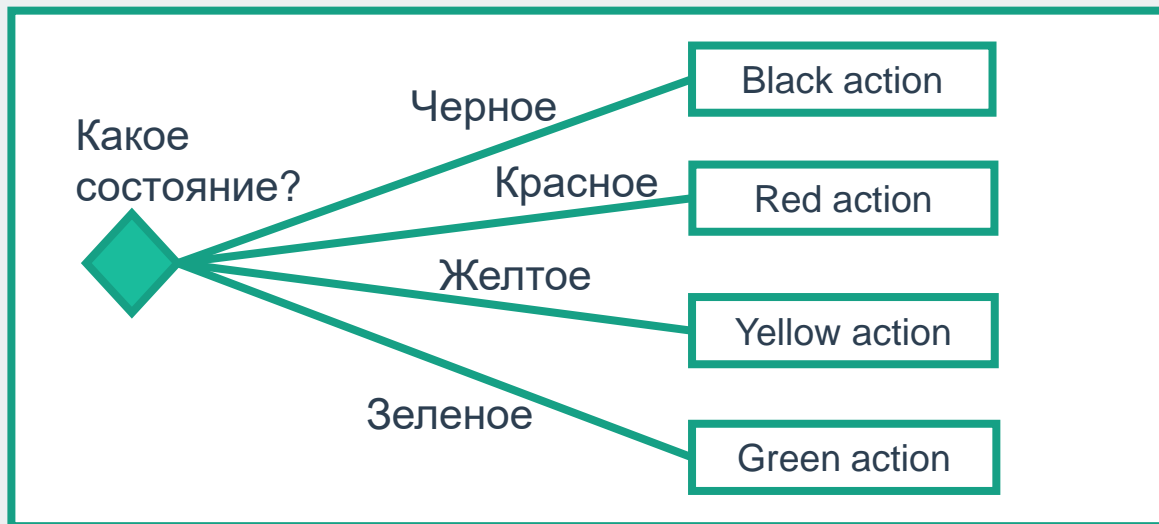
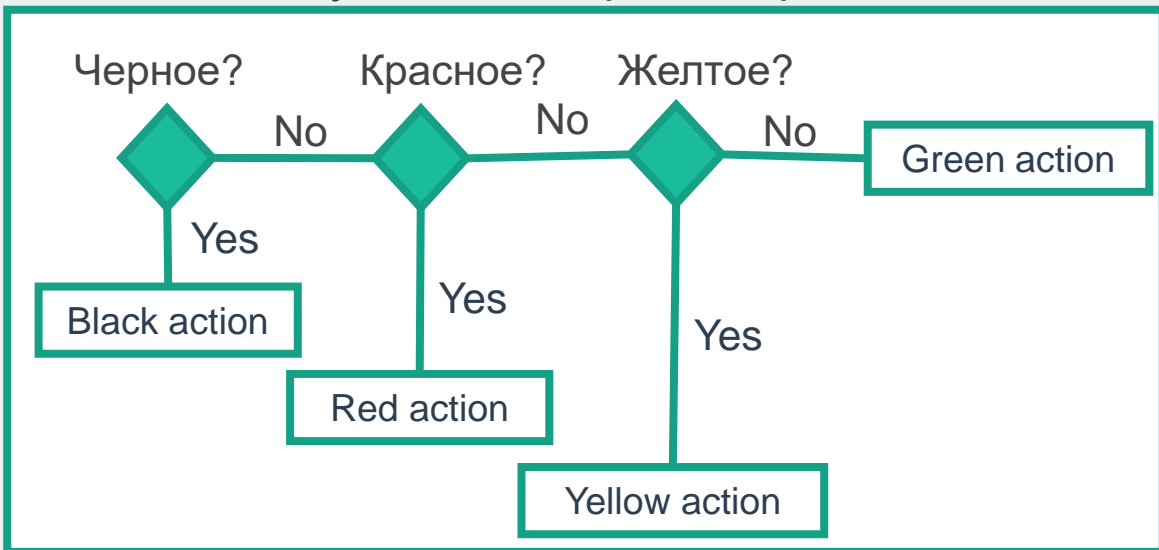
If **A OR B** then action 1, otherwise action 2

Сложность решений

Ветвление

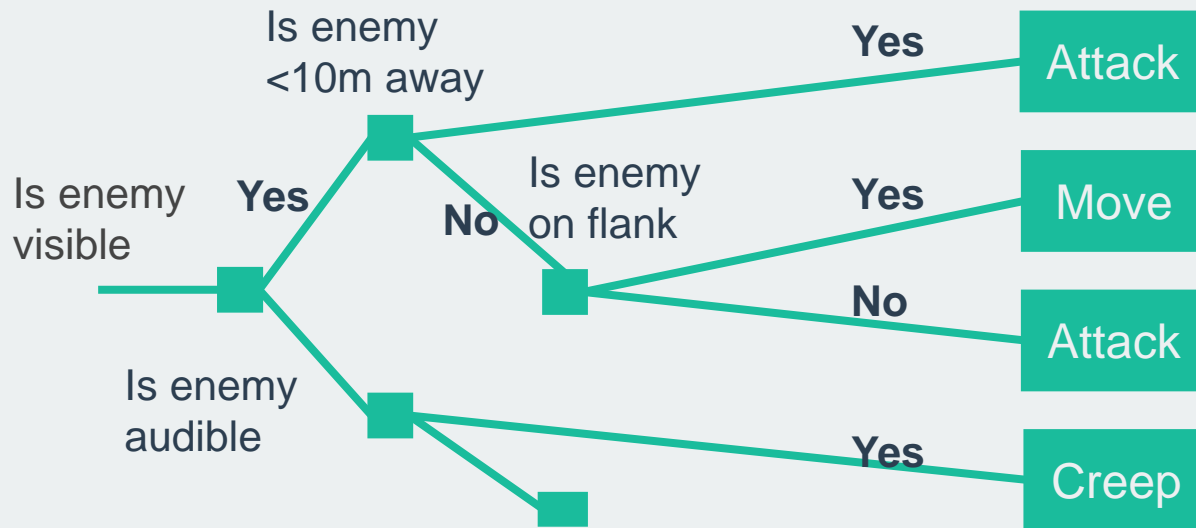


Глубокое бинарное дерево



Плоское дерево с 4 ветвями

Представление деревьев решений



```
{  
  question: "Is enemy  
  visible?",  
  yes: {  
    question: "Is enemy  
    <10m away",  
    yes: { action: "Attack"  
    },  
    no: { ... }  
  },  
  no: { ... }  
}
```

JSON

```
<question> Is enemy visible?  
  <question if="yes">Is enemy <10m away?  
    <answer if="yes">Attack</answer>  
    <question if="no">Is enemy on flank  
      <answer if="yes">Move</answer>  
      <answer if="no">Attack</answer>  
    </question>  
  ...  
</question>
```

XML

```
<decision_tree>  
  <decision id="1" yes="2" no="3">  
    Is enemy visible?  
  </decision>  
  <decision id="2" yes="4" no="5">  
    Is enemy <10m away?  
  </decision>  
  <action id=4>Attack</action>  
  ...  
</decision_tree>
```

XML

Обучение деревьев решений

11

- Деревья решений могут быть эффективно обучены
 - Построены динамически на основе наборов наблюдений и правильных действий
- Обученные деревья могут использоваться обычным образом для принятия решений
- Существует **широкий диапазон методов обучения** деревьев решений, используемых для классификации, предсказания, статистического анализа
- **ID3** сокращение для “**Inductive Decision tree algorithm 3**” или “**Iterative Dichotomizer 3**”
 - В промышленности повсеместно заменён оптимизированными алгоритмами: **C4, C4.5, and C5**

Алгоритм ID3

- Использует **набор примеров наблюдение-действие**
 - Наблюдения в ID3 обычно называются «атрибуты»
- Алгоритм начинает с **одного листа** в дереве решений и назначает набор примеров этому узлу
 - Затем он **разбивает текущий узел** (в начале – единственный узел) таким образом, чтобы примеры разделились на **две группы**
 - Деление выполняется на основании **выбранного атрибута**, который выбирается так, чтобы получить наиболее эффективное дерево
 - Когда деление выполнено, **каждый из двух узлов** получает **подмножество примеров**, которые к нему применимы, затем алгоритм повторяется для каждого из узлов
- **Алгоритм рекурсивный**
 - Начиная с одного узла заменяет его решением пока всё дерево не будет создано
 - В каждой ветви набор примеров делится между потомками пока все примеры не будут соответствовать одному действию

Несколько слов о физике

Законы термодинамики

Закон сохранения и превращения энергии: энергия закрытой системы при любых процессах, происходящих в системе, остается неизменной

1. Увеличение внутренней энергии закрытой системы равно общему количеству энергии, добавленной в систему
2. Общая энтропия изолированной системы не может уменьшаться со временем
3. Энтропия системы приближается к постоянному значению, когда её температура приближается к абсолютному нулю

Энтропия и information gain (1)¹⁴

- Энтропия – мера количества информации в наборе примеров
 - Если все примеры имеют одинаковые действия, то энтропия равна 0
 - Если действия в примерах распределены равномерно, то энтропия равна 1
- Information gain – простое сокращение общей энтропии
- Пример
 - Два возможных действия: attack и defend
 - Три атрибута: health, cover и ammo
 - Для простоты предположим, что можно рассматривать атрибуты как булевские
 - Здоров или ранен, под прикрытием или обнаружен, с патронами или «пустой»
- Энтропия
 - $E = -p_A \log_2 p_A - p_D \log_2 p_D = 0.971$
 - p_A - пропорция действия attack
 - p_D - пропорция действия defend

Healthy	In cover	With ammo	Attack
Hurt	In cover	With ammo	Attack
Healthy	In cover	Empty	Defend
Hurt	In cover	Empty	Defend
Hurt	Exposed	With ammo	Defend

Энтропия и information gain (2)¹⁵

- Посчитаем энтропию, связанную с каждым делением

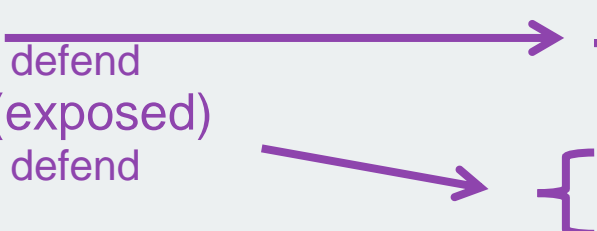
- Information gain – деление энтропии на энтропию дочерних узлов (наборов)

	Entropy for “true”	Entropy for “false”
Health	$-\frac{1}{2} \log_2(\frac{1}{2}) - \frac{1}{2} \log_2(\frac{1}{2}) = 1$	$-\frac{1}{3} \log_2(\frac{1}{3}) - \frac{2}{3} \log_2(\frac{2}{3}) = 0.918$
Cover	$-\frac{2}{4} \log_2(\frac{2}{4}) - \frac{2}{4} \log_2(\frac{2}{4}) = 1$	$-0 \log_2(0) - \frac{1}{1} \log_2(\frac{1}{1}) = 0$
Ammo	$-\frac{1}{3} \log_2(\frac{1}{3}) - \frac{2}{3} \log_2(\frac{2}{3}) = 0.918$	$-0 \log_2(0) - \frac{2}{2} \log_2(\frac{2}{2}) = 0$

- $G = E_s - p_T E_T - p_F E_F$
- здесь E_s - энтропия действия
 - Вычислена на предыдущем слайде (значение 0.971)
- p_T - пропорция примеров для которых атрибут истинен
- E_T - энтропия этих примеров

- Например, вычисление для cover

- cover = true
 - 2 attacks, 2 defend
- cover = false (exposed)
 - 0 attacks, 1 defend

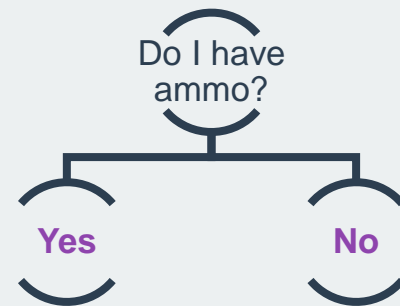


Healthy	In cover	With ammo	Attack
Hurt	In cover	With ammo	Attack
Healthy	In cover	Empty	Defend
Hurt	In cover	Empty	Defend
Hurt	Exposed	With ammo	Defend

Энтропия и information gain (3)¹⁶

	Entropy for "true"	Entropy for "false"	Information gain
Health	1	0.918	$G_{health} = 0.971 - \frac{2}{5} * 1 - \frac{3}{5} * 0.918 = 0,02$
Cover	1	0	$G_{cover} = 0.971 - \frac{4}{5} * 1 - \frac{1}{5} * 0 = 0,171$
Ammo	0.918	0	$G_{ammo} = 0.971 - \frac{3}{5} * 0.918 - \frac{2}{5} * 0 = \mathbf{0,42}$

- Т.о. для данных трёх атрибутов **ammo** будет лучшим индикатором для действия
 - Имеет смысл, т.к. мы не можем атаковать без ammo



Healthy	In cover	With ammo	Attack
Hurt	In cover	With ammo	Attack
Healthy	In cover	Empty	Defend
Hurt	In cover	Empty	Defend
Hurt	Exposed	With ammo	Defend

Другие случаи

- **Более двух действий**

- То же самое работает, если действий больше двух
- $E = - \sum_{i=1..n} p_i \log_2 p_i$
 - здесь n – количество действий, p_i - пропорция каждого действия в наборе

- **Не бинарные дискретные атрибуты**

- Если есть более, чем две категории, тогда будет больше двух дочерних узлов для решения
- $G = E_s - \sum_{i=1..n} |S_i|/|S| * E_{S_i}$
 - здесь S_i - набор примеров для каждого из n значений атрибута
- К сожалению, как видели ранее, гибкость получения более двух ветвей на решение – не очень полезно

функция Обучение_деревьев_решений (examples, attribs, parent_examples)

если examples==пусто **то вернуть** *default*(parent_examples)
иначе если все примеры имеют одну и ту же классификацию
то вернуть классификация
иначе если attribs==пусто **то вернуть** *default*(examples)
иначе

best \leftarrow **выбрать_атрибут**(attribs, examples) // по information gain
tree \leftarrow новое дерево решений с результатом best проверки
корневого узла

для каждого значения v_i результата best

exs \leftarrow {элементы множества examples со значениями best= v_i }
subtree \leftarrow Обучение_деревьев_решений(exs, attribs-best, examples)
добавить к tree ветвь с меткой best= v_i и поддеревом subtree

вернуть tree

После формирования дерева необходимо исключить узлы (превратить в листья) те, что не несут достаточно информации (соответствуют «null-гипотезе»). Обычно за основу берут χ^2 -отсечение.

Обучение – общий случай

Выбор модели и оптимизация

- **Предположение стационарности** для примеров E_j
 - $P(E_j) = P(E_{j+1}) = P(E_{j+2}) = \dots$
- **Предположение независимости**
 - $P(E_j) = P(E_j | E_{j-1}, E_{j-2}, \dots)$
- **Оптимальное соответствие должно минимизировать коэффициент ошибки, когда $h(x) \neq y$**
 - **Эмпирическая** оценка ошибки (L – **функция потерь**)
 - $\text{EmpLoss}_{L,E}(h) = \sum_{(x,y) \in E} L(y, h(x)) / N$
 - $h^* = \operatorname{argmin}_{h \in H} \text{EmpLoss}_{L,E}(h)$
 - С использованием **регуляризации**
 - $\text{Cost}(h) = \text{EmpLoss}(h) + \lambda * \text{Complexity}(h)$
 - $h^* = \operatorname{argmin}_{h \in H} \text{Cost}(h)$
- **Выделяют наборы данных**
 - training set
 - validation set (development set)
 - test set
- **Методы подбора**
 - k-fold cross-validation
 - leave-one-out cross-validation (LOOCV)

- **Линейная регрессия**

- $h_w(x) = w_1 x + w_0$
- $w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}$
- $w_0 = (\sum y_j - w_1(\sum x_j)) / N$

- **Градиентный спуск**

$w \leftarrow$ любая точка в пространстве поиска

пока не сошлись

для всех w_i в w

$$w_i \leftarrow w_i - \alpha * \partial \text{Loss}(w) / \partial w_i$$

- **Мультивариативная линейная регрессия**

- $h_w(x_j) = w * x_j = w^T * x_j = \sum w_i * x_{j,i}$

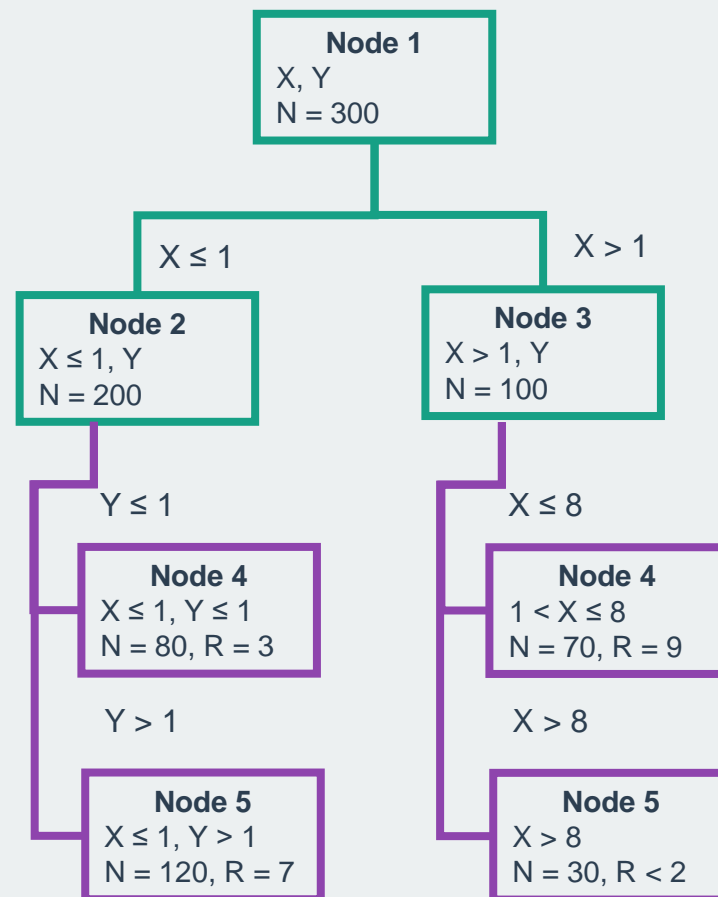
Регрессионные деревья

- Регрессионные деревья

- Как результат – значение целевой функции
- Предсказанный результат может быть вещественным числом

- Деревья классификации

- Каждый узел представляет собой кластер объектов (cases), который разбивается на подветви
- Предсказанный результат – класс



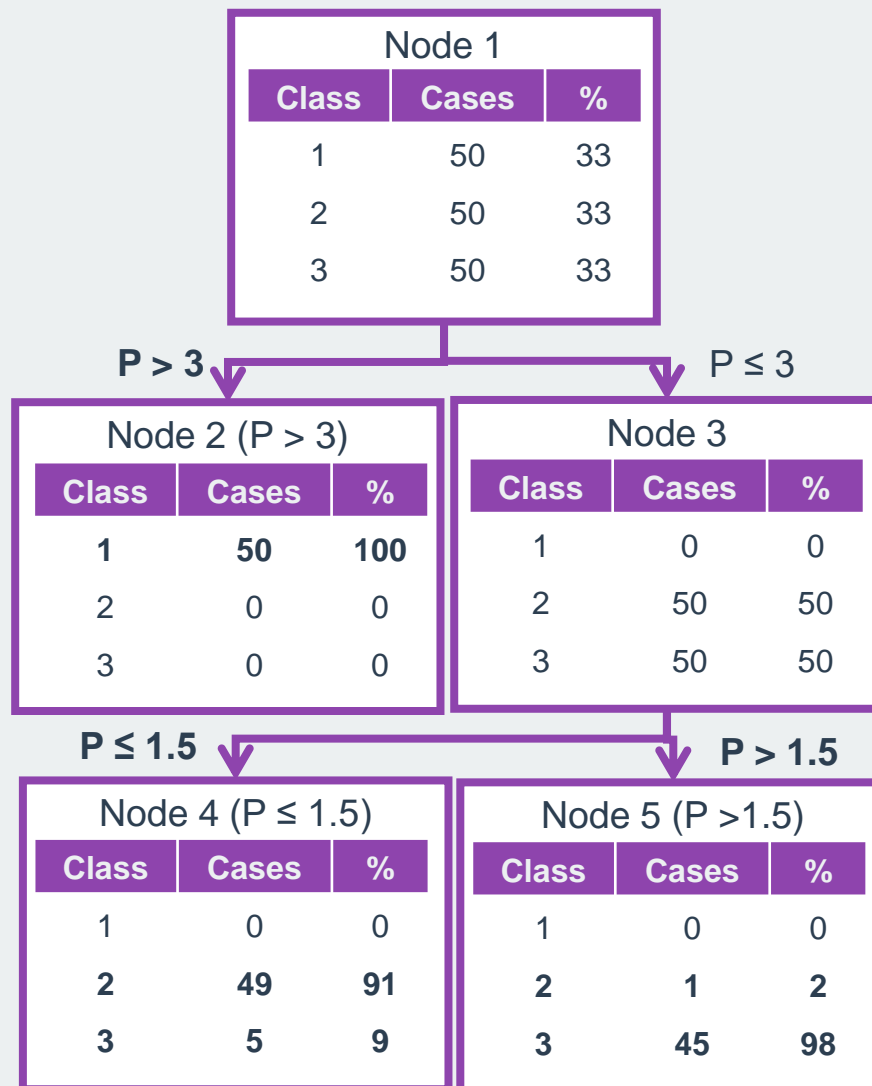
Деревья классификации

- Регрессионные деревья

- Как результат – значение целевой функции
- Предсказанный результат может быть вещественным числом

- Деревья классификации

- Каждый узел представляет собой кластер объектов (cases), который разбивается на подветви
- Предсказанный результат – класс



Пример данных для дерева решений²³

Предсказание урона на основе выбора оружия

Вес	Rate	Ёмкость	Расстояние	Тип	Повреждения
Light	47	10	40 m	Handgun	5%
Heavy	200	500	100 m	Automatic weapon	10%
Very easy	6	6	25 m	Handgun	4%
Very heavy	280	1000	200 m	Automatic weapon	13%

Аммо?

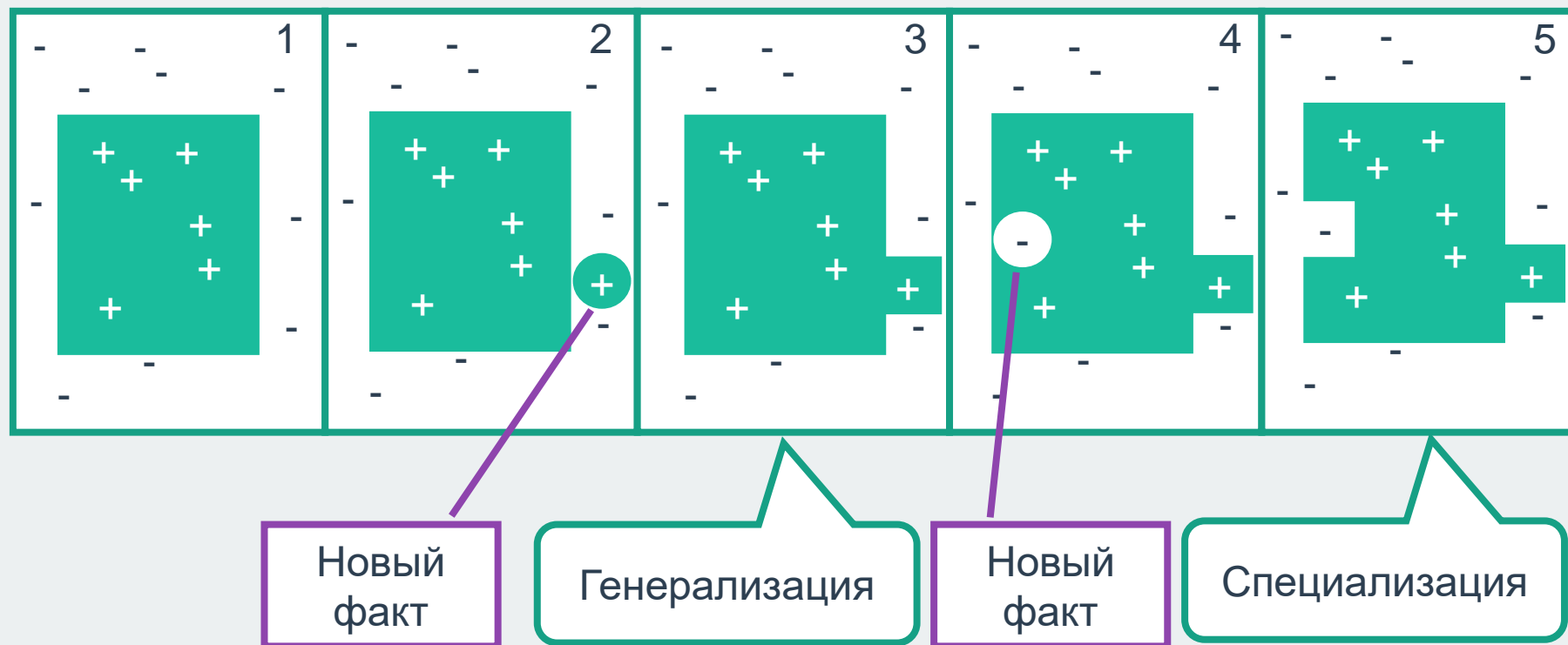
Методы построения дерева

- 1. Classification and regression trees (CART)**
- 2. Random forest**
- 3. Stochastic Gradient Boosting**

Непараметрические методы

- **Модели ближайшего соседа**
 - Расстояние Минковского
 - $L_p(x_j, x_q) = (\sum |x_{j,i} - x_{q,i}|^p)^{1/p}$
 - Расстояния Махаланобиса, Хэмминга
- Поиск ближайших соседей с использованием **k-d деревьев**
- **Хэширование с учетом местоположения (locality-sensitive hash – LSH)**
- **Непараметрическая регрессия**
 - k-nearest-neighbors regression
 - locally weighted regression
 - Использование «ядер»
- **Метод опорных векторов (SVM)**
- **Обучение ансамблей**
- **Bagging**
- **Случайный лес**
- **Stacking**
- **Boosting (например, AdaBoost), gradient boosting**

Генерализация/специализация гипотез



Current-best-hypothesis

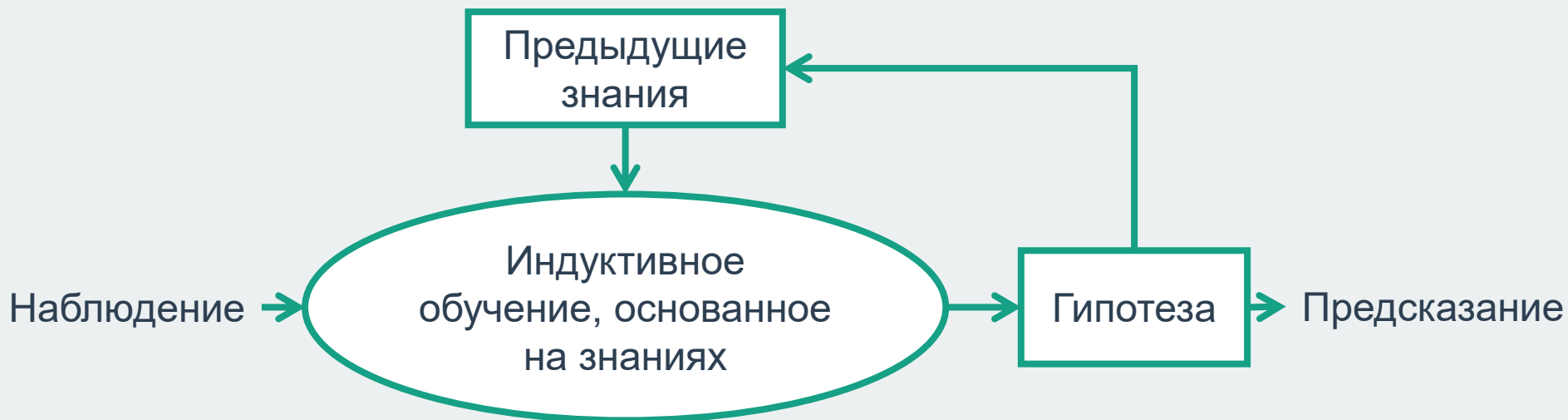
1. поиск одной гипотезы, соответствующей всем фактам
2. поддержание одной гипотезы в условиях, когда появляются новые факты

Обучение и поиск лучшей гипотезы

26

функция CURRENT-BEST-LEARNING(examples, h)
если examples пусто **то вернуть** h
e ← первый(examples)
если e **совместим** с h **то вернуть**
CURRENT-BEST-LEARNING(остаток(examples), h)
иначе если e **ложно позитивно** к h **то**
для каждого h' в **специализации** h совместимой с examples
h'' ← CURRENT-BEST-LEARNING(остаток(examples), h')
если h'' != fail **то вернуть** h''
else if e **ложно негативно** к h **то**
для каждого h' в **генерализации** h совместимой с examples
h'' ← CURRENT-BEST-LEARNING(остаток(examples), h')
если h'' != fail **то вернуть** h''
вернуть fail

Процесс обучения



Общая схема:

Hypothesis \wedge Descriptions \rightarrow Classifications

Background \rightarrow Hypothesis

Explanation-based learning (EBL):

- предполагался как инструмент обучения на примерах, но
- использует предыдущие знания и должен уметь объяснять гипотезу, а гипотеза должна объяснять наблюдение!

Какой из этого следует вывод?

Новая схема (relevance-based learning, RBL):

Hypothesis \wedge Descriptions \rightarrow Classifications

Background \wedge Descriptions \wedge Classifications \rightarrow Hypothesis

EBL, RBL



Подходы к обучению

- **Explanation-based learning (EBL)**
 - достаёт общие правила из примеров, которые объясняет и обобщает
 - предлагает **дедуктивный метод**
- **Relevance-based learning (RBL)**
 - **использует предыдущие знания** в форме «определений», чтобы уменьшить пространство гипотез и ускорить обучение
- **Knowledge-based inductive learning (KBIL)**
 - **индуктивный поиск** гипотез для объяснения наблюдаемых фактов на основе базы знаний
- **Inductive logic programming (ILP)**
 - использует KBIL по отношению к знаниям, представленным в виде описаний **логики первого порядка**
 - может работать как снизу вверх, так и сверху вниз
 - создаёт новые предикаты, соответствующие теории

Байесовское обучение

- $D = \{d\}$ – наблюдаемые данные
- $P(h_i)$ – априорные вероятности при гипотезе h_i
- $P(d | h_i)$ – вероятность наблюдения d при гипотезе h_i
- тогда
 - **вероятность гипотезы**

$$P(h_i | d) = \alpha P(d|h_i)P(h_i)$$
 - **предсказание неизвестного значения X**

$$P(X | d) = \sum_{h_i} P(X | h_i)P(h_i | d)$$
- Пример
 - два вида конфет помещены в одинаковую упаковку
 - h_i – гипотезы по вероятностям каждого вида
 пусть мы знаем, что их ровно пять и ровно одна верна:
 (100%, 0%), (75%, 25%), (50%, 50%), (25%, 75%), (0%, 100%)
 - $P(h_1), \dots, P(h_5) = \langle 0.1, 0.2, 0.4, 0.2, 0.1 \rangle$
 - тогда вероятность наблюдения $P(d | h_i) = \prod_j P(d_j | h_i)$
 - пусть верна h_5 и, соответственно, первые 10 конфет соответствуют 2-му виду, тогда
 - $P(d|h_3) = 0.5^{10}$

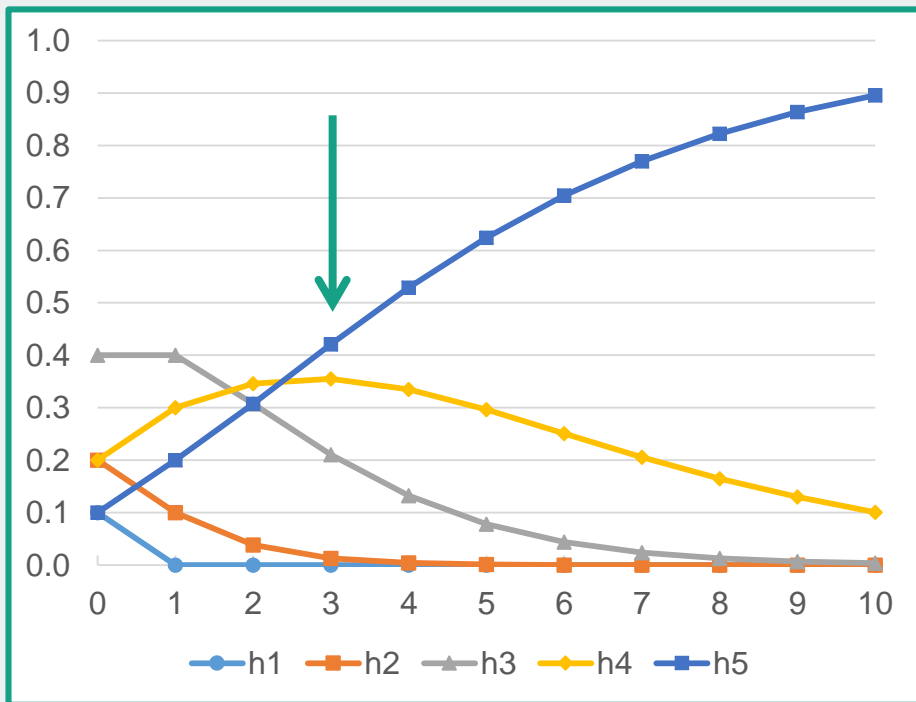
Продолжение



Байесовское обучение

Пример с конфетами

		h_1	h_2	h_3	h_4	h_5	
	$P(d_2 h_i)$	0	0.25	0.5	0.75	1	α
0	$P(h_i)$	0.100	0.200	0.400	0.200	0.100	1.000
1	$P(h_i d_2)$	0.000	0.100	0.400	0.300	0.200	2.000
2	$P(h_i d_2)$	0.000	0.038	0.308	0.346	0.308	3.077
3	$P(h_i d_2)$	0.000	0.013	0.211	0.355	0.421	4.211
4	$P(h_i d_2)$	0.000	0.004	0.132	0.335	0.529	5.289
5	$P(h_i d_2)$	0.000	0.001	0.078	0.296	0.624	6.244
6	$P(h_i d_2)$	0.000	0.000	0.044	0.251	0.705	7.047
7	$P(h_i d_2)$	0.000	0.000	0.024	0.206	0.770	7.702
8	$P(h_i d_2)$	0.000	0.000	0.013	0.165	0.822	8.224
9	$P(h_i d_2)$	0.000	0.000	0.007	0.130	0.864	8.636
10	$P(h_i d_2)$	0.000	0.000	0.003	0.101	0.896	8.956



Для **больших пространств** суммирование $P(X | d)$ (интегрирование в непрерывном случае) – **не представляется возможным**.

Упрощение – использование наиболее вероятной гипотезы (maximum a posteriori, MAP): $P(X | d) \approx P(X | h_{\text{MAP}})$.

В случае с конфетами **после 3 примера** $h_{\text{MAP}} = h_5$.

Обучение по методу наибольшего правдоподобия (дискретная модель)

1. Запишите **выражение для вероятности** данных в зависимости от параметра (параметров)
2. Запишите **производную логарифмической вероятности** по каждому параметру
3. Найдите такие значения параметров, чтобы **производные были равны нулю**

• Пример

- **два вида** конфет с неизвестной вероятностью распределения – бесконечное количество гипотез
- пусть **пропорция** конфет θ первого типа (гипотеза h_θ) и $1-\theta$ второго типа
- пусть из **N попыток**: c – первого типа, $\ell = N - c$ – второго
- тогда **вероятность** $P(d|h_\theta) = \prod_{j=1..N} P(d_j | h_\theta) = \theta^c \cdot (1-\theta)^\ell$
- **логарифм вероятности**

$$L(d|h_\theta) = \log P(d|h_\theta) = \sum_j \log P(d_j | h_\theta) = c \log \theta + \ell \log (1-\theta)$$
- **производная по θ**

$$dL(d|h_\theta) / d\theta = c / \theta - \ell / (1-\theta) = 0 \rightarrow \theta = c / (c + \ell) = c / N$$
- гипотеза максимального правдоподобия, что **количество конфет первого типа = c / N** – логично, но данный подход работает и в более сложных случаях

Обучение по методу наибольшего правдоподобия (непрерывная модель)

- Последовательность шагов та же самая
- Предположим переменную с нормальным законом распределения
- $P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- Наблюдаем значения x_1, \dots, x_N
- $L = N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^N \frac{(x_j - \mu)^2}{2\sigma^2}$
- Частные производные
- $\frac{\partial L}{\partial \mu} = -\frac{1}{\sigma^2} \sum_{j=1}^N (x_j - \mu) \Rightarrow \mu = \frac{\sum x_j}{N}$
- $\frac{\partial L}{\partial \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^N (x_j - \mu)^2 \Rightarrow \sigma = \sqrt{\frac{\sum (x_j - \mu)^2}{N}}$
- Результат логичен, но **данный подход работает и в более сложных случаях**

Обучение со скрытыми переменными (кластеризация: линейная смесь гауссиан)

- **Не контролируемое обучение**
- Дано N точек, они принадлежат к кластерам
 - $P(x) = \sum_{i=1..k} P(C=i) P(x|C=i)$
 - C – компонент с номером $i=1..k$
- Если бы было известно
 - **какая точка какой компонентой «порождена»**, мы бы могли **восстановить гауссианы** каждой из компонент
 - **параметры распределения** каждой компоненты, то мы бы могли **восстановить вероятность отнесения** каждой точки к компонентам
- **Идея ЕМ-алгоритма**
 - притвориться, что **мы знаем параметры модели**
 - **оценить вероятность принадлежности** точки к компонентам
 - **выполнить перепривязку** точек к компонентам

ЕМ-алгоритм – обучение без учителя (на примере линейной смеси гауссиан)

- Сначала **предполагаем, что мы знаем** параметры модели (μ_i , Σ_i , w_i), **затем** последовательно **повторяем** шаги: ЕМЕМЕМ...
- **Е-шаг (expectation step)**
 - вычислить вероятности $p_{ij} = P(C=i | x_j)$
 - по правилу Байеса $p_{ij} = \alpha P(x_j | C=i) P(C=i)$
 - $P(x_j | C=i)$ – вероятность того что x_j принадлежит i
 - $P(C=i)$ – вес w_i
 - определим «эффективный» номер точки, привязанной к компоненту i :
 $n_i = \sum p_{ij}$
 - Е-шаг может рассматриваться как вычисление ожидаемого значения p_{ij} скрытой переменной Z_{ij}
 - $Z_{ij} = 1$, если x_j относится к i , иначе $Z_{ij} = 0$
- **М-шаг (maximization step)**
 - вычислить новые
 - $\mu_i \leftarrow \sum p_{ij} x_j / n_i$
 - $\Sigma_i \leftarrow \sum p_{ij} (x_j - \mu_i) (x_j - \mu_i)^T / n_i$
 - $w_i \leftarrow n_i / N$
 - здесь N – общее количество точек с данными
 - на М-шаге **максимизируется логарифм** вероятности **принадлежности** данных с учётом значения скрытых переменных

Искусственные нейронные сети

35

- Нейронные сети состоят из **узлов**, соединенных **направленными связями**
- **Дуга** из узла i к узлу j служит для **распространения** a_i активации i к j
- Каждая дуга также имеет числовой вес $w_{i,j}$, связанный с ней, который определяет силу и знак
- Каждый узел j сначала вычисляет **взвешенную сумму** своих входных данных

$$\bullet \quad in_j = \sum_{i=0..n} w_{i,j} a_i$$

- Затем он применяет **функцию активации** g к этой сумме, чтобы вывести результат

$$\bullet \quad a_j = f(in_j) = f(\sum_{i=0..n} w_{i,j} a_i)$$

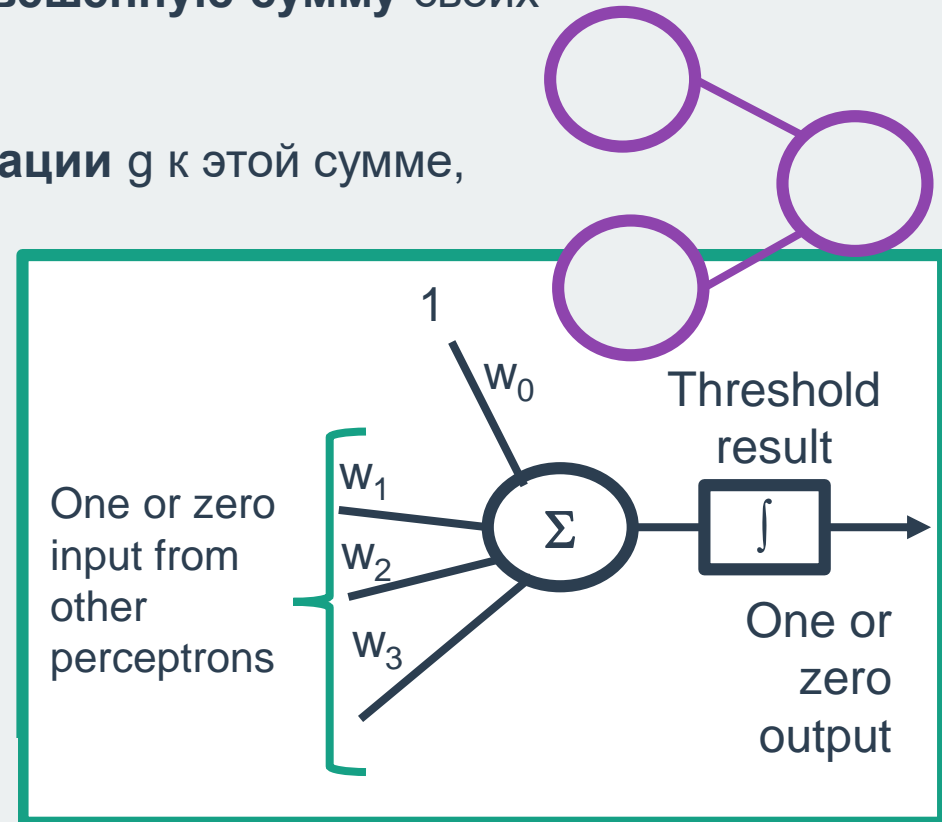
- Функция активации g обычно является либо **жестким порогом**, в этом случае узел называется **персептроном**, либо логистической функцией, в этом случае иногда используется термин **сигмоидный персептрон**

- Существует **огромное количество вариантов** $f(z)$

$$\bullet \quad f(z) = z$$

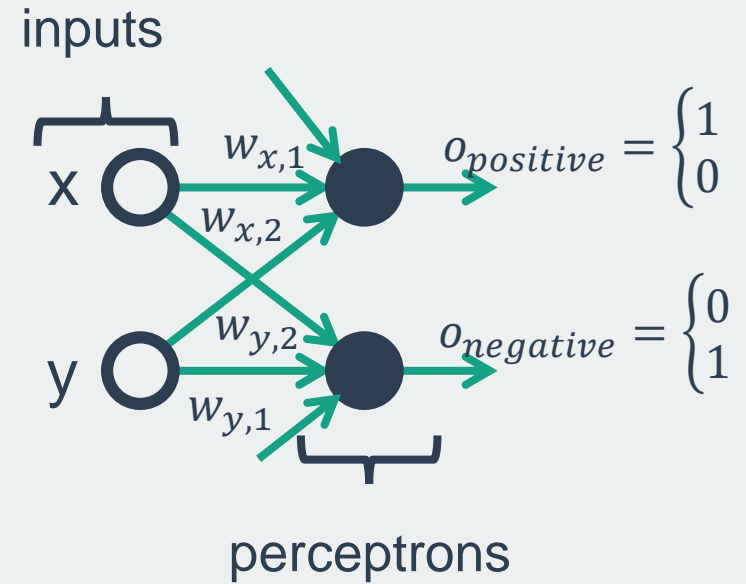
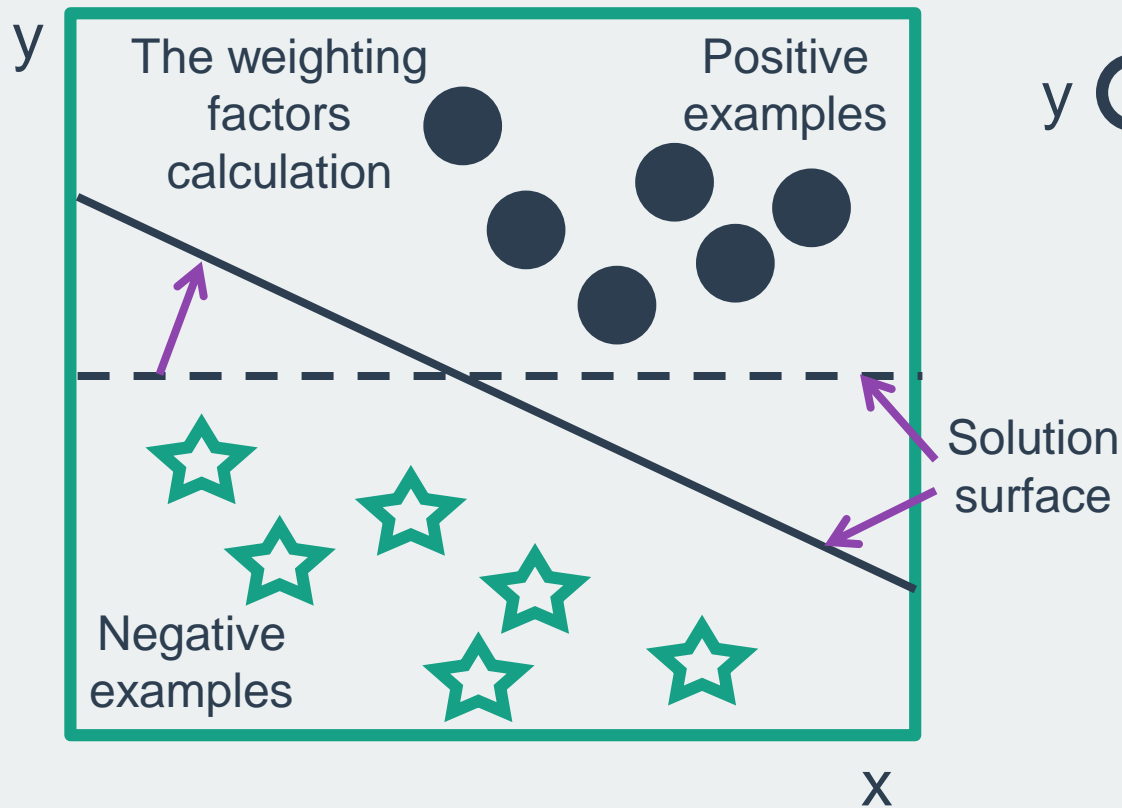
$$\bullet \quad f(z) = \frac{1}{1+e^{-\alpha z}}$$

sigmoid



Графическая интерпретация

36



Обучение персептрона

37

- Общее дельта-правило
 - Каждый весовой фактор **вносит свой вклад** в формирование выходной величины
 - **Важность этого вклада** зависит от входных значений
 - **Рассогласование** между фактическим и желаемым значениями выходного сигнала может быть устранено путем корректировки весовых коэффициентов
- Могут использоваться различные функции ошибки
 - Например $\delta = \frac{1}{2} (t - a)^2$, где t – ожидаемое значение, a – полученное значение
 - На базисе наименьших среднеквадратичных (least mean square – LMS)
- При обучении должны быть скорректированы все весовые коэффициенты (w_i)
- Существует **два различных подхода** к обучению
 - **Каждый пример используется отдельно** от других и обучение идет последовательно
 - Весовые коэффициенты должны обновляться после каждого изученного примера
 - **Весь набор примеров используется в виде пакета**
 - Весовые коэффициенты следует обновлять только после того, как весь набор примеров усвоен

Обучение персептрона

Последовательное обучение

- инициализировать веса случайным образом
- пока целевая функция не удовлетворена
 - для каждого примера
 - активировать персептрон
 - если результат не корректен
 - для всех входов i
 - $\text{delta} = \text{desired} - \text{output}$
 - $\text{weights}[i] += \text{learning_rate} * \text{delta} * \text{inputs}[i]$

Обучение персептрона

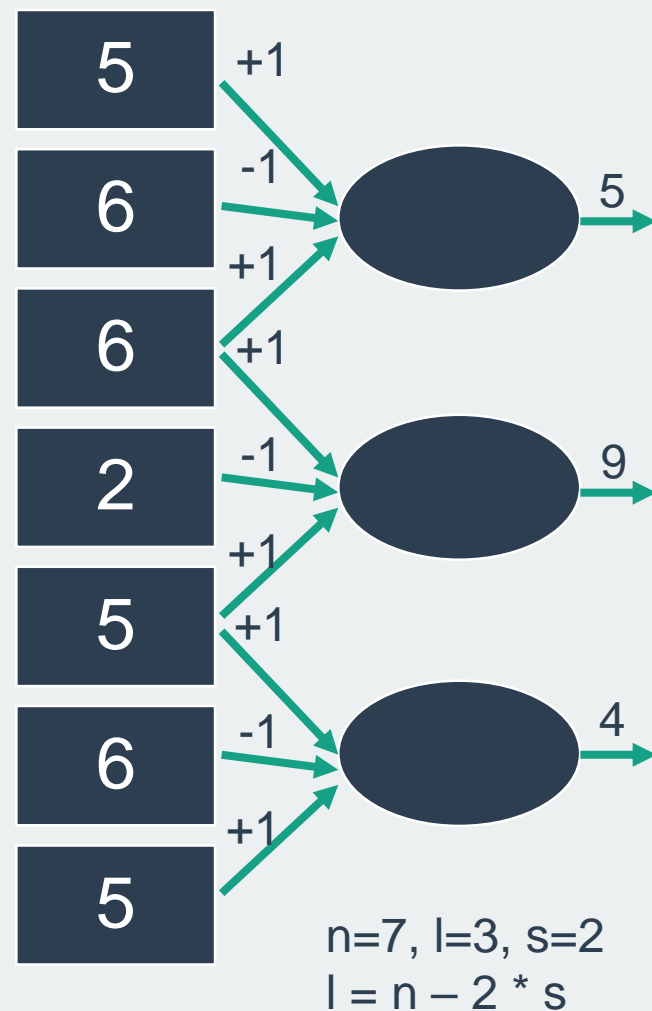
Пакетное обучение

- инициализировать веса случайным образом
- пока условие прекращения не корректно
 - сброс счётчика блока до 0
 - для каждого примера
 - моделирование персептрона
 - для каждого веса i
 - $\text{delta} = \text{desired} - \text{output}$
 - $\text{steps}[i] += \text{delta} * \text{inputs}[i]$
 - для каждого веса i
 - $\text{weights}[i] += \text{learning_rate} * \text{steps}[i]$

Сверточные ANN (CNN)

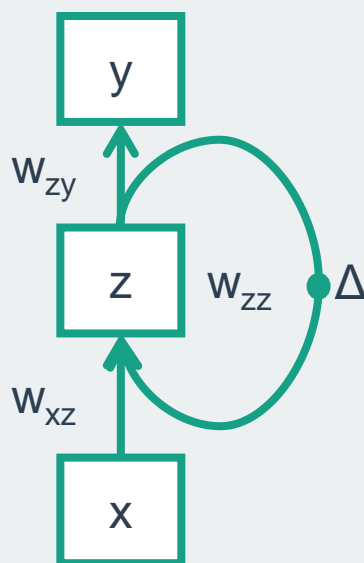
- На входе свёртки
 - вектор x длины n
 - вектор ядра k длины l
 - и «шаг» (отступ от «края») s
- На выходе – свёртка $z = x * k$
 - $z_i = \sum k_j x_{j+i-(l+1)/2}, j=1..l$
 - $k = [+1, -1, +1]$
- В виде матриц

$$\begin{pmatrix} 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 6 \\ 2 \\ 5 \\ 6 \\ 5 \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \\ 4 \end{pmatrix}$$

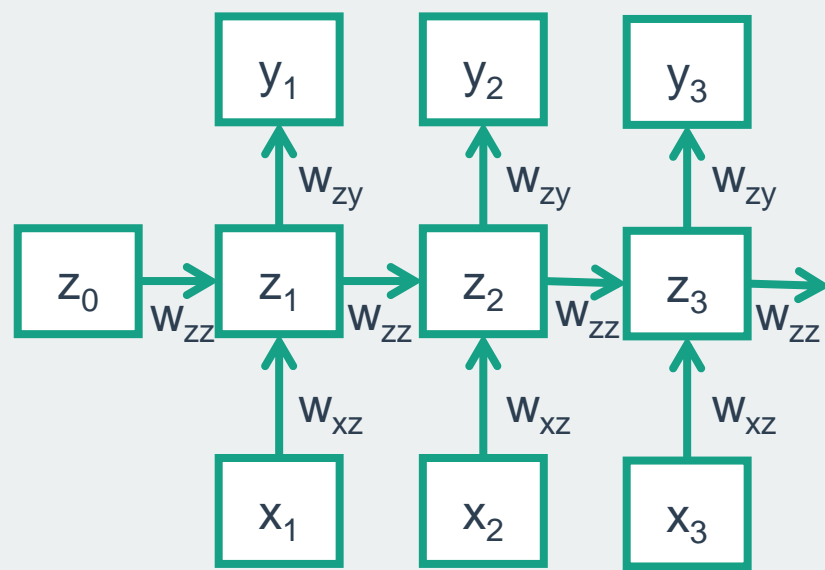


Рекуррентные ANN (RNN)

- Имеют **внутреннее состояние** (память)
- **Скрытый слой z** получает информацию от всех предыдущих слоев
 - $z_t = f_w(z_{t-1}, x_t) = g_z(W_{z,z}z_{t-1} + W_{x,z}x_t)$
 - $y_t = g_y(W_{z,y}z_t)$
 - здесь g – функции активации
- Long short-term memory RNNs (**LSTM**)
 - forget gate f
 - input gate I
 - output gate o



Пример RNN со скрытым слоем z



Пример RNN, развернутый на 3 шага

Reinforcement learning – обучение с подкреплением

- Обучение с подкреплением – название, данное целому **ряду методов обучения**, основанных на опыте
- Решаемая проблема
 - Необходимо, чтобы персонаж с течением времени **выбирал действия** всё более **соответствующие ситуации**
 - Необходимо иметь возможность предоставить персонажу **свободный выбор любого действия** в любых обстоятельствах и ему **самому выработать**, какие **действия** лучше всего **подходят** для каждой ситуации
 - **Качество действия** обычно **не ясно** в момент его совершения
 - Персонаж должен усвоить, что **все действия, ведущие к событию**, также являются **правильными**, даже если во время их выполнения не было никакой обратной связи

Обучение с подкреплением

Reinforcement learning (RL)

- Основывается на вознаграждении
 - Например, в шахматах: $\langle 1, \frac{1}{2}, 0 \rangle == \langle \text{победа, ничья, проигрыш} \rangle$
 - Что делать с промежуточными состояниями?
- RL на основе модели
 - Обычно изучает/использует модель перехода и изучает функцию полезности состояния, $u(s)$
- RL без модели
 - Изучает функцию полезности действия
 - Q-learning
 - Q-функция: $Q(s,a)$ – сумма вознаграждения при выполнении a из состояния s
 - Поиск политики
 - Изучает $\pi(s)$, которая определяет действия в состоянии
- В случае пассивного RL
 - Многократно из начального состояния реализуется политика π вплоть до финального состояния, затем пересчитывается U

Напоминаю уравнение Беллмана:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' | s, a) * [R(s, a, s') + \gamma * U(s')]$$

Пассивное RL / Адаптивное динамическое программирование

функция PASSIVE-ADP-LEARNER(percept)

// percept – восприятие с текущим состоянием s' и сигналом вознаграждения r

// π – фиксированная политика (при этом уравнения Беллмана линейные), mdp – MDP с моделью P , вознаграждением R , действиями A , дисконтированием γ ,

U – таблица функций полезности для s (сначала пустая),

$N_{s'|s,a}$ – таблица векторов подсчета, индексированная по s и a (сначала пустая), s , a – предыдущие состояние и действие (сначала пустые)

если s' новое **то** $U[s'] \leftarrow 0$

если $s \neq \text{null}$ **то**

увеличить $N_{s'|s,a}[s,a][s']$ // как часто попадаем в s' из s и a

$R[s, a, s'] \leftarrow r$ // запомнить вознаграждение

добавить a к $A[s]$ // сохранить выполненное действие

$P(\cdot | s,a) \leftarrow \text{нормализовать}(N_{s'|s,a}[s,a][s'])$ // посчитать вероятности

$U \leftarrow \text{оценка_политики}(\pi, U, mdp)$ // возможны варианты...

$s,a \leftarrow s', \pi[s']$ // подготовка к следующему вызову

вернуть a

- Скорость обучения стандартная по отношению к остальным RL-алгоритмам и ограничена возможностью изучить модель переходов
- Не применим для больших пространств (например, для нарда будет 10^{20} уравнений и 10^{20} неизвестных)

Пассивное RL / Обучение с временными различиями (TD)

функция PASSIVE-TD-LEARNER(percept)

// percept – восприятие с текущим состоянием s' и сигналом вознаграждения r

// π – фиксированная политика, s – предыдущее состояние (сначала пустое), U – таблица функций полезности для s (сначала пустая), N_s – таблица частот для состояний (сначала пустая)

если s' новое **то** $U[s'] \leftarrow 0$

если $s \neq \text{null}$ **то**

увеличить $N_s[s]$

// пошаговая функция $\alpha(n)$ – коэффициент обучения

$U[s] \leftarrow U[s] + \alpha(N_s[s]) * (r + \gamma U[s'] - U[s])$

$s \leftarrow s'$

вернуть $\pi[s']$ // возвращаем действие

- Называется **Temporal difference (TD)**, потому что использует **разность** между «успешными» состояниями
- TD не нуждается в модели переходов для обновления
- **Отличия TD и ADP**
 - TD обновляет состояние **для соответствия наблюдаемому предшественнику**
 - ADP – **для соответствия всем предшественникам**

Активное RL / TD Q-learning

46

функция Q-LEARNING-AGENT(percept)

// percept – восприятие с текущим состоянием s' и сигналом вознаграждения r

// Q – таблица действий, проиндексированная по состояниям и действиям (сначала пустая), N_{sa} – таблица частот для пар state–action (сначала пустая), s, a – предыдущие состояние и действие (сначала null)

если $s \neq \text{null}$ **то**

увеличить $N_{sa}[s,a]$

// пошаговая функция $\alpha(n)$ – коэффициент обучения

$Q[s,a] \leftarrow Q[s,a] + \alpha * (N_{sa}[s,a]) * (r + \gamma * \max_{a'} Q[s',a'] - Q[s,a])$

$s, a \leftarrow s', \text{argmax}_{a'} f(Q[s',a'], N_{sa}[s',a'])$

вернуть a

- **Q-learning** – обучение без политики

- «Насколько полезно это действие в этом состоянии, если предположить, что я перестану использовать ту политику, которую я сейчас использую, и начну действовать в соответствии с политикой, которая выбирает наилучшее действие»?

- $f(u, n)$ – **функция исследования**, определяющая параметры жадного алгоритма (с какой вероятностью будет выбрано произвольное действие)

- например, $f(u, n) = R^+$, если $n < N$, иначе u , здесь R^+ - оптимистичная оценка лучшего вознаграждения

Представление Q-learning о мире ⁴⁷

- **Q-learning** интерпретирует игровой мир как **конечный автомат**
 - Q-learning-это **алгоритм без моделей**, потому что он не пытается построить модель того, как работает мир
 - Он просто рассматривает всё как состояния
- В любой момент времени алгоритм находится в некотором состоянии
 - Состояние должно **кодировать** все соответствующие **детали об окружающей** персонажа **среде и внутренних данных**
 - Q-обучения **не требуется понимание составляющих состояния**
 - Что касается алгоритма, то они могут быть просто целым числом: номером состояния
- Для каждого состояния алгоритм должен понимать **доступные ему действия**
- После того, как персонаж выполняет одно действие в текущем состоянии, функция подкрепления должна дать ему **обратную связь**
 - Обратная связь может быть **положительной или отрицательной** и часто равна нулю, если нет четкого указания на то, насколько хорошим было действие
 - После выполнения действия персонаж, скорее всего, перейдет в новое состояние
- Эти четыре элемента «начальное состояние», «предпринятое действие», «значение подкрепления» и «результатирующее состояние» называются **кортежем опыта**, часто записываемым как **<s, a, r, s'>**

Непосредственно обучение

- Алгоритм **сохраняет значение** для каждого состояния и действия, которое он **попробовал**
 - Набор информации качества (Q-значений) каждого возможного состояния и действия
- Правило **Q-learning**
 - $Q(s, a) = (1 - \alpha) * Q(s, a) + \alpha * (r + \gamma * \max(Q(s', a')))$
 - где α -**скорость обучения** из $[0, 1]$, а γ -**скорость дисконтирования** из $[0, 1]$, значение r - **новое подкрепление** из кортежа опыта
- Правило Q-learning объединяет два компонента, используя параметр скорости обучения для управления линейным смешением
- $\gamma * \max(Q(s', a'))$ рассматривает **все возможные действия**, которые можно было бы предпринять из этого состояния, и выбирает наибольшее соответствующее Q-значение
- **Параметр дисконтирования** контролирует насколько Q-значение текущего состояния и действия зависит от Q-значения состояния, в которое оно ведёт
- Q-значение - это **смесь между его текущим значением и новым значением**, которое сочетает в себе усиление действия и значения состояния, к которому привело действие
- Системы обучения с подкреплением также требуют **стратегии исследования, политики выбора действий**, которые следует предпринять в любом данном состоянии - часто просто называют политикой
 - Основная **стратегия** исследования Q-learning **частично случайна**
 - Большую часть времени алгоритм будет выбирать действие с наибольшим Q-значением из текущего состояния
- Если проблема всегда остается одной и той же, а вознаграждения последовательны, то Q-значения в конечном счете **сходятся**

Q-learning

Настройка параметров в алгоритме

- **Alpha**: скорость обучения
 - Ноль – алгоритм **не учится**
 - Единица – не придаёт **никакого значения предыдущему опыту**
 - 0.3 является разумным начальным предположением, хотя настройка необходима
 - Параметры скорости обучения во многих алгоритмах машинного обучения выигрывают от изменения с течением времени
- **Gamma**: ставка дисконтирования-определяет, насколько Q-значение действия зависит от Q-значения в состоянии (или состояниях), к которому оно приводит
 - Ноль – будет оцениваться каждое действие только с точки зрения **вознаграждения**, которое оно **непосредственно** обеспечивает
 - Единица – **вознаграждение** за текущее действие так же важно, как и **качество состояния**, к которому оно приводит
 - Более высокие значения благоприятствуют **более длительным последовательностям действий**, но требуют соответственно большего времени для обучения
 - Значение 0.75 является хорошим начальным значением
 - С этим значением действие с вознаграждением 1 внесет 0.05 к Q-значению действия на десять шагов раньше в последовательности действий

Q-learning

Настройка параметров функции f

- **Rho**: случайность для исследования – управляет тем, **как часто** алгоритм будет выполнять **случайное действие**, а не лучшее действие, которое он знает до сих пор
 - Ноль - чистая стратегия **эксплуатации**
 - Алгоритм будет использовать свое текущее обучение, усиливая то, что он уже знает
 - Единица – чистая стратегия **исследования**
 - Алгоритм всегда будет пробовать что-то новое, никогда не извлекая выгоду из имеющихся знаний
 - Online-обучение требует низкого значения (0.1 или меньше должно быть нормально)
 - Хорошей отправной точкой для этого параметра в offline обучении является 0.2
- **Nu**: количество шагов – управляет количеством итераций, которые будут выполняться в последовательности связанных действий
 - Ноль – алгоритм всегда использует **состояние**, достигнутое в **предыдущей итерации**, в качестве начального состояния для следующей итерации
 - Единица – каждая итерация начинается со **случайного состояния**
 - Если все состояния и все действия одинаково вероятны, то это оптимальная стратегия. Она охватывает максимально широкий диапазон состояний и действий в минимально возможное время.
 - Значения около 0.1 являются подходящими
 - Это приводит к последовательностям в среднем примерно из девяти действий подряд

Q-learning

Ограничения алгоритма

- Q-обучение требует, чтобы игра была представлена как **набор состояний, связанных действиями**
- Алгоритм очень **чувствителен** в своих требованиях к памяти, к количеству состояний и действий
 - Состояние игры обычно очень сложное
- Так же, как для алгоритмов поиска пути
 - Можно **разделить области** игрового уровня
 - Можно **квантовать значения** (здоровья, уровни боеприпасов и другие биты состояния), чтобы они могли быть представлены несколькими различными дискретными значениями
 - Можно представлять **гибкие действия** (например, движение в двух измерениях) с дискретными приближениями
- Игровое состояние состоит из комбинации всех этих элементов, порождая огромную проблему
 - Например 100 мест в игре, 20 персонажей, каждый с 4 возможными уровнями здоровья, 5 возможных видов оружия и 4 возможных уровня боеприпасов – это $(100 * 4 * 4 * 5)^{10}$ состояний (примерно 10^{50})
- Алгоритм **работает слишком долго**, чтобы опробовать каждое действие в каждом состоянии несколько раз.
- Подход ограничивается примерно **100 000 состояниями с 10 действиями** в каждом состоянии
 - Можно выполнить около 5 000 000 итераций алгоритма, чтобы получить работоспособные (но не отличные) результаты, и это может быть сделано в разумных временных масштабах (несколько минут) и с разумной памятью
- Онлайн-обучение, вероятно, должно быть ограничено задачами с менее чем 100 состояниями

Обучение с обратным подкреплением

Inverse reinforcement learning (IRL)

- **Наблюдение за действиями эксперта** (и результирующими состояниями) и попытки определить, **какую функцию вознаграждения эксперт максимизирует**, тогда можно разработать **оптимальную политику** в отношении этой функции
 - Предполагаем, что **эксперт рационален** и действует оптимально (или почти оптимально)
 - Предполагаем, что **эксперт не знает о наблюдателе**
- Простейший алгоритм IRL – **feature matching**
 - Предполагается, что функция вознаграждений записывается как сумма «особенностей»
 - $R_{\theta}(s,a,s') = \sum_i \theta_i f_i(s,a,s') = \theta * f$
 - Тогда функция ожидаемой полезности
 - $U^{\pi}(s) = \sum_i \theta_i \mu_i(\pi) = \theta * \mu(\pi)$
 - $\mu_i(\pi)$ – ожидаемое дисконтированное значение f_i при выполнении π
 - Например, f_i – превышение скорости, тогда μ_i – среднее превышение скорости по всей траектории

Вопросы для самопроверки

- В чем отличие контролируемого и неконтролируемого обучения?
- Что такое x , y , f , h ?
- Что такое деревья решений и как их обучают (ID3)? При чём тут энтропия и как её можно использовать?
- Как осуществляется оценка ошибки? При чем здесь регуляризация?
- Чем отличаются регрессионные деревья от деревьев классификации?
- О чем говорят, когда говорят о генерализации или специализации гипотез?
- Что такое байесовское обучение?
- Что такое обучение по методу наибольшего правдоподобия?
- Что такое ANN, CNN, RNN? Чем они отличаются? Какие варианты обучения ANN можете назвать?
- Какие бывают варианты обучения с подкреплением?