VOLGOGRAD
STATE
TECHNICAL
UNIVERSITY

Машинное обучение и нейросетевые модели

Лекция 7. Точный инференс дискретных переменных

Лектор: Кравченя Павел Дмитриевич

Волгоград 2025

VOLGOGRAD STATE TECHNICAL UNIVERSITY

План лекции

1. Подходы к точному инференсу дискретных латентных переменных.
2. Процедура устранения переменной.
3. Графы факторов и байесовские сети.
4. Сообщения между факторами и переменными.
5. Передача сообщений в общем виде.
6. Алгоритм суммирования-произведения. Пример.
7. Алгоритм Junction Tree.
8. Случай plate-графов. Алгоритм устранения тензорной переменной. Пример.
9. Особенности реализации алгоритма в Pyro.
10. Механика работы алгоритма устранения тензорной переменной в Pyro.

Подходы к точному инференсу дискретных латентных переменных

- В случае *дискретных латентных переменных* *интеграл* в формуле *апострериорного распределения* превращается в *сумму*, которую можно *точно посчитать.*
- Однако, если количество дискретных переменных и число значений, которые они могут принимать, *большое*, то вычисление этой суммы является *вычислительно сложным* и требующим *огромного объёма памяти*.
- Ситуация может *осложняться* наличием в модели как дискретных, так и непрерывных переменных.
- Часто вместо *непосредственного* суммирования функции по *всем значениям переменных* используют другой подход.

Устранение переменной (Variable elimination)

- Рассмотрим в качестве примера следующую *вероятностную модель*:

$$p(a, b, c, d) = p(a \mid b)p(b \mid c)p(c \mid d)p(d).$$

$$a \leftarrow b \leftarrow c \leftarrow d$$

- Рассчитаем *вероятность* $p(a = 0)$. Согласно *правилу суммирования*:

$$p(a = 0) = \sum_{b,c,d} p(a = 0, b, c, d) = \sum_{b,c,d} p(a = 0 \mid b)p(b \mid c)p(c \mid d)p(d).$$

- Можно вычислять эти суммы, *последовательно* складывая все слагаемые с конкретными значениями b, c, d. Однако, более эффективный способ – *вынести слагаемые* за знаки сумм там, где это возможно:

$$p(a = 0) = \sum_{b} p(a = 0 \mid b) \sum_{c} p(b \mid c) \underbrace{\sum_{d} p(c \mid d)p(d)}_{\gamma_d(c)}.$$

- Переменная $\gamma_d(c)$ называется <u>*потенциалом*</u>. Она содержит *всю информацию* о переменной d.
- Выполняя *аналогичные операции* по <u>другим переменным</u>, получаем:

$$p(a=0)=\sum_b p(a=0\mid b)\underbrace{\sum_c p(b\mid c)\,\gamma_d(c)}_{\gamma_c(b)}=\sum_b p(a=0\mid b)\,\gamma_c(b)$$

- Проведённая процедура называется **устранением переменной** (*variable elimination*), т.к. каждый раз, когда суммируются состояния переменной, она <u>исключается из распределения</u>.
- *Устранение переменной* можно рассматривать как <u>передачу сообщения *(информации)*</u> соседнему узлу графа.

- Часто вместо *байесовской сети* вероятностная модель представляется в виде *графа факторов*, что порой может оказаться удобнее для расчетов.
- Любое *совместное распределение* $p(\mathbf{X})$, $\mathbf{X} = \{x_1, x_2, \dots x_n\}$, может быть представлено как произведение факторов f_i, $f_i(x_i) > 0$:

$$p(\mathbf{X}) = \prod_i f_i(x_i)\,.$$

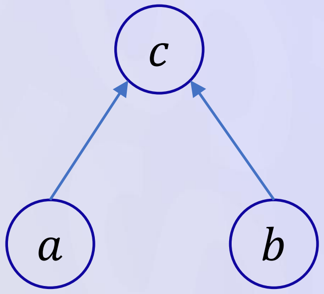
- При этом один фактор (*некоторая функция* переменных x_i) ставится в соответствие одной клике (*clique*): подмножеству переменных $\{x_1, x_2, \dots x_n\}$, в котором каждая переменная соединена со всеми другими.
- В *графе факторов* вершины, соответствующие факторам, обозначаются *квадратом*, а вершины, соответствующие переменным, – *кружком*.

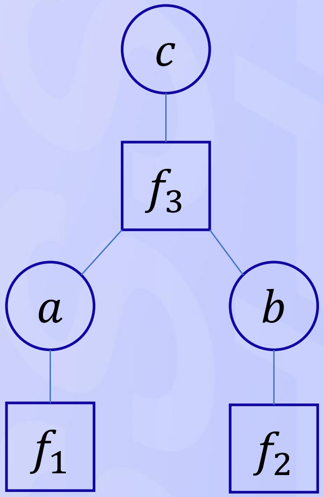
Графы факторов (Factor Graphs)

- Граф факторов является *двудольным графом*.
- Пример: вероятностная модель, представленная байесовской сетью и соответствующим ей графом факторов.

$$p(a, b, c) = p(a)p(b)p(c \mid a, b).$$

$$p(a, b, c) = f_1(a)f_2(b)f_3(a, b, c).$$

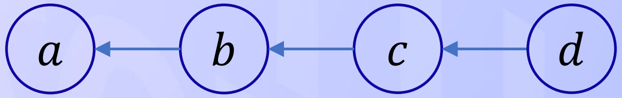




Сообщения между переменными (variable to variable messages)

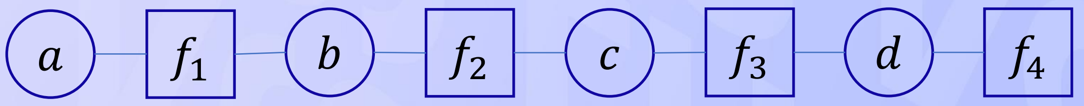
- Рассмотрим графическую модель, не содержащую ветвлений. Например:

$$p(a,b,c,d) = p(a \mid b)p(b \mid c)p(c \mid d)p(d).$$



- Представим её в виде *графа факторов*:

$$p(a,b,c,d) = f_1(a,b)f_2(b,c)f_3(c,d)f_4(d).$$

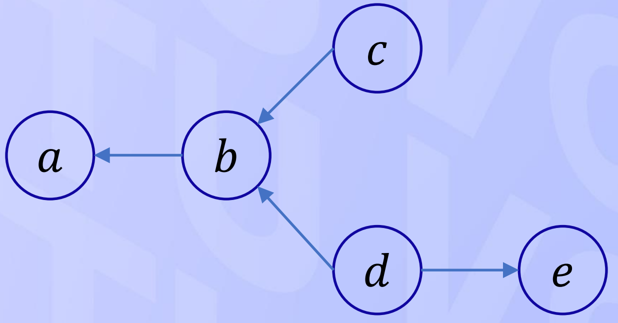


- Вычислим маргинальное распределение:

$$p(a,b,c) = \sum_d f_1(a,b)f_2(b,c)f_3(c,d)f_4(d) = f_1(a,b)f_2(b,c)\underbrace{\sum_d f_3(c,d)f_4(d)}_{\mu_{d\to c}(c)}.$$

- В данном выражении $\mu_{d\to c}(c)$ определяет *сообщение* от узла d к узлу c.
- Аналогично: $$p(a,b) = f_1(a,b)\underbrace{\sum_c f_2(b,c)\,\mu_{d\to c}(c)}_{\mu_{c\to b}(b)}.$$

Сообщения между факторами и переменными



- Рассмотрим более *сложный случай*:

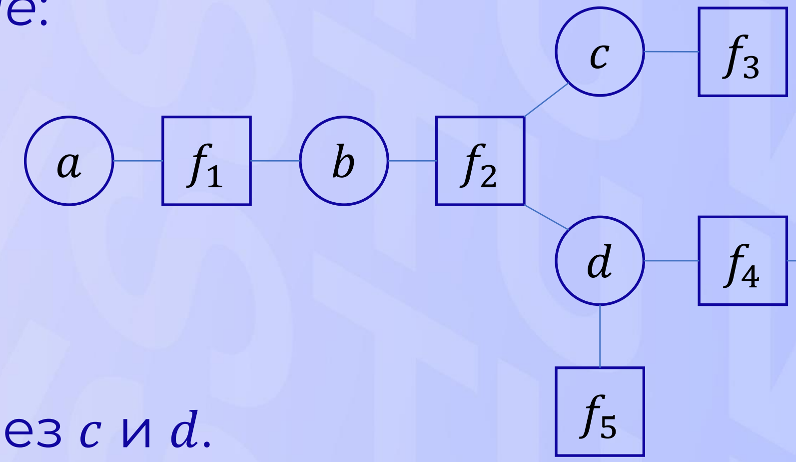
$$p(a,b,c,d,e) = p(a \mid b)p(b \mid c,d)p(c)p(e \mid d)p(d).$$

- Или в <u>нотации факторов</u>:

$$p(a,b,c,d,e) = f_1(a,b)f_2(b,c,d)f_3(c)f_4(d,e)f_5(d).$$

- Вычислим *маргинальное распределение*:

$$p(a,b) = f_1(a,b)\underbrace{\sum_{c,d} f_2(b,c,d)f_3(c)f_5(d)\sum_{e} f_4(d,e)}_{\mu_{f_2\to b}(b)}.$$



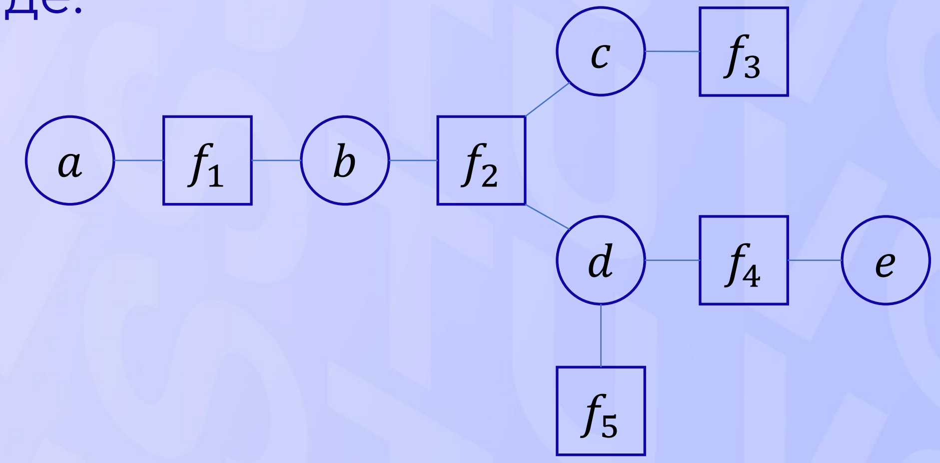
- Здесь $\mu_{f_2\to b}(b)$ обозначает <u>*сообщение от фактора к переменной*</u>. Оно состоит из <u>сообщений</u>, полученных <u>из двух ветвей</u> через c и d.

VOLGOGRAD STATE TECHNICAL UNIVERSITY

Сообщения между факторами и переменными

- Выражение $\mu_{f_2\to b}(b)$ можно записать в виде:

$$\mu_{f_2\to b}(b) = \sum_{c,d} f_2(b,c,d)\ \underbrace{f_3(c)}_{\mu_{c\to f_2}(c)}\ \underbrace{f_5(d)\sum_{e} f_4(d,e)}_{\mu_{d\to f_2}(d)}.$$



- Аналогично:

$$\mu_{d\to f_2}(d) = \underbrace{f_5(d)}_{\mu_{f_5\to d}(d)}\ \underbrace{\sum_{e} f_4(d,e)}_{\mu_{f_4\to d}(d)}.$$

- Таким же способом можно посчитать *маргинальное распределение* $p(a)$:

$$p(a) = \underbrace{\sum_{b} f_1(a,b)\mu_{f_2\to b}(b)}_{\mu_{f_1\to a}(a)},$$

$$\mu_{f_1\to a}(a) = \sum_{b} f_1(a,b)\,\underbrace{\mu_{f_2\to b}(b)}_{\mu_{b\to f_1}(b)}.$$

Сообщения между факторами и переменными

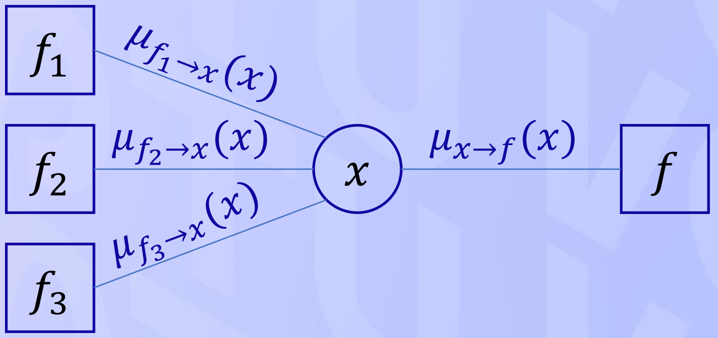
- Можно увидеть, что *сообщение от фактора к переменной* формируется с помощью *суммирования произведений* входящих *сообщений от переменной к фактору*.
- Аналогично, *сообщение от переменной к фактору* определяется *произведением* входящих *сообщений от фактора к переменной*.
- *Удобство* такого подхода заключается в *возможности повторного использования* рассчитанных сообщений для оценки других маргинальных распределений.
- Например, *маргинальное распределение* $p(b)$ может быть выражено так:

$$p(b) = \underbrace{\sum_a f_1(a, b)}_{\mu_{f_1 \to b}(b)} \mu_{f_2 \to b}(b).$$

Передача сообщений в общем виде

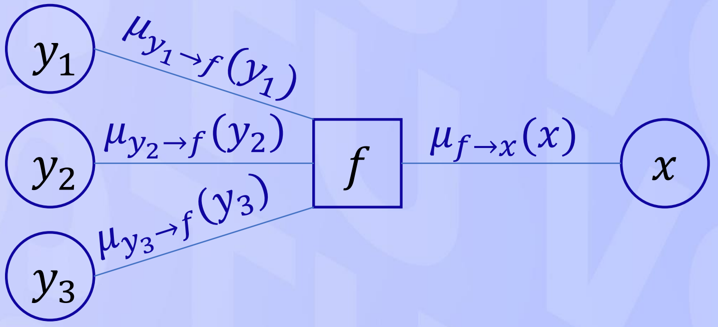
- Таким образом, *<u>передачу сообщений</u>* в общем виде можно записать так:
  - Сообщение <u>от переменной к фактору</u>:

$$\mu_{x\to f}(x) = \prod_{g\,\in\,\text{neighbors}(x)\setminus f} \mu_{g\to x}(x)\,,$$



  - Сообщение <u>от фактора к переменной</u>:

$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f\setminus x} f(\mathcal{X}_f) \prod_{y\,\in\,\text{neighbors}(f)\setminus x} \mu_{y\to f}(y)\,,$$

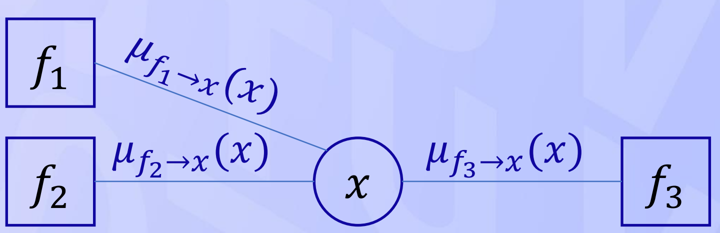


Суммирование выполняется по всем значениям переменных $\mathcal{X}_f\setminus x$.

Алгоритм суммирования-произведения (Sum-Product algorithm)

- Сообщения от <u>факторов в *листьях* графа</u> инициализируются <u>*фактором*</u>.
- Сообщения от <u>переменных в *листьях* графа</u> устанавливаются в <u>*единицу*</u>.
- <u>Маргинализация</u> переменной в этом случае может быть рассчитана так:

$$p(x) = \frac{1}{Z} \prod_{f \in \mathrm{neighbors}(x)} \mu_{f \to x}(x) \, .$$



- Данная процедура называется <u>*алгоритмом суммирования-произведения*</u>
- Сообщения в алгоритме суммирования-произведения рассчитываются в *определённом порядке*. Он часто называется планировкой сообщений (*message schedule*). Сообщения, <u>от которых зависят</u> другие сообщения, рассчитываются <u>*перед ними*</u>.

Пример алгоритма суммирования-произведения

- Рассмотрим следующий *<u>факторный граф</u>* с *<u>бинарными</u>* переменными x_1, x_2, x_3 и факторами $f_1(x_1, x_2)$ и $f_2(x_2, x_3)$, заданными, например, *<u>таблично</u>*:

$$x_1 - \boxed{f_1} - x_2 - \boxed{f_2} - x_3 \qquad\qquad p(x_1, x_2, x_3) = \frac{1}{Z} \cdot f_1(x_1, x_2) \cdot f_2(x_2, x_3).$$

- Требуется вычислить *<u>маргинальное распределение</u>* $p(x_2)$.
- *Инициализация*: $\mu_{x_1 \to f_1}(x_1) = 1$, $\mu_{x_3 \to f_2}(x_3) = 1$.
- Вычисление *сообщений от факторов* f_1 и f_2 *к переменной* x_2:

$$\mu_{f_1 \to x_2}(x_2) = \sum_{x_1} f_1(x_1, x_2) \mu_{x_1 \to f_1}(x_1) = \sum_{x_1} f_1(x_1, x_2) \cdot 1 = \sum_{x_1} f_1(x_1, x_2)\,;$$

$$\mu_{f_2 \to x_2}(x_2) = \sum_{x_3} f_2(x_2, x_3) \mu_{x_3 \to f_2}(x_3) = \sum_{x_3} f_2(x_2, x_3) \cdot 1 = \sum_{x_3} f_2(x_2, x_3)\,.$$

Пример алгоритма суммирования-произведения

- *Маргинальное распределение* $p(x_2)$ пропорционально *произведению* всех *входящих сообщений* в x_2:

$$p(x_2) \propto \mu_{f_1 \to x_2}(x_2) \cdot \mu_{f_2 \to x_2}(x_2).$$

- С учетом *полученных сообщений*:

$$p(x_2) \propto \left(\sum_{x_1} f_1(x_1, x_2)\right) \cdot \left(\sum_{x_3} f_2(x_2, x_3)\right).$$

- Для получения *точного распределения* $p(x_2)$ потребуется *нормализовать* результат по *всем значениям переменной* x_2:

$$p(x_2) = \frac{\left(\sum_{x_1} f_1(x_1, x_2)\right) \cdot \left(\sum_{x_3} f_2(x_2, x_3)\right)}{\sum_{x_2}\left(\sum_{x_1} f_1(x_1, x_2)\right) \cdot \left(\sum_{x_3} f_2(x_2, x_3)\right)}.$$

Структура Junction Tree

- Факторные графы представляют собой *совместное распределение вероятностей* в виде произведения факторов.
- Если *факторный граф достаточно сложный*, то прямое вычисление маргинальных или условных вероятностей может быть вычислительно сложным из-за большого числа переменных и их зависимостей.
- Эту проблему решает **Junction tree**, преобразуя граф в *древовидную структуру*, где вычисления становятся *локальными* и *эффективными*.
- Junction tree – это *дерево*, построенное на основе исходного графа факторов, и содержащее:
  - ✓ *вершины* (кластеры переменных, которые связаны между собой в исходном графе);
  - ✓ *рёбра* (сепараторы, содержащие переменные, общие для обоих кластеров).

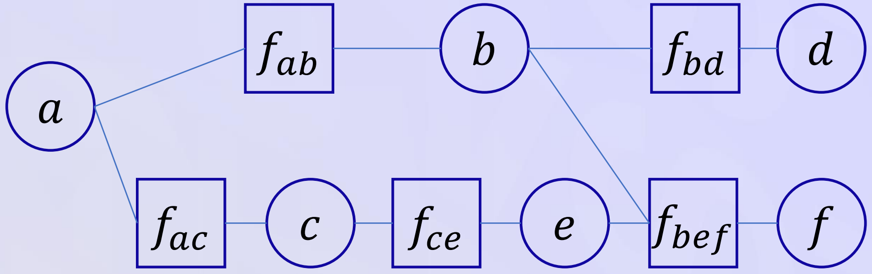
Алгоритм построения Junction Tree

1. Построение <u>морального графа</u> (moral graph).
   - ✓ <u>*Морализованная*</u> копия направленного ациклического графа образуется добавлением рёбер между всеми парами узлов, которые имеют общих детей, а затем преобразования всех рёбер в графе в неориентированные.
     - Для каждого фактора нужно <u>*соединить все переменные*</u>, которые входят в этот фактор. При этом образуется <u>*клика*</u> (полный подграф).
     - А затем нужно <u>*удалить все факторы*</u>, оставив только переменные и ребра между ними.
2. Выполнение <u>триангуляции графа</u>.
   - ✓ Граф называется <u>*хордальным*</u>, если каждый из его длинных циклов (имеющих четыре ребра и более) имеет хорду. Для построения **`junction tree`** граф должен быть *хордальным*. Процесс добавления ребер для устранения длинных циклов называется <u>*триангуляцией*</u>.
   - ✓ Для выбранного <u>*порядка устранения переменных*</u> (variables elimination):
     - для каждой *устраняемой переменной* нужно <u>*соединить ребром всех её соседей*</u> в текущем графе, чтобы избежать длинных циклов. Добавляемые рёбра <u>*остаются в графе*</u>.
     - Этот процесс <u>*повторяется для всех узлов*</u> в графе.

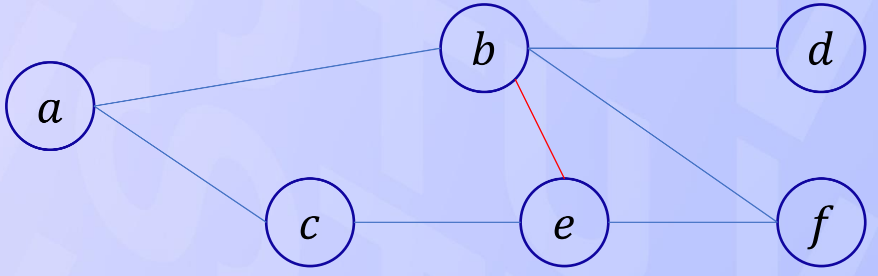
Алгоритм построения Junction Tree

3. Определение максимальных клик (clique) в графе.
   - ✓ *Максимальная клика* — это *полный подграф*, который не может быть расширен добавлением новых вершин без потери свойства полноты.
4. Объединение максимальных клик в граф кластеров.
   - ✓ Нужно построить *взвешенный граф* из определённых *максимальных клик*, в котором:
     - каждая *максимальная* клика (кластер) соответствует *одной вершине*;
     - каждое *ребро между вершинами графа* (кликами) имеет *вес*, равный *количеству общих переменных* в вершинах.
5. Определение максимального остовного дерева в графе (**Junction Tree**).
   - ✓ *Максимальное остовное дерево* — это *дерево*, включающее все вершины *взвешенного неориентированного графа*, *максимизирующее общий вес рёбер*.
6. Распределение факторов.
   - ✓ *Каждый фактор* из исходного факторного графа должен быть *назначен одному из кластеров* (вершин, клик) `junction tree`, который содержит *все переменные этого фактора*.

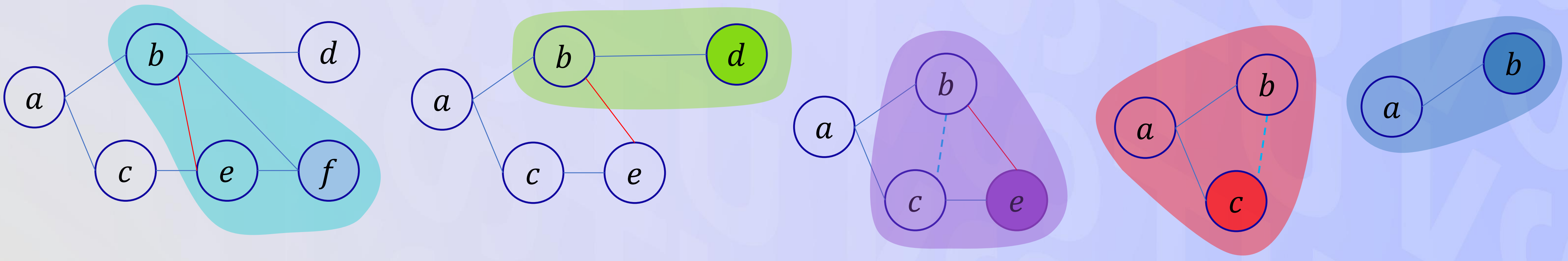
Пример построения Junction Tree



Исходный факторный граф



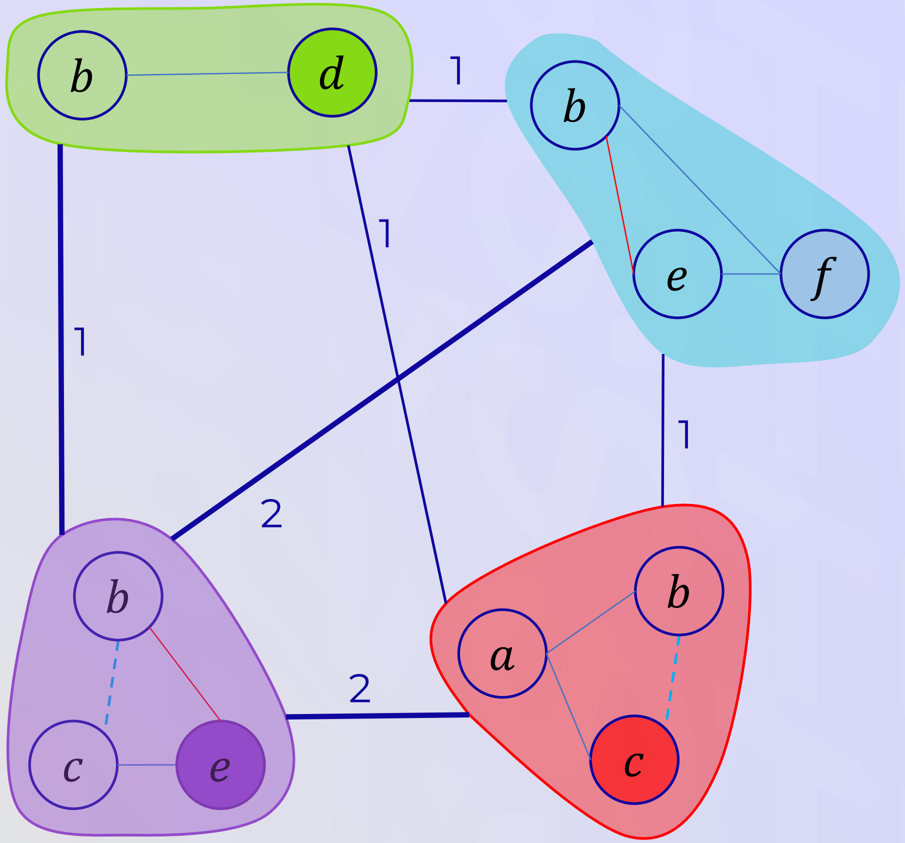
Моразлизованная версия графа



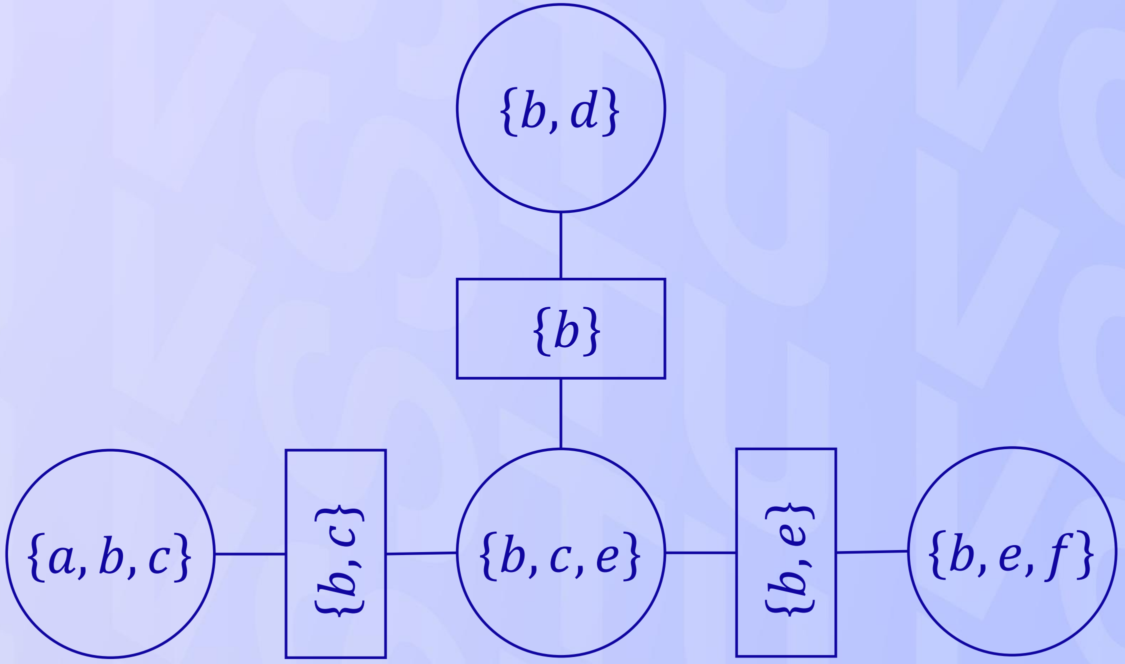
Триангуляция графа в последовательности $\{f, d, e, c, b, a\}$

VOLGOGRAD STATE TECHNICAL UNIVERSITY

Пример построения Junction Tree



Граф кластеров

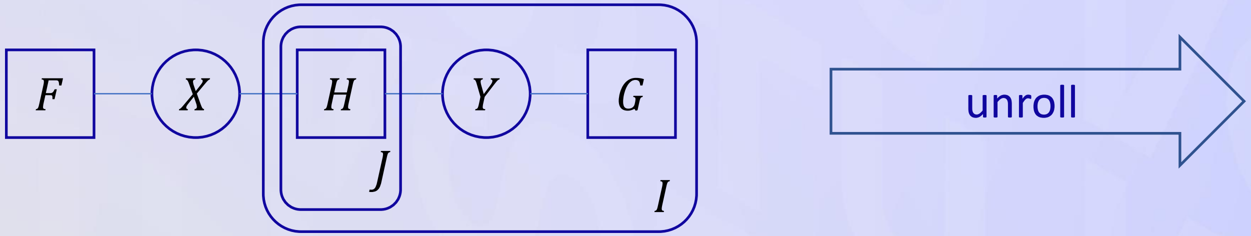


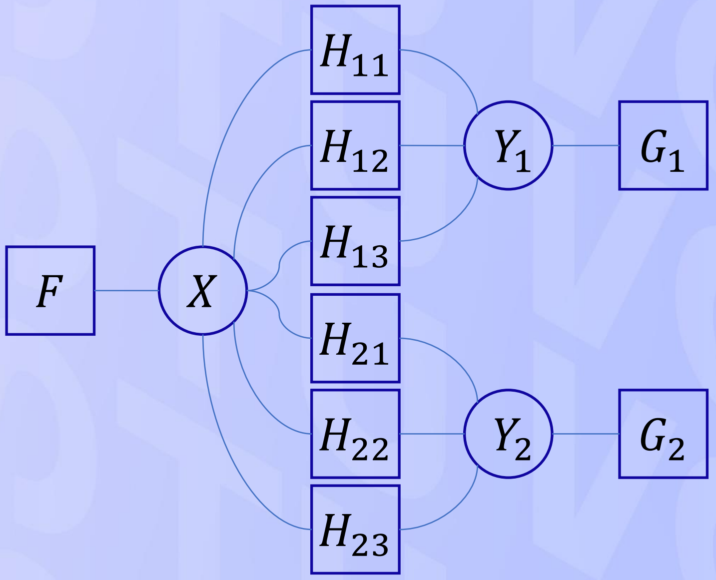
Junction Tree

- Основная идея *алгоритма Junction Tree* заключается в преобразовании факторного графа в дерево кластеров, на котором эффективно выполняется *алгоритм устранения переменных*.
- Затем в Junction Tree просто выполняется передача сообщений.
- Однако, построение Junction Tree *неоднозначно*. Например, всегда можно найти *тривиальное Junction Tree с одной вершиной*, содержащей все переменные исходного графа. Но данное дерево *бесполезно*.
- *Оптимальными* можно считать деревья, которые делают кластеры настолько *маленькими и модульными*, насколько это возможно.
- Но нахождение *оптимального дерева* является *NP-сложной задачей*.

Алгоритм суммирования-произведения для plate-графов факторов

- А как выполнять процедуру *variable elimination* для *plated-графов*?
- Можно реализовать алгоритм *суммирования-произведения* на *графе факторов с plate-нотацией*, применив его к *развернутой версии* графа:





- Однако, такое решение *ограничивает параллелизм*, *неэффективно использует память* и *усложняет связь* с исходной моделью.
- В 2019 году F. Obermeyer et al. предложили *новый алгоритм: Tensor variable elimination.*

Алгоритм устранения тензорной переменной (Tensor variable elimination)

Предложенный алгоритм в *псевдокоде* записывается следующим образом:

$S \leftarrow \mathbf{list}([])$.

while (есть факторы в графе G):

$L \leftarrow$ *leaf plate set* в G максимального размера;
$G_L \leftarrow$ подграф графа G в L;
for подграф G_C in $\mathbf{Partition}(G_L)$:
$f \leftarrow \mathbf{SumProduct}(G_C)$;
Удалить подграф G_C из G
If $V_f = \emptyset$ (множество оставшихся в f переменных):
$S.append(\mathbf{Product}(f, L))$
else:
$L' \leftarrow$ plate всех переменных f в G;
if $L' = L$ **then error** (“Intractable!”)
$f' \leftarrow \mathbf{Product}(f, L - L')$;
вставить f' в G;

return $\mathbf{SumProduct}(S, \{\})$.

Вычисляет сильно связанные компоненты двудольного графа

Выполняет устранение переменных в группе структурно идентичных факторных графов

Выполняет поэлементное произведение факторов по одному или нескольким индексам в plate

Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow \textbf{list}([])$.

while (есть факторы в графе G):

$L \leftarrow$ *leaf plate set* в G максимального размера;

$G_L \leftarrow$ подграф графа G в L;

for подграф G_C in $\textbf{Partition}(G_L)$:

$f \leftarrow \textbf{SumProduct}(G_C)$;

Удалить подграф G_C из G

If $V_f = \emptyset$ (множество оставшихся в f переменных):

$S.append(\textbf{Product}(f, L))$

else:

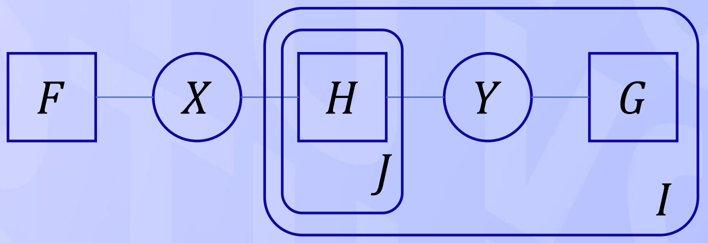
$L' \leftarrow$ plate всех переменных f в G;

if $L' = L$ **then error** ("Intractable!")

$f' \leftarrow \textbf{Product}(f, L - L')$;

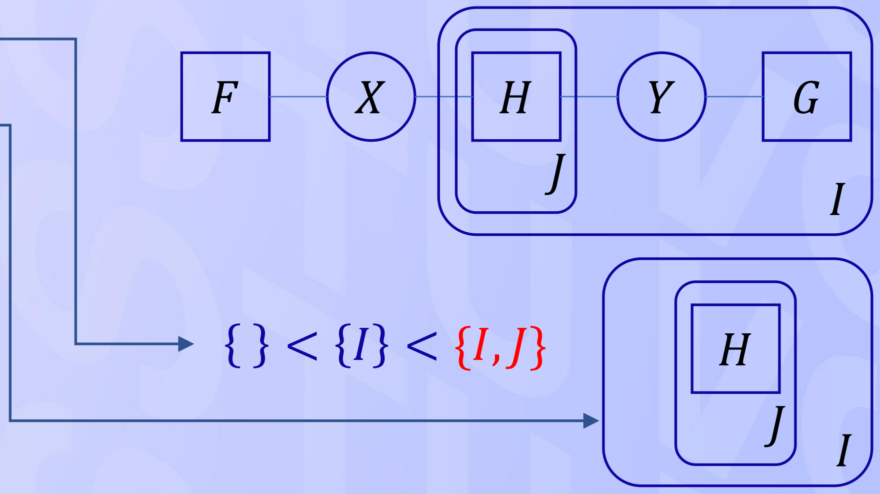
вставить f' в G;

return $\textbf{SumProduct}(S, \{\})$.



Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow$ **list**([]).
while (есть факторы в графе G):
    $L \leftarrow$ <u>*leaf plate set*</u> в G максимального размера;
    $G_L \leftarrow$ подграф графа G в L;
    for подграф G_C in **Partition**(G_L) :
        $f \leftarrow$ **SumProduct**(G_C);
        Удалить подграф G_C из G
        If $V_f = \emptyset$ (множество оставшихся в f переменных):
            $S.append(\mathbf{Product}(f, L))$
        else:
            $L' \leftarrow$ plate всех переменных f в G;
            if $L' = L$ **then error** ("Intractable!")
            $f' \leftarrow \mathbf{Product}(f, L - L')$;
            вставить f' в G;
return SumProduct$(S, \{\})$.



VOLGOGRAD STATE TECHNICAL UNIVERSITY

Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow \mathbf{list}([])$.

while (есть факторы в графе G):

  $L \leftarrow$ *leaf plate set* в G максимального размера;

  $G_L \leftarrow$ подграф графа G в L;

  for подграф G_C in $\mathbf{Partition}(G_L)$:

    $f \leftarrow \mathbf{SumProduct}(G_C)$;

    Удалить подграф G_C из G

    If $V_f = \emptyset$ (множество оставшихся в f переменных):

      $S.append(\mathbf{Product}(f, L))$

    else:

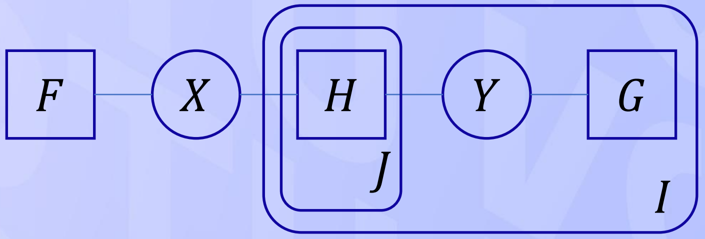
      $L' \leftarrow$ plate всех переменных f в G;

      if $L' = L$ **then error** ("Intractable!")

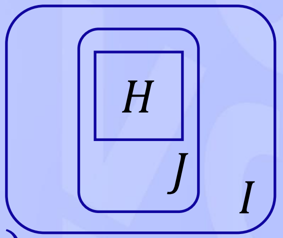
      $f' \leftarrow \mathbf{Product}(f, L - L')$;

      вставить f' в G;

return $\mathbf{SumProduct}(S, \{\})$.



$\{\} < \{I\} < \{I, J\}$



$$A_i(x, y_i) = \prod_j H_{ij}(x, y_i)$$

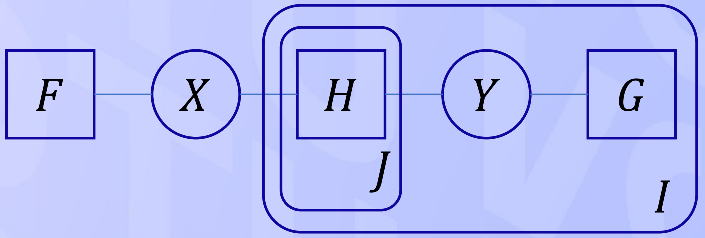
Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow$ **list**([]).

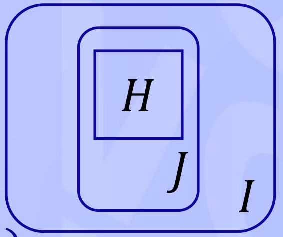
while (есть факторы в графе G):

  $L \leftarrow$ <u>*leaf plate set*</u> в G максимального размера;
  $G_L \leftarrow$ подграф графа G в L;
  for подграф G_C in $\mathbf{Partition}(G_L)$:
    $f \leftarrow \mathbf{SumProduct}(G_C)$;
    Удалить подграф G_C из G
    If $V_f = \emptyset$ (множество оставшихся в f переменных):
      $S.append(\mathbf{Product}(f, L))$
    else:
      $L' \leftarrow$ plate всех переменных f в G;
      if $L' = L$ **then error** ("Intractable!")
      $f' \leftarrow \mathbf{Product}(f, L - L')$;
      вставить f' в G;

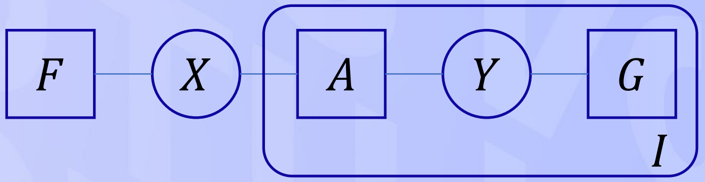
return $\mathbf{SumProduct}(S, \{\})$.



$\{\,\} < \{I\} < \{I, J\}$



$$A_i(x, y_i) = \prod_j H_{ij}(x, y_i)$$



Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow \textbf{list}([])$.

while (есть факторы в графе G):

  $L \leftarrow$ *leaf plate set* в G максимального размера;

  $G_L \leftarrow$ подграф графа G в L;

  for подграф G_C in $\textbf{Partition}(G_L)$:

    $f \leftarrow \textbf{SumProduct}(G_C)$;

    Удалить подграф G_C из G

    If $V_f = \emptyset$ (множество оставшихся в f переменных):

      $S.append(\textbf{Product}(f, L))$

    else:

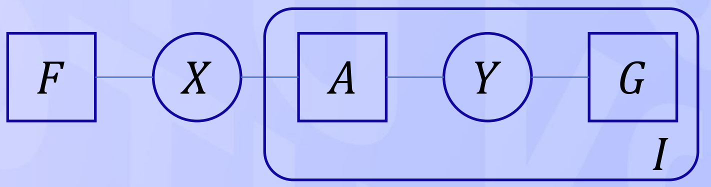
      $L' \leftarrow$ plate всех переменных f в G;

      if $L' = L$ **then error** ("Intractable!")

      $f' \leftarrow \textbf{Product}(f, L - L')$;

      вставить f' в G;

return $\textbf{SumProduct}(S, \{\})$.



VOLGOGRAD STATE TECHNICAL UNIVERSITY

Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow \mathbf{list}([])$.

while (есть факторы в графе G):

  $L \leftarrow$ *leaf plate set* в G максимального размера;

  $G_L \leftarrow$ подграф графа G в L;

  for подграф G_C in $\mathbf{Partition}(G_L)$:

    $f \leftarrow \mathbf{SumProduct}(G_C)$;

    Удалить подграф G_C из G

    If $V_f = \emptyset$ (множество оставшихся в f переменных):

      $S.append(\mathbf{Product}(f, L))$

    else:

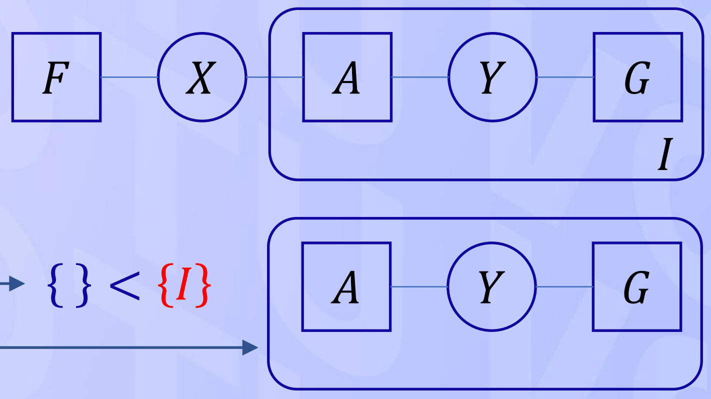
      $L' \leftarrow$ plate всех переменных f в G;

      if $L' = L$ **then error** (“Intractable!”)

      $f' \leftarrow \mathbf{Product}(f, L - L')$;

      вставить f' в G;

return $\mathbf{SumProduct}(S, \{\})$.



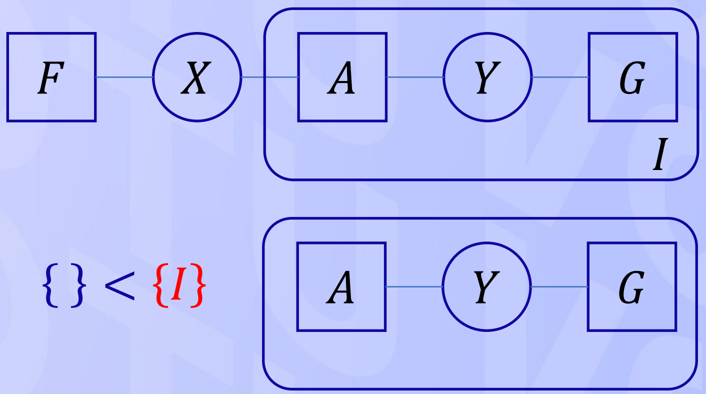
Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow$ **list**([]).

while (есть факторы в графе G):

  $L \leftarrow$ *leaf plate set* в G максимального размера;
  $G_L \leftarrow$ подграф графа G в L;
  for подграф G_C in **Partition**(G_L) :
    $f \leftarrow$ **SumProduct**(G_C);
    Удалить подграф G_C из G
    If $V_f = \emptyset$ (множество оставшихся в f переменных):
      $S.append($**Product**$(f, L))$
    else:
      $L' \leftarrow$ plate всех переменных f в G;
      if $L' = L$ **then error** ("Intractable!")
      $f' \leftarrow$ **Product**$(f, L - L')$;
      вставить f' в G;

return SumProduct$(S, \{\})$.



$$B(x) = \prod_i \sum_{y_i} A_i(x, y_i) \cdot G_i(y_i)$$

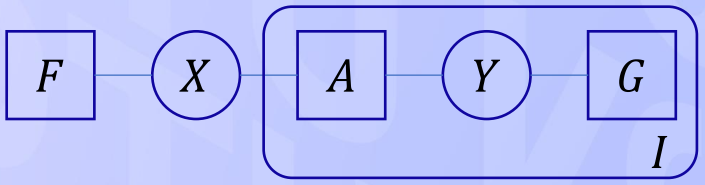
Алгоритм устранения тензорной переменной (Tensor variable elimination)

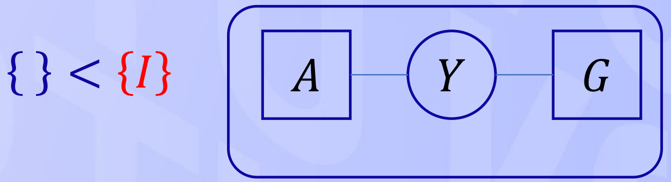
$S \leftarrow$ **list**([]).

while (есть факторы в графе G):

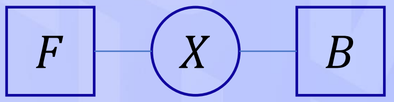
    $L \leftarrow$ <u>*leaf plate set*</u> в G максимального размера;
    $G_L \leftarrow$ подграф графа G в L;
    for подграф G_C in $\mathbf{Partition}(G_L)$:
        $f \leftarrow \mathbf{SumProduct}(G_C)$;
        Удалить подграф G_C из G
        If $V_f = \emptyset$ (множество оставшихся в f переменных):
            $S.append(\mathbf{Product}(f, L))$
        else:
            $L' \leftarrow$ plate всех переменных f в G;
            if $L' = L$ **then error** (“Intractable!”)
            $f' \leftarrow \mathbf{Product}(f, L - L')$;
            вставить f' в G;

return $\mathbf{SumProduct}(S, \{\})$.





$$B(x) = \prod_i \sum_{y_i} A_i(x, y_i) \cdot G_i(y_i)$$



VSTU VOLGOGRAD STATE TECHNICAL UNIVERSITY

Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow \mathbf{list}([])$.

while (есть факторы в графе G):

  $L \leftarrow$ *leaf plate set* в G максимального размера;

  $G_L \leftarrow$ подграф графа G в L;

  for подграф G_C in $\mathbf{Partition}(G_L)$:

    $f \leftarrow \mathbf{SumProduct}(G_C)$;

    Удалить подграф G_C из G

    If $V_f = \emptyset$ (множество оставшихся в f переменных):

      $S.append(\mathbf{Product}(f, L))$

    else:

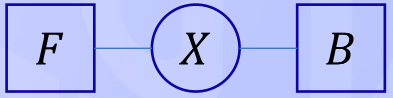
      $L' \leftarrow$ plate всех переменных f в G;

      if $L' = L$ **then error** ("Intractable!")

      $f' \leftarrow \mathbf{Product}(f, L - L')$;

      вставить f' в G;

return $\mathbf{SumProduct}(S, \{\})$.



Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow \mathbf{list}([])$.

while (есть факторы в графе G):

  $L \leftarrow$ leaf plate set в G максимального размера;

  $G_L \leftarrow$ подграф графа G в L;

  for подграф G_C in $\mathbf{Partition}(G_L)$:

    $f \leftarrow \mathbf{SumProduct}(G_C)$;

    Удалить подграф G_C из G

    If $V_f = \emptyset$ (множество оставшихся в f переменных):

      $S.append(\mathbf{Product}(f, L))$

    else:

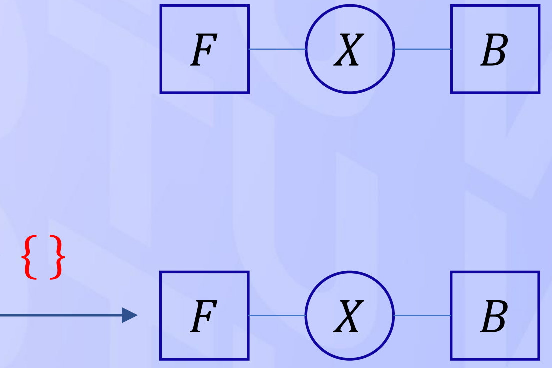
      $L' \leftarrow$ plate всех переменных f в G;

      if $L' = L$ **then error** ("Intractable!")

      $f' \leftarrow \mathbf{Product}(f, L - L')$;

      вставить f' в G;

return $\mathbf{SumProduct}(S, \{\})$.



VSTU VOLGOGRAD STATE TECHNICAL UNIVERSITY

Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow$ **list**([]).

while (есть факторы в графе G):

  $L \leftarrow$ *leaf plate set* в G максимального размера;

  $G_L \leftarrow$ подграф графа G в L;

  for подграф G_C in **Partition**(G_L) :

    $f \leftarrow$ **SumProduct**(G_C);

    Удалить подграф G_C из G

    If $V_f = \emptyset$ (множество оставшихся в f переменных):

      $S.append$(**Product**(f, L))

    else:

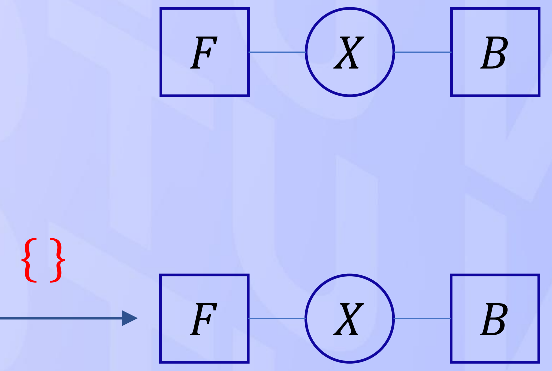
      $L' \leftarrow$ plate всех переменных f в G;

      if $L' = L$ **then error** (“Intractable!”)

      $f' \leftarrow$ **Product**($f, L - L'$);

      вставить f' в G;

return **SumProduct**(S, {}).



$$C = \prod_1 \sum_{x_i} F_i(x_i) \cdot B_i(x_i)$$

Алгоритм устранения тензорной переменной (Tensor variable elimination)

$S \leftarrow \mathbf{list}([])$.

while (есть факторы в графе G):

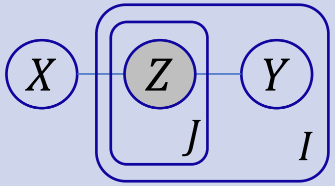
  $L \leftarrow$ *leaf plate set* в G максимального размера;
  $G_L \leftarrow$ подграф графа G в L;
  for подграф G_C in $\mathbf{Partition}(G_L)$:
    $f \leftarrow \mathbf{SumProduct}(G_C)$;
    Удалить подграф G_C из G
    If $V_f = \emptyset$ (множество оставшихся в f переменных):
      $S.append(\mathbf{Product}(f, L))$
    else:
      $L' \leftarrow$ plate всех переменных f в G;
      if $L' = L$ **then error** ("Intractable!")
      $f' \leftarrow \mathbf{Product}(f, L - L')$;
      вставить f' в G;

return $\mathbf{SumProduct}(S, \{\})$.

VSTU VOLGOGRAD STATE TECHNICAL UNIVERSITY

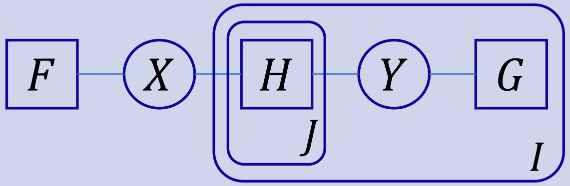
Реализация алгоритма устранения тензорной переменной в Pyro

Высокоуровневый интерфейс для определения модели:




```
@pyro.infer.config_enumerate
def model(z):
  I, J = z.shape
  x = pyro.sample("x", Bernoulli(Px))
  with pyro.plate("I", I):
    y = pyro.sample("y", Bernoulli(Py))
    with pyro.plate("J", J):
      pyro.sample(
        "z", Bernoulli(Pz[x,y]),obs=z)
```

Низкоуровневый интерфейс для определения графа факторов:




```
pyro.ops.contract.einsum(
  "x,iy,ijxy->",
  F, G, H,
  plates="ij"
)
```

- Декоратор *config_enumerate* предписывает выполнить устранение всех дискретных переменных.
- Если требуется выполнить процедуру устранения только для некоторых дискретных переменных, то при их объявлении в метод *sample()* нужно передать параметр: *infer={"enumerate": "parallel"}* *или* *infer={"enumerate": "sequential"}* .
- Все дискретные переменные, участвующие в процедуре *устранения*, должны отсутствовать в вариационном распределении (*guide*). Если guide формируется автоматически, устраняемые переменные необходимо указать *явно* при создании класса guide:

```
guide = AutoNormal(poutine.block(model, hide=["x", "data"]))
```

Особенности реализации алгоритма устранения переменной в Pyro

- Стратегия устранения переменной может использоваться *совместно* с методами *HMC*, *NUTS*, *SVI*.
- Для работы с *процедурой устранения переменной* необходимо использовать *варианты классов ELBO* с приставкой "*Enum_*".
- Для получения доступа к *предсказанным в ходе вариационного инференса* дискретным переменным необходимо использовать обработчик *infer_discrete*:

```
serving_model = infer_discrete(model, first_available_dim=-1)
```

- Работа процедуры *устранения переменной* базируется на работе с *формой тензоров* и использует правила *broadcasting*.

Форма тензора и распределения в PyTorch и Pyro

- Тензор, содержащий количество элементов некоторого тензора *вдоль каждого его измерения*, называется *формой тензора* (*shape*).
- Тензор в PyTorch имеет только *один* атрибут формы (*.shape*):

```
x = torch.Tensor([[1, 2], [3, 4], [5,6]])
assert x.shape == torch.Size([3, 2])
```

- Распределение имеет *два* атрибута формы (*.batch_shape* и *.event_shape*)

```
y = dist.MultivariateNormal(torch.zeros([2, 3]), diag(torch.ones(3)))
assert (y.batch_shape, y.event_shape) == (torch.Size([2]), torch.Size([3]))
```

- *Индексы* в *.batch_shape* определяют условно-независимые переменные, а в *.event_shape* – зависимые (один семпл из распределения). Их *комбинация* определяет *полную форму*.

Преобразование формы распределений (Reshaping distributions)

- В Pyro можно изменять *размерности* и *формы* распределений.
- Для получения *батча семплов* можно к распределению применить метод expand() или воспользоваться *нотацией* plate. Данные батчи *условно независимы*. Приведенные способы изменяют *batch_shape* распределения.
- Некоторую размерность распределения можно объявить *зависимой*. Для этого используется метод распределения to_event(n), в который передается количество размерностей в *batch_shape* *справа*, которые должны интерпретироваться как зависимые. После вызова метода они добавляются к размерностям в *event_shape* слева.

```
d = Bernoulli(torch.tensor([0.1, 0.2, 0.3, 0.4])).expand([3, 4]).to_event(1)
```

- Предполагать *зависимость* переменных всегда *безопасно*.

Механика работы алгоритма устранения переменной в Pyro (enumeraton)

- Главная идея *перечисления* (*enumeration*) заключается в интерпретации операторов sample() для дискретных переменных как полного перечисления, а не как *случайной выборки*. В дальнейшем, другие алгоритмы вывода могут *суммировать перечисляемые значения*.
- Для этого для каждой *устраняемой* переменной в тензорах, зависящих от нее, вводится *новая размерность*, значения тензоров в которой зависят от *конкретного значения* устраняемой переменной.
- Данная размерность должная быть введена “*левее*” уже существующих размерностей, что контролируется *параметрами first_available_dim* и *max_plate_nesting*, передаваемыми в функции *во время вывода*.
- Для сложной индексации по устраняемой переменной применяют Vindex

Демонстрация практических примеров

Заключение

1. Рассмотрели подходы к точному инференсу дискретных латентных переменных и разобрали особенности этих подходов.
2. Познакомились с процедурой устранения переменной.
3. Ввели понятия графа факторов и сравнили его с байесовской сетью.
4. Поговорили про механизм сообщений между факторами и переменными, сформулировали метод суммирования-произведения для маргинализации.
5. Познакомились с алгоритмом Junction Tree.
6. Поговорили про особенности процедуры устранения переменной в plate-графах, разобрали в псевдокоде и на примере алгоритм устранения тензорной переменной, обсудили его реализацию в Pyro.
7. Поговорили об особенностях реализации алгоритма в Pyro.

VOLGOGRAD STATE TECHNICAL UNIVERSITY

Спасибо за внимание!

Волгоград 2025