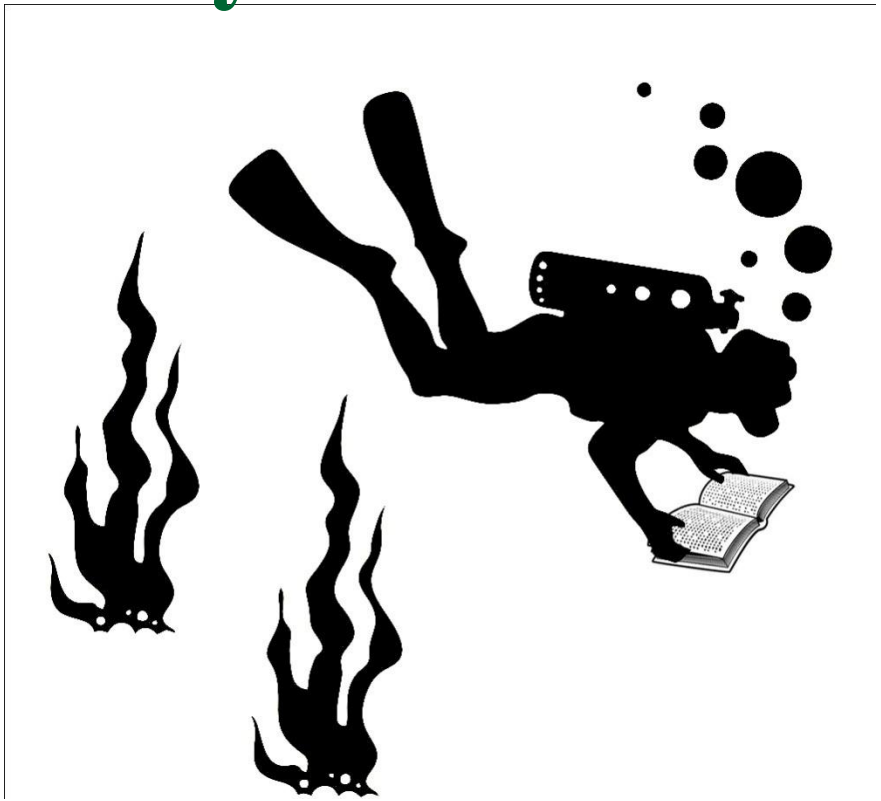
Глубокое обучение, вероятностное программирование и метавычисления: точка пересечения

Алексей Потапов
2017

SmartData, Санкт-Петербург

Глубокое обучение: польза



Машины опорных векторов: 2000-е

Проблема нелинейной оптимизации

Сделай свой выбор

$$= \frac{da_p^{(n_l)}}{dz_p^{(n_l)}} \cdot \left(\sum_{q=1}^{s_{n_l-1}} W_{p\leftarrow q}^{(n_l-1)} \cdot \left(\frac{da_q^{(n_l-1)}}{dz_q^{(n_l-1)}} \cdot \left(\sum_{k=1}^{s_{n_l-2}} W_{q\leftarrow k}^{(n_l-2)} \cdot \left(\frac{da_k^{(n_l-2)}}{dz_k^{(n_l-2)}} \cdot W_{k\leftarrow i}^{(n_l-3)} \cdot \frac{da_i^{(n_l-3)}}{dz_i^{(n_l-3)}} \cdot a_j^{(n_l-4)} \right) \right) \right) \right)$$

$$= \sum_{q=1}^{s_{n_l-1}} \sum_{k=1}^{s_{n_l-2}} \frac{da_p^{(n_l)}}{dz_p^{(n_l)}} \cdot W_{p\leftarrow q}^{(n_l-1)} \cdot \frac{da_q^{(n_l-1)}}{dz_q^{(n_l-1)}} \cdot W_{q\leftarrow k}^{(n_l-2)} \cdot \frac{da_k^{(n_l-2)}}{dz_k^{(n_l-2)}} \cdot W_{k\leftarrow i}^{(n_l-3)} \cdot \frac{da_i^{(n_l-3)}}{dz_i^{(n_l-3)}} \cdot a_j^{(n_l-4)}$$

$$= \left(\sum_{q=1}^{s_{n_l-1}} \sum_{k=1}^{s_{n_l-2}} \frac{da_p^{(n_l)}}{dz_p^{(n_l)}} \cdot W_{p\leftarrow q}^{(n_l-1)} \cdot \frac{da_q^{(n_l-1)}}{dz_q^{(n_l-1)}} \cdot W_{q\leftarrow k}^{(n_l-2)} \cdot \frac{da_k^{(n_l-2)}}{dz_k^{(n_l-2)}} \cdot W_{k\leftarrow i}^{(n_l-3)} \right) \cdot \frac{da_i^{(n_l-3)}}{dz_i^{(n_l-3)}} \cdot a_j^{(n_l-4)}$$

или

```
z = tf.sigmoid(tf.matmul(x, Wz) + bz)
logits = tf.matmul(z, Wy) + by
loss = tf.losses.sigmoid_cross_entropy(y, logits)
grad = tf.gradients(loss, x)
```

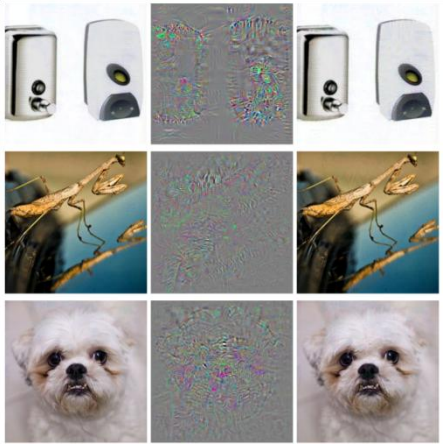
SMV/2000=

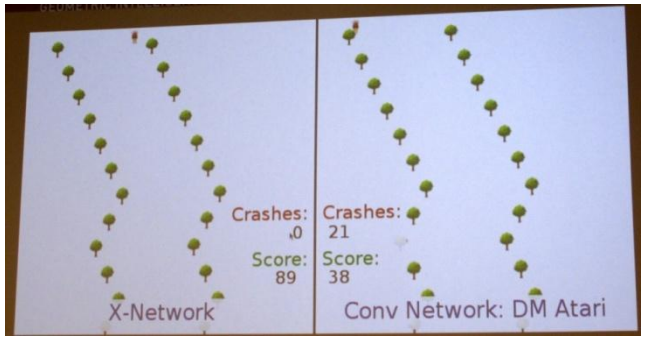
DL/2015=

??/2025

Критика глубокого обучения

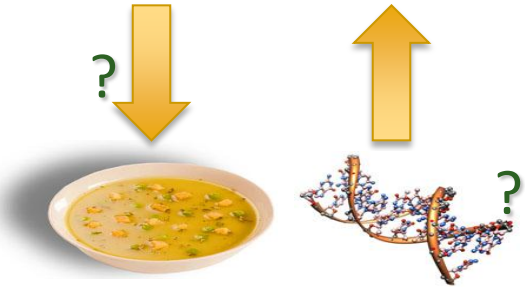
- Слабое обобщение
  - Необходимы большие выборки; нет обучения по одному примеру
  - Невозможность обучения инвариантам
  - Наличие уязвимостей
  - Проблемы с трансферным обучением и обучением без учителя
- А также
  - Кодирует статистики, но не логические, каузальные или структурные отношения
  - Трудности в рассуждениях и планировании



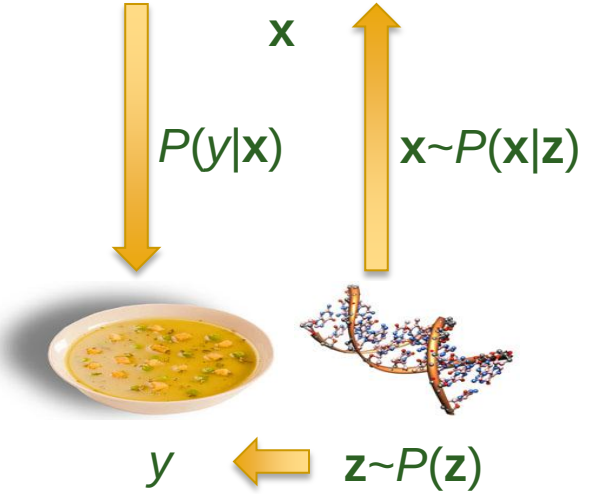


Изображения из: Szegedy, C. et al. Intriguing properties of neural networks. arXiv 1312.6199 (2013).
Gary Marcus. Keynote @ AGI-16

Генеративные и дискриминантные модели



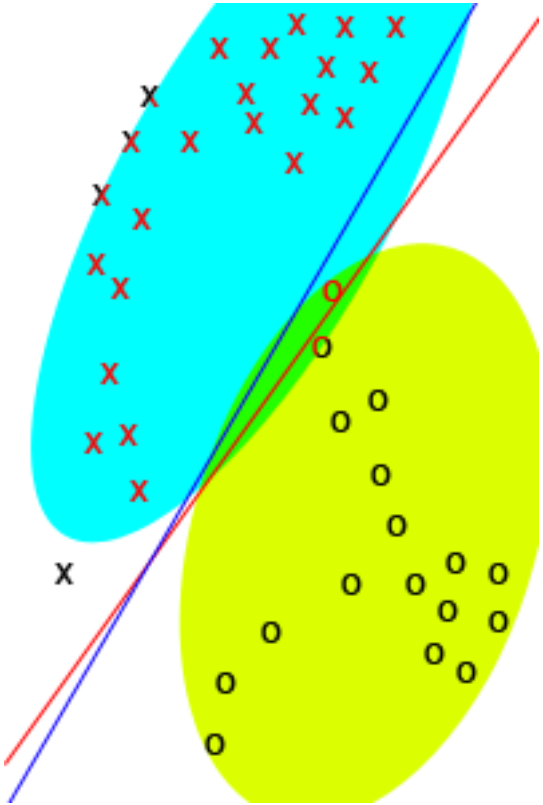
Добавим вероятностей



Вероятностные модели

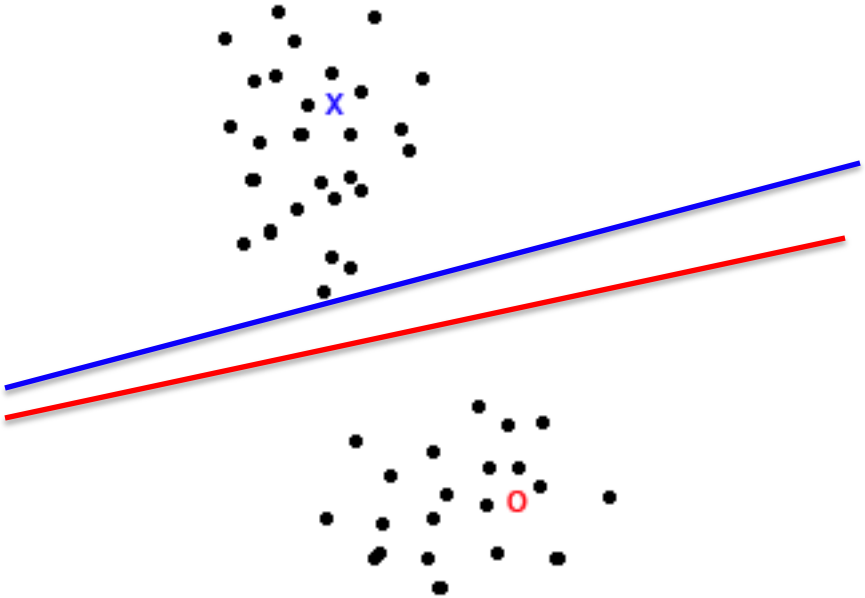
	Дискриминантные	Генеративные
	Отображение из пространства данных в пространство решений $P(y\|\mathbf{x})$	Отображение из пространства решений в пространство данных $\mathbf{z}\sim P(\mathbf{z})$, $\mathbf{x}\sim P(\mathbf{x}\|\mathbf{z})$
Pros	• Эффективные • Меньше предположений о распределении данных	• Гибкие • Обучение без учителя и с частично размеченными данными
Cons	• Вывод только в одну сторону • Только обучение с учителем	• Дополнительные предположения о распределении данных • Вывод вычислительно неэффективен

Вероятностные модели



генеративная

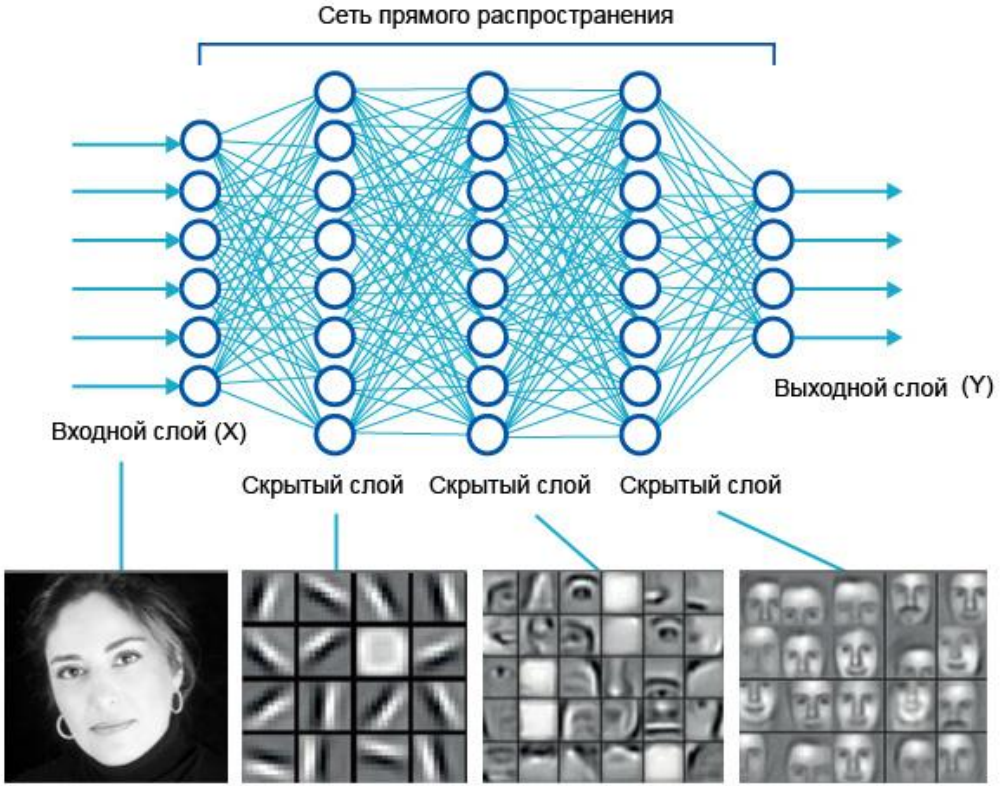
дискриминантная



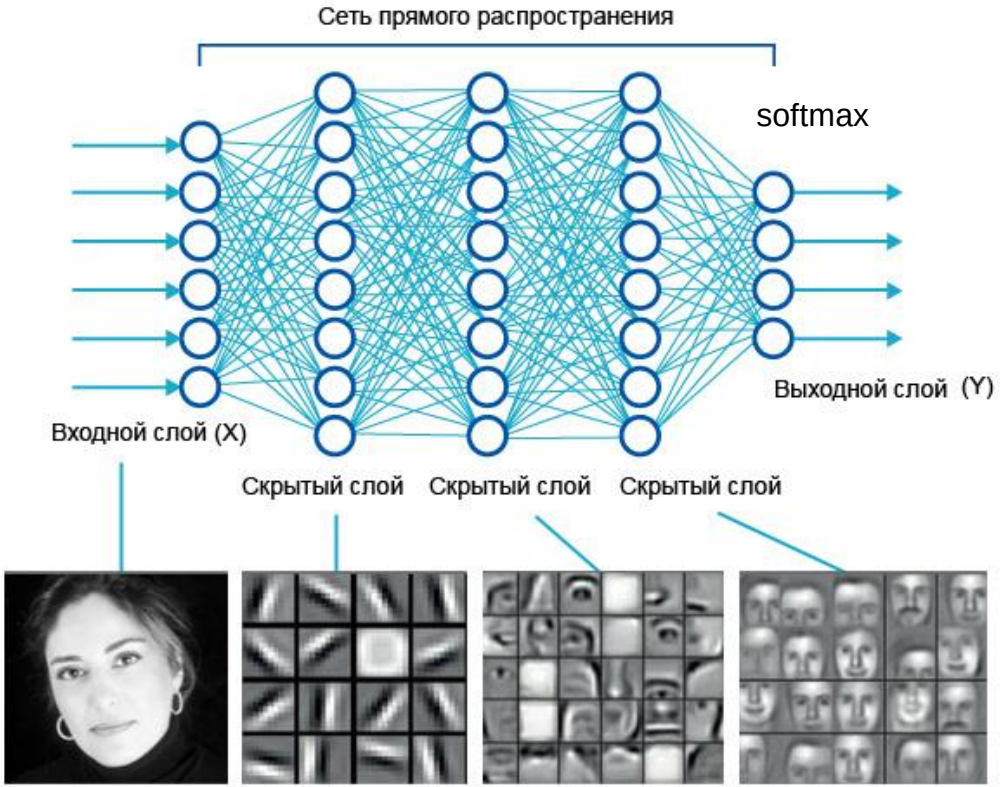
Много данных

Частичная разметка

FFNN как дискриминантные модели



FFNN как дискриминантные модели



Все вместе – прямая аппроксимация $P(y|\mathbf{x})$

Генеративные модели

- Порождение x по z
- В обратную сторону?

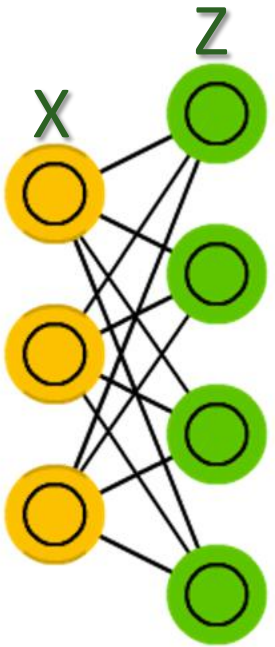
$$P(\mathbf{z}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{\sum_{\mathbf{z}} P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}$$

Генеративные модели: RBM

- Порождение x по z
- В обратную сторону?

$$P(\mathbf{z}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{\sum_{\mathbf{z}} P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}$$

- Простейшая нетривиальная вероятностная графическая модель



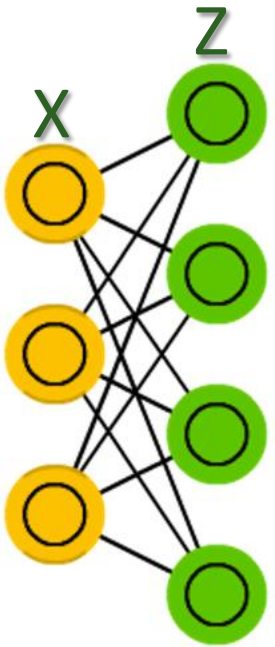
$$P(\mathbf{x}) = \sum_{\mathbf{z}} P(\mathbf{x},\mathbf{z}) = \frac{1}{Z}\sum_{\mathbf{z}} e^{\mathbf{bx}+\mathbf{cz}+\mathbf{x}^T W\mathbf{z}} \qquad P(z_i = 1|\mathbf{x}) = \sigma\left(\sum_j w_{ij}x_j + c_i\right)$$

Генеративные модели: RBM

- Порождение x по z
- В обратную сторону?

$$P(\mathbf{z}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}{\sum_{\mathbf{z}} P(\mathbf{x}|\mathbf{z})P(\mathbf{z})}$$

- Простейшая нетривиальная вероятностная графическая модель



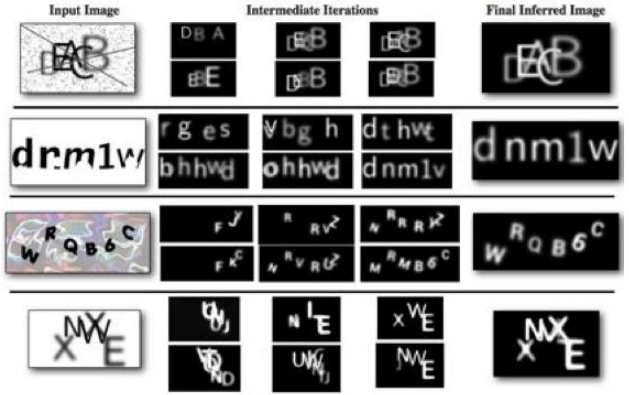
$$P(\mathbf{x}) = \sum_{\mathbf{z}} P(\mathbf{x},\mathbf{z}) = \frac{1}{Z}\sum_{\mathbf{z}} e^{\mathbf{bx}+\mathbf{cz}+\mathbf{x}^T W\mathbf{z}} \qquad P(z_i = 1|\mathbf{x}) = \sigma\left(\sum_j w_{ij}x_j + c_i\right)$$

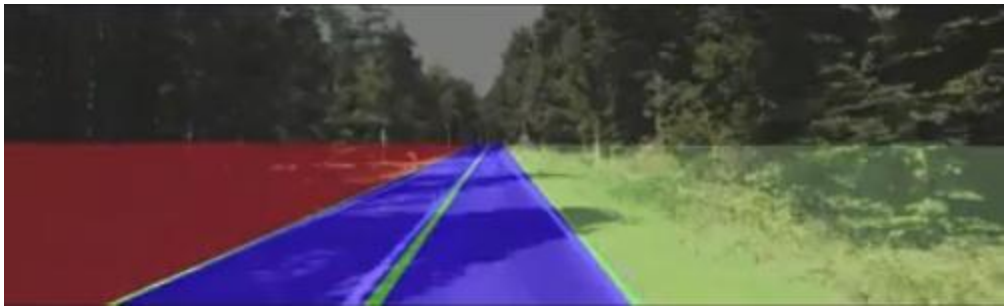
- Та же самая сигмоида, но выучивается $P(\mathbf{x}, \mathbf{z})$

Развитие вероятностных моделей

- Графические модели в компьютерном зрении, представлении знаний, глубоком обучении
- Непараметрические байесовские модели
- Вероятностное программирование

```
// Units in arbitrary, uncentered renderer coordinate system.
ASSUME road_width (uniform_discrete 5 8)
ASSUME road_height (uniform_discrete 70 150)
ASSUME lane_pos_x (uniform_continuous -1.0 1.0)
ASSUME lane_pos_y (uniform_continuous -5.0 0.0)
ASSUME lane_pos_z (uniform_continuous 1.0 3.5)
ASSUME lane_size (uniform_continuous 0.10 0.35)
ASSUME eps (gamma 1 1)
ASSUME port 8000 // External renderer and likelihood server.
ASSUME load_image (load_remote port "load_image")
ASSUME render_surfaces (load_remote port "road_renderer")
ASSUME incorporate_stochastic_likelihood (load_remote port "likelihood")
ASSUME theta_left (list 0.13 ... 0.03)
ASSUME theta_right (list 0.03 ... 0.02)
ASSUME theta_road (list 0.05 ... 0.07)
ASSUME theta_lane (list 0.01 ... 0.21)
ASSUME data (load_image "frame201.png")
ASSUME surfaces (render_surfaces lane_pos_x lange_pos_y lange_pos_z
  road_width road_height lane_size)
OBSERVE (incorporate_stochastic_likelihood theta_left theta_right
  theta_road theta_lane data surfaces eps) True
```





Изображения из: Mansinghka, V., Kulkarni, T., Perov, Y., Tenenbaum, J.: Approximate Bayesian Image Interpretation using Generative Probabilistic Graphics Programs. Advances in NIPS, arXiv:1307.0060 [cs.AI] (2013).

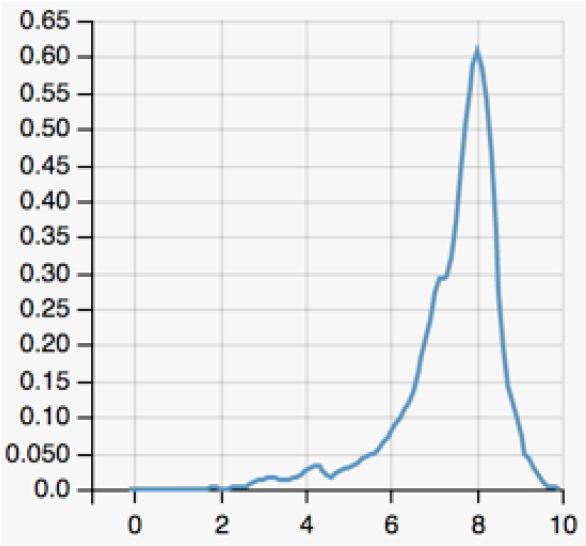
Вероятностное программирование: пример

```
var task = [10, 8, -8, -12, 15, 3]
var target = 1
var generate = function() {
    var subset = repeat(task.length, flip)
    var sum = reduce(function(x, acc)
                     { return acc + (x[1] ? x[0] : 0) },
                     0, zip(task, subset))
    condition(sum == target)
    return subset
}
Infer({method: "rejection",
    samples: 100, model: generate})
```

→ {"probs":[0.48,0.52],
"support":[[true,true,true,true,false,true],
[true,false,false,true,false,true]]}

Вероятностное программирование: обучение

- Любые модели – графические, непараметрические, рекуррентные
- Идентификация моделей, структурное обучение
- Байесовская бритва Оккама бесплатно




```
var xs = [0   , 1   , 2   , 3   ]
var ys = [0.01, 0.99, 4.02, 5.97]
var linreg = function() {
    var a = gaussian(0, 1)
    var b = gaussian(0, 1)
    var sigma = gamma(1, 1)
    var f = function(x) { return a * x + b }
    var check = function(x, y) { observe(Gaussian({mu: f(x), sigma: sigma}), y) }
    map2(check, xs, ys)
    return f(4)
}
Infer({method: 'MCMC', samples: 10000, model: linreg})
```

Обучение вероятностным программам

```
(lambda (stack-id)
  (* 2.0 (* (*
      (* -1.0 (safe-uc 0.0 2.0))
      (safe-uc (safe-uc 4.0
        (+ (safe-log 2.0) -1.0))
        (* (safe-div 2.0
          -55.61617747203855)
          (if (< (safe-uc
            (safe-uc
            27.396810474207317
             (safe-uc -1.0 2.0)) 2.0) 2.0)
            4.0 -1.0)))) -1.0)))
```

```
(lambda (par stack-id) (* (begin (define sym0 0.0)
    (exp (safe-uc -1.0 (safe-sqrt (safe-uc
    (safe-div (safe-uc 0.0 (safe-uc 0.0 3.14159))
      par) (+ 1.0 (safe-uc (begin (define sym2
      (lambda (var1 var2 stack-id) (dec var2)))
      (sym2 (safe-uc -2.0 (* (safe-uc 0.0 (begin
      (define sym4 (safe-uc sym0 (* (+ (begin
      (define sym5 (lambda (var1 var2 stack-id)
      (safe-div (+ (safe-log (dec 0.0)) -1.0) var1)))
      (sym5 (exp par) 1.0 0)) 1.0) 1.0))) (if (< (safe-uc
      par sym4) 1.0) sym0 (safe-uc 0.0 -1.0)))) sym0))
      (safe-div sym0 (exp 1.0)) 0)) 0.0)))))))) par))
```

- Модель обучения вероятностным программам можно представить в виде вероятностной программы
- Нетривиальные вероятностные программы удается вывести, но только достаточно простые

https://arxiv.org/pdf/1407.2646v1.pdf

Универсальная индукция

- Универсальные приоры $P(\mu) = 2^{-l(\mu)}$

μ – программы для универсальной машины Тьюринга

- Условное распределение $P(x|\mu) = \begin{cases} 1, & if \quad U(\mu) = x* \\ 0, & otherwise \end{cases}$

  x – произвольные строки

- MAP $\mu^* = \arg\max_{\mu} P(\mu|x) = \arg\min_{\mu: U(\mu)=x*} l(\mu|x)$

- Маргинальная вероятность $M_U(x) = \sum_{\mu: U(\mu)=x*} 2^{-l(\mu)}$

- Предсказание $M_U(y|x) = M_U(xy) / M_U(x)$

- Теорема о бесплатных обедах?

Сходимость! $\mathrm{E}_Q \left[\sum_{i=1}^{n} \left(Q(x_{i+1} = 1 \mid x_{1:i}) - P_U(x_{i+1} = 1 \mid x_{1:i}) \right)^2 \right] \le \frac{\ln 2}{2} K_U(Q)$

Универсальная индукция и ВП

```
var universal_induction = function() {
    var expr = generate_program()
    var output = interpret(expr)
    condition(output == [0, 1, 0, 1, 0, 1, 0, 1])
    return expr
}
```

Универсальная индукция и ВП

```
var universal_induction = function() {
    var expr = generate_program()
    var output = interpret(expr)
    condition(output == [0, 1, 0, 1, 0, 1, 0, 1])
    return expr
}
```

- Например, комбинаторная логика

```
var generate_program = function() {
    if(flip(0.4)) return [gen(), gen()]
    return categorical([1, 1, 1, 1], ['K', 'S', 0, 1])
}
var interpret = function(expr) { ... }
```

Универсальная индукция и ВП

```
var universal_induction = function() {
    var expr = generate_program()
    var output = interpret(expr)
    condition(output == [0, 1, 0, 1, 0, 1, 0, 1])
    return expr
}
```

- Например, комбинаторная логика

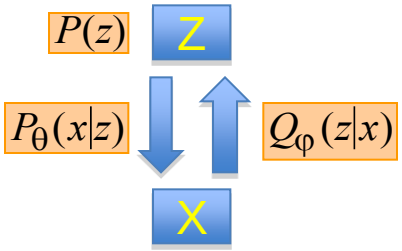
```
var generate_program = function() {
    if(flip(0.4)) return [gen(), gen()]
    return categorical([1, 1, 1, 1], ['K', 'S', 0, 1])
}
var interpret = function(expr) { ... }
```

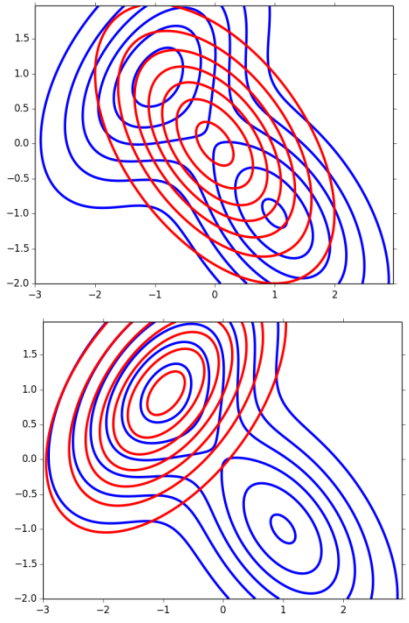
➔ Неэффективно, как и любая другая прямая реализация универсальной индукции

Вариационные приближения

- Вычислять апостериорное распределение всегда сложно

$$P(\mathbf{z}_i \mid \mathbf{x}_i, \theta) = \frac{P(\mathbf{x}_i, \mathbf{z}_i \mid \theta)}{P(\mathbf{x}_i \mid \theta)} = \frac{P(\mathbf{x}_i, \mathbf{z}_i \mid \theta)}{\int P(\mathbf{x}_i, \mathbf{z} \mid \theta) d\mathbf{z}}$$





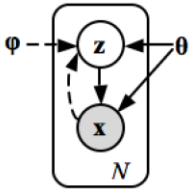
- Будем его аппроксимировать чем-то легко вычислимым $Q(\mathbf{z}|\mathbf{x}_i, \varphi)$
- Критерий: дивергенция Кульбака-Лейблера / вариационная нижняя граница маргинального правдоподобия

$$\log P(\mathbf{x}_i|\theta) = D_{KL}\left(Q(\mathbf{z}|\mathbf{x}_i,\varphi)\|P(\mathbf{z}|\mathbf{x}_i,\theta)\right) + L(\theta,\varphi|\mathbf{x}_i)$$

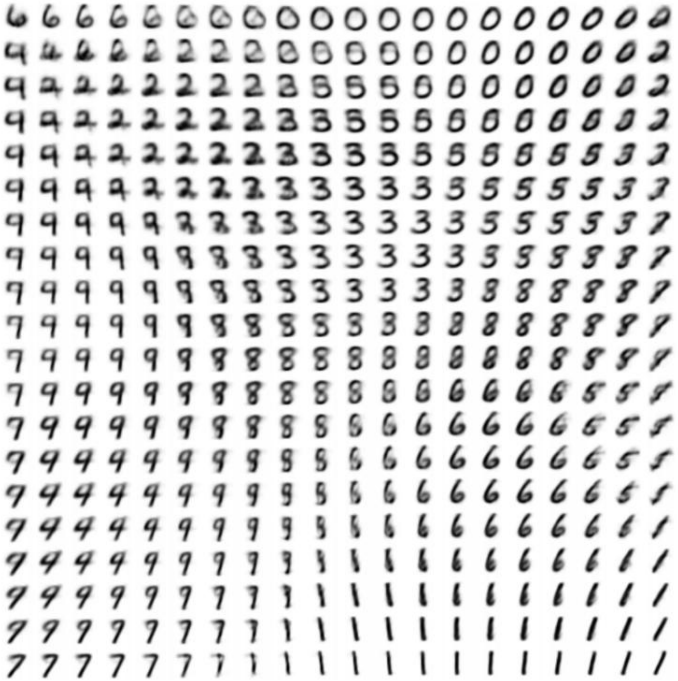
$$D_{KL}(Q\|P) = -\int Q(\mathbf{z}|\mathbf{x}_i,\varphi)\left[\log\frac{P(\mathbf{z}|\mathbf{x}_i,\theta)}{Q(\mathbf{z}|\mathbf{x}_i,\varphi)}\right]d\mathbf{z} \qquad L(\theta,\varphi|\mathbf{x}_i) = \int Q(\mathbf{z}|\mathbf{x}_i,\varphi)\left[\log\frac{P(\mathbf{x}_i,\mathbf{z}|\theta)}{Q(\mathbf{z}|\mathbf{x}_i,\varphi)}\right]d\mathbf{z}$$

- Опять эти формулы ☹

Вариационные автоэнкодеры



- Давайте учить генеративную модель и ее вариационное приближение одновременно
- Давайте эти распределения представлять как глубокие сети
- Добавим эвристик



https://arxiv.org/abs/1312.6114

Генеративные конкурирующие сети

• Качество генеративной модели оценивается напрямую дискриминантной моделью, которая пытается отличить образы, порожденные генеративной моделью, от реальных и которая учится вместе с генеративной моделью

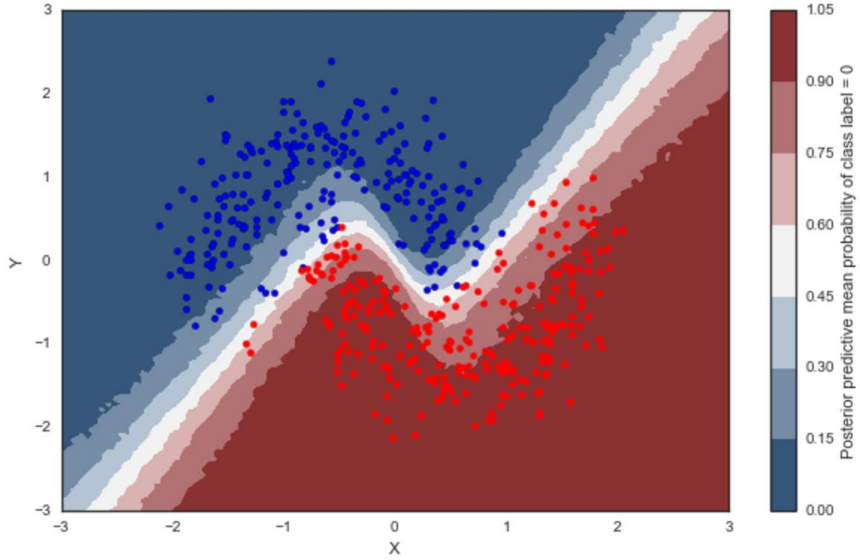
• Никакого семплирования; обучение градиентным спуском;

• Множество приложений, особенно, в генерации изображений

• DCGAN, WGAN, LSGAN, … BiGAN, InfoGAN, Bayesian GAN

Нейробайесовский подход

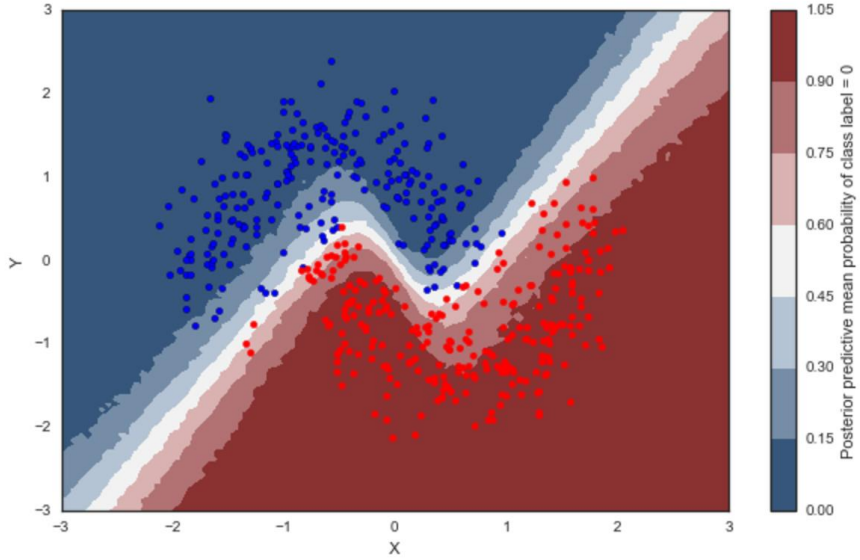
- Представить распределения вероятностей в виде глубоких сетей
- Ввести распределение вероятностей над весами связей
- Использование приоров
- Оценка неопределенности
- Градиентный спуск + техники байесовского вывода
- Эвристики DL как приближения к Байесовскому выводу



Изображение из: http://twiecki.github.io/blog/2016/06/01/bayesian-deep-learning/

Нейробайесовский подход

- Представить распределения вероятностей в виде глубоких сетей
- Ввести распределение вероятностей над весами связей
- Использование приоров
- Оценка неопределенности
- Градиентный спуск + техники байесовского вывода



- Эвристики DL как приближения к Байесовскому выводу
- Но писать формулы для распределений вероятностей нам не менее лень, чем рассчитывать градиенты вручную!

Изображение из: http://twiecki.github.io/blog/2016/06/01/bayesian-deep-learning/

Вероятностное программирование: нейробайес

```
import edward as ed
from edward.models import Normal
def neural_network(X):
    h = tf.tanh(tf.matmul(X, W_0) + b_0)
    h = tf.tanh(tf.matmul(h, W_1) + b_1)
    h = tf.matmul(h, W_2) + b_2
    return tf.reshape(h, [-1])
W_0 = Normal(loc=tf.zeros([D, 10]), scale=tf.ones([D, 10]))
...
b_2 = Normal(loc=tf.zeros(1), scale=tf.ones(1))
X = tf.placeholder(tf.float32, [N, D])
y = Normal(loc=neural_network(X), scale=0.1 * tf.ones(N))
qW_0 = Normal(loc=tf.Variable(tf.random_normal([D, 10])),
     scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, 10]))))
...
inference = ed.KLqp({W_0: qW_0,… W_2: qW_2, b_2: qb_2}, data={X: X_train, y: y_train})
```

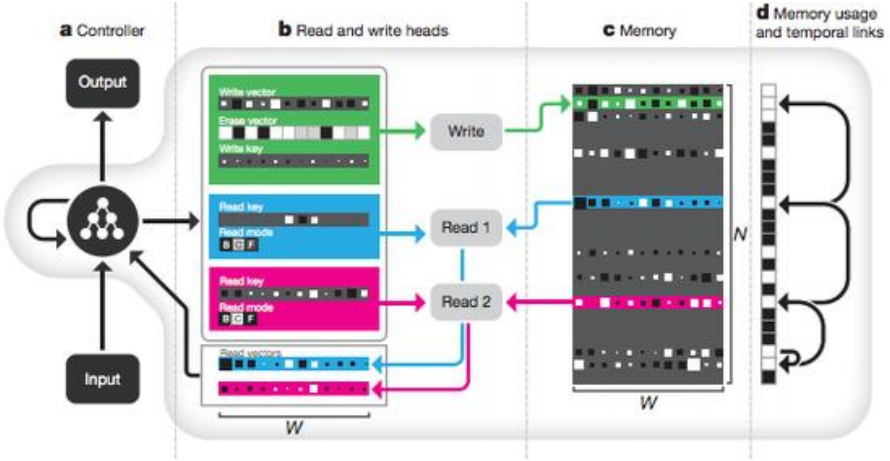
Вероятностное программирование: нейробайес

```
import edward as ed
from edward.models import Normal
def neural_network(X):
    h = tf.tanh(tf.matmul(X, W_0) + b_0)
    h = tf.tanh(tf.matmul(h, W_1) + b_1)
    h = tf.matmul(h, W_2) + b_2
    return tf.reshape(h, [-1])
W_0 = Normal(loc=tf.zeros([D, 10]), scale=tf.ones([D, 10]))
...
b_2 = Normal(loc=tf.zeros(1), scale=tf.ones(1))
X = tf.placeholder(tf.float32, [N, D])
y = Normal(loc=neural_network(X), scale=0.1 * tf.ones(N))
qW_0 = Normal(loc=tf.Variable(tf.random_normal([D, 10])),
     scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, 10]))))
...
inference = ed.KLqp({W_0: qW_0,… W_2: qW_2, b_2: qb_2}, data={X: X_train, y: y_train})
```

- Удобно, но немножко не универсально

К универсальности в глубоком обучении

- RNN вместо конечных автоматов
- Внешняя память с мягкой адресацией
- Полностью дифференцируемые алгоритмы



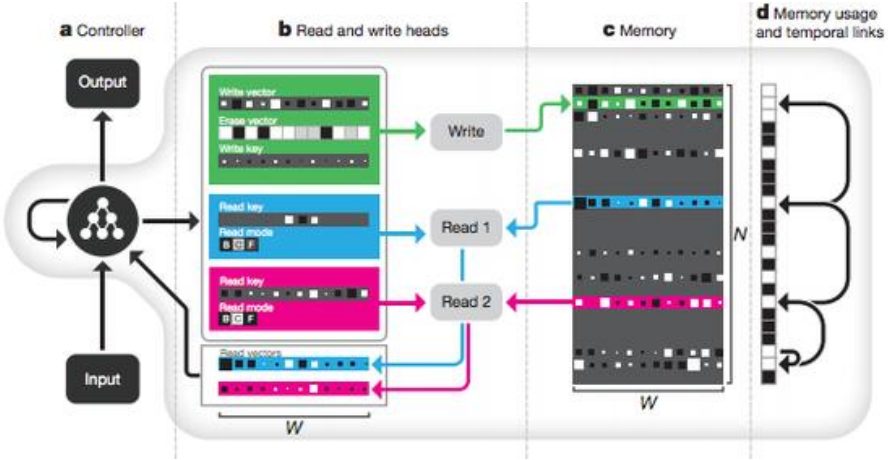
- Нейронная машина Тьюринга, нейронная GPU, нейронный программист-интерпретатор, дифференцируемых Forth и т.д.

+ Memory augmented NNs

Изображение из: https://deepmind.com/blog/differentiable-neural-computers/

К универсальности в глубоком обучении

- RNN вместо конечных автоматов
- Внешняя память с мягкой адресацией
- Полностью дифференцируемые алгоритмы



- Нейронная машина Тьюринга, нейронная GPU, нейронный программист-интерпретатор, дифференцируемых Forth и т.д.

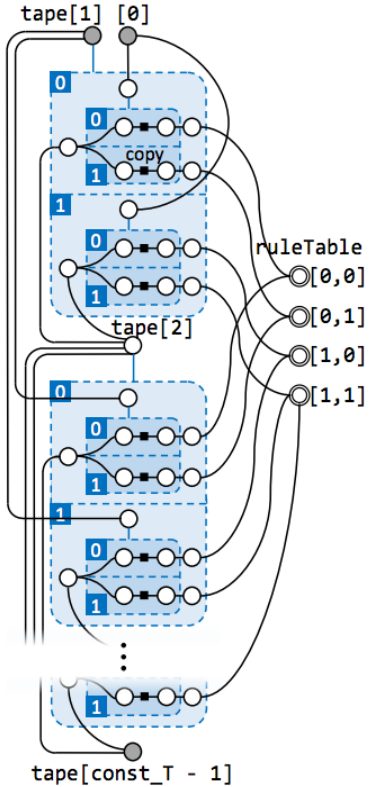
+ Memory augmented NNs

=> Как-то работают, но не панацея

Изображение из: https://deepmind.com/blog/differentiable-neural-computers/

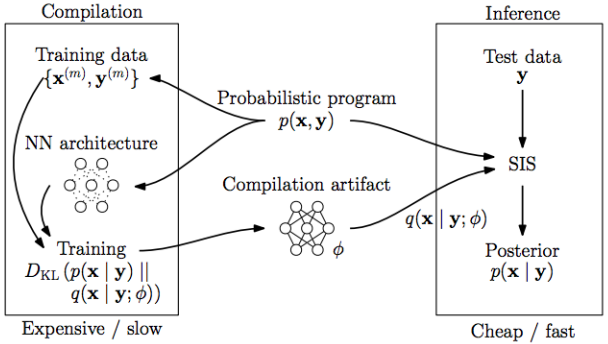
Сравнение бэкэндов при обучении алгоритмам

```
const_T = 5

###########################################################
#  Source code parametrisation                            #
###########################################################
ruleTable = Param(2)[2, 2]

###########################################################
#  Interpreter model                                      #
###########################################################
tape = Var(2)[const_T]

#__IMPORT_OBSERVED_INPUTS__
for t in range(1, const_T - 1):
    with tape[t] as x1:
        with tape[t - 1] as x0:
            tape[t + 1].set_to(ruleTable[x0, x1])
#__IMPORT_OBSERVED_OUTPUTS__
```



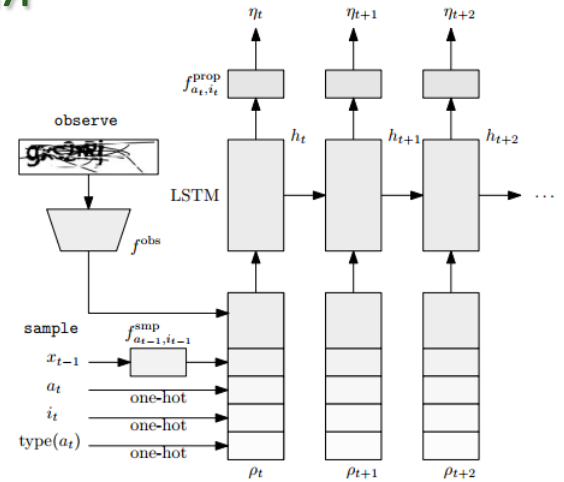
TerpreT: https://arxiv.org/abs/1608.04428

Вариационное приближение в ВП



- Генеративная модель в форме вероятностной программы «компилируется» в сеть глубокого обучения

```
procedure CAPTCHA
    ν ~ p(ν)
    κ ~ p(κ)
Generate letters:
    Λ ← {}
    for i = 1, ..., ν do
        λ ~ p(λ)
        Λ ← append(Λ, λ)
Render:
    γ ← render(Λ, κ)
    π ~ p(π)
    γ ← noise(γ, π)
    return γ
```



https://arxiv.org/abs/1610.09900

Специализация программ

- Пусть $p_L(x,y)$ – некоторая программа от двух аргументов на языке L
- Специализатор $spec_R$ – это такая программа (на языке R), которая получает на вход p_L и x_0, так что

$$(\forall y) spec_R(p_L, x_0)(y) = p_L(x_0, y)$$

- $spec_R(p_L,x_0)$ – результат глубокой трансформации p_L, которая может быть гораздо эффективнее исходной p_L

Специализация программ

- Пусть $p_L(x,y)$ – некоторая программа от двух аргументов на языке L
- Специализатор $spec_R$ – это такая программа (на языке R), которая получает на вход p_L и x_0, так что

$$(\forall y) spec_R(p_L, x_0)(y) = p_L(x_0, y)$$

- $spec_R(p_L,x_0)$ – результат глубокой трансформации p_L, которая может быть гораздо эффективнее исходной p_L

Проекции Футамуры-Турчина

$$(\forall x) spec_R(intL, p_L)(x) = intL(p_L, x)$$

Специализация программ

- Пусть $p_L(x,y)$ – некоторая программа от двух аргументов на языке L
- Специализатор $spec_R$ – это такая программа (на языке R), которая получает на вход p_L и x_0, так что

$$(\forall y) spec_R(p_L, x_0)(y) = p_L(x_0, y)$$

- $spec_R(p_L,x_0)$ – результат глубокой трансформации p_L, которая может быть гораздо эффективнее исходной p_L

Проекции Футамуры-Турчина

$$(\forall x) spec_R(intL, p_L)(x) = intL(p_L, x)$$

$$(\forall p_L, x) spec_R(spec_R, intL)(p_L)(x) = intL(p_L, x)$$

Специализация программ

- Пусть $p_L(x,y)$ – некоторая программа от двух аргументов на языке L
- Специализатор $spec_R$ – это такая программа (на языке R), которая получает на вход p_L и x_0, так что

$$(\forall y) spec_R(p_L, x_0)(y) = p_L(x_0, y)$$

- $spec_R(p_L,x_0)$ – результат глубокой трансформации p_L, которая может быть гораздо эффективнее исходной p_L

Проекции Футамуры-Турчина

$$(\forall x) spec_R(intL, p_L)(x) = intL(p_L, x)$$

$$(\forall p_L, x) spec_R(spec_R, intL)(p_L)(x) = intL(p_L, x)$$

$$(\forall intL) spec_R(spec_R, spec_R)(intL) = comp_{L \to R}$$

Еще одна проекция

- Пусть p_L – некоторая генеративная модель (программа на вероятностном языке L)
- Пусть *inferL* – это процедура вывода (интерпретатор языка L)

$$(\forall x) spec_R(inferL, p_L)(x) = inferL(p_L, x)$$

```
var inference = function(has_dots, is_bright) {
  var pL = function() {
    var gene1 = flip()
    var gene2 = flip()
    var gene3 = flip()
    condition(has_dots == (gene1 && gene2) &&
                  is_bright == (gene1 || gene3))
    var poisonous = gene1
    return poisonous
  }
  return Infer({model: pL})
}
```

Еще одна проекция

- Пусть p_L – некоторая генеративная модель (программа на вероятностном языке L)
- Пусть *inferL* – это процедура вывода (интерпретатор языка L)

$$(\forall x) spec_R(inferL, p_L)(x) = inferL(p_L, x)$$

```
var inference = function(has_dots, is_bright) {
  var pL = function() {
    var gene1 = flip()
    var gene2 = flip()
    var gene3 = flip()
    condition(has_dots == (gene1 && gene2) &&
                  is_bright == (gene1 || gene3))
    var poisonous = gene1
    return poisonous
  }
  return Infer({model: pL})
}
```

$spec_L$(Infer, pL)

```
var inference =
 function(has_dots, is_bright) {
    if(!has_dots) {
       if(!is_bright) return false
       else return flip()
    }
    return true
}
```

Специализация вывода, или чего не хватает Байесу

- Специализация специализатора по специализатору → бесполезная абстракция
- Специализация специализатора по общей процедура вывода → искусственный интеллект
- Специализация универсальной индукции по алгоритмически неполной опорной машине → узкий метод машинного обучения
- Специализация общей процедуры вывода по генеративной модели → дискриминантная модель
- Приближенная специализация с одновременным обучением → вариационный вывод

Специализация вывода, или чего не хватает Байесу

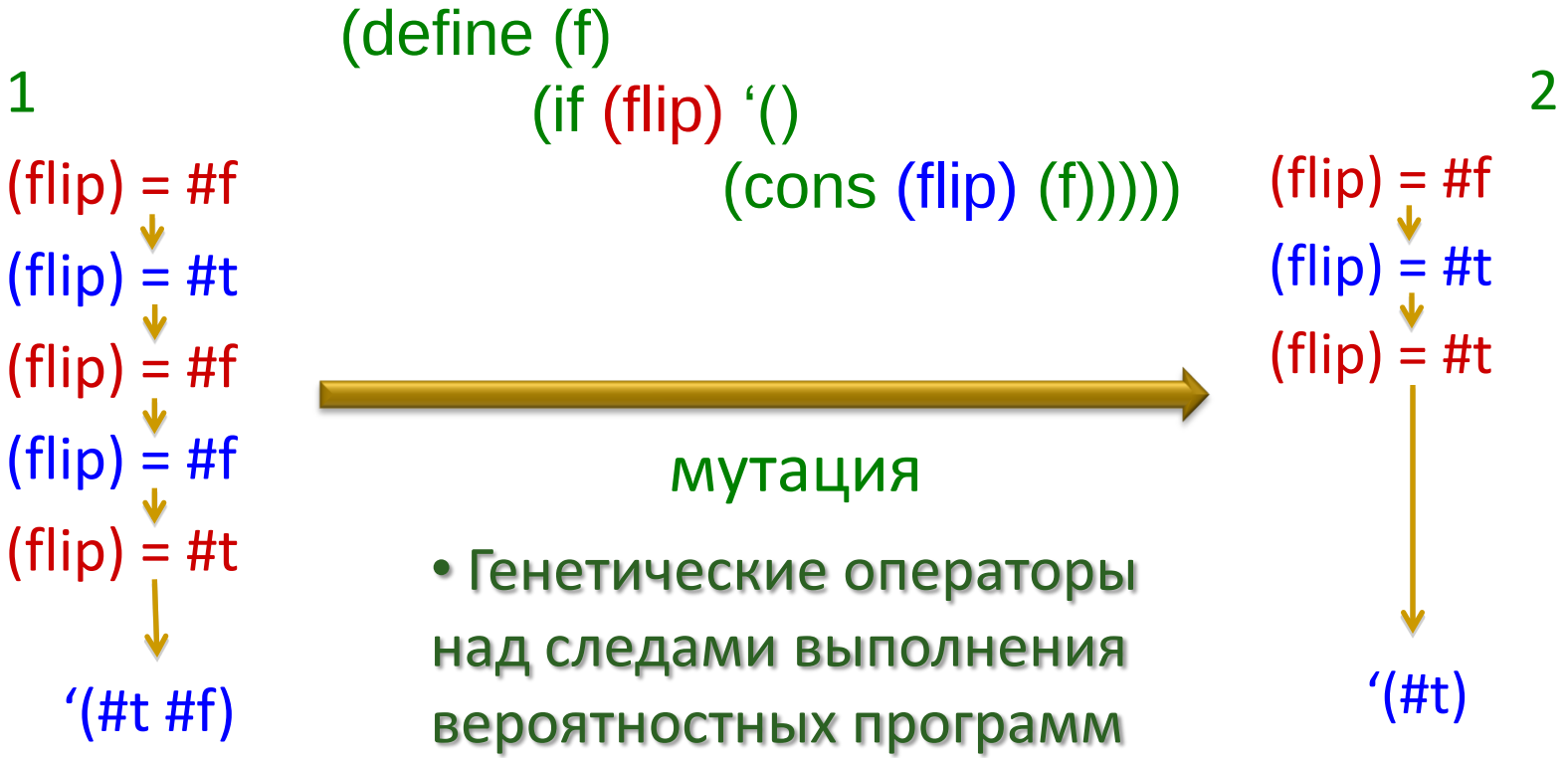
- Специализация специализатора по специализатору → бесполезная абстракция
- Специализация специализатора по общей процедура вывода → искусственный интеллект
- Специализация универсальной индукции по алгоритмически неполной опорной машине → узкий метод машинного обучения
- Специализация общей процедуры вывода по генеративной модели → дискриминантная модель
- Приближенная специализация с одновременным обучением → вариационный вывод

- Частичная специализация → ?

Не только дискриминантные модели

- Дискриминантные модели (прямое вычисление z по x) не всегда возможны, но это не значит, что ситуация безнадежна
- Имеется широкий спектр ‘частичной’ специализации общей процедуры вывода
- Слепой поиск: семплируем решение $z \sim P(z)$; генерируем наблюдения $x \sim P(x|z)$; проверяем совместимость
- Метаэвристический поиск: семплируем $z \sim P(z|z')$ или $z \sim P(z|z',z'')$, где z',z'' – предыдущие кандидаты; P не зависит от задачи
- **Специализированный поиск, управляемый данными: $z \sim Q_\phi(z|x,z',z'')$, где Q обучается для конкретной задачи**
- Дискриминантные модели: $Q_\phi(z|x,z',z'')=Q_\phi(z|x)$

ГП как машина вывода в ВП



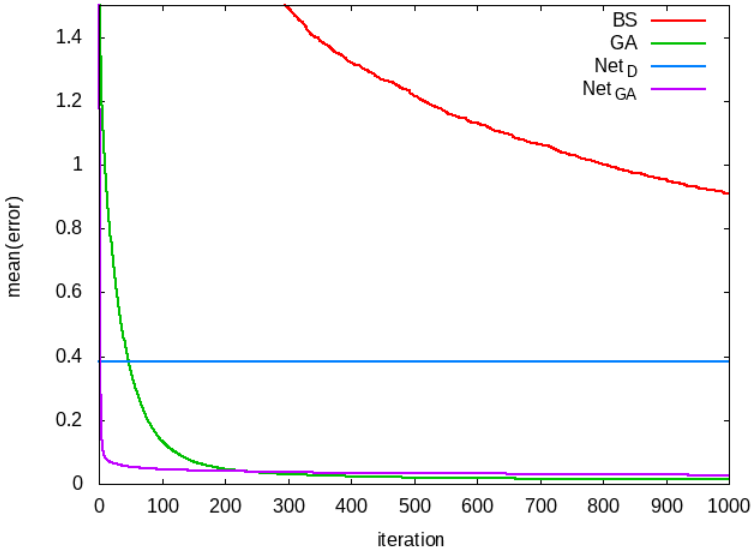
- Генетические операторы над следами выполнения вероятностных программ

Batishcheva V., Potapov A. Genetic Programming on Program Traces as an Inference Engine for Probabilistic Languages. AGI 2015, Lecture Notes in Artificial Intelligence. V. 9205. P. 14–24.

Пример эксперимента

Задача: $f(\mathbf{z}|\mathbf{A},\mathbf{b}) = |\mathbf{Az} - \mathbf{b}|^2$ $\mathbf{z}^* = \mathbf{A}^{-1}\mathbf{b}$

- Net_D – сеть прямого распространения, которая учится по A и b выдавать z*
- NetGA – сеть, которая по A, b, z', z'' учится выдавать лучший z
- GA – генетические алгоритмы с обычным скрещиванием
- BS – Brute force search

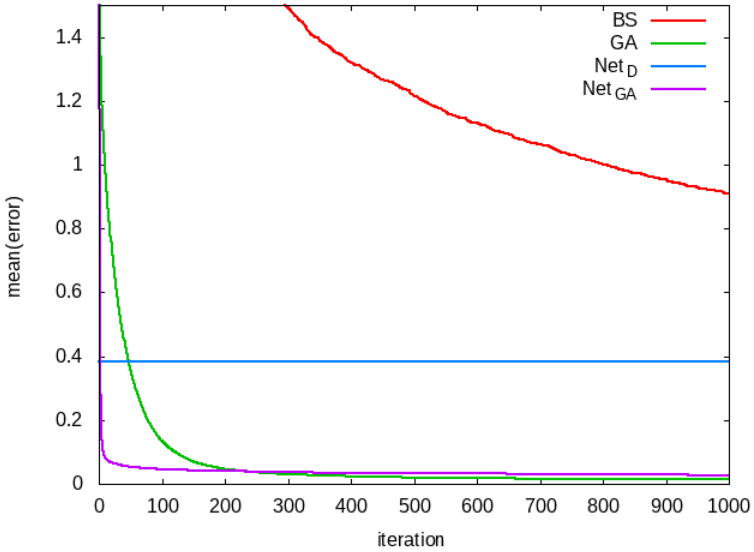


Potapov A. Rodionov S. Genetic Algorithms with DNN-Based Trainable Crossover as an Example of Partial Specialization of General Search // Proc. Artificial General Intelligence, AGI'17. P. 101-111.

Пример эксперимента

Задача: $f(\mathbf{z}|\mathbf{A},\mathbf{b}) = |\mathbf{Az} - \mathbf{b}|^2$ $\mathbf{z}^* = \mathbf{A}^{-1}\mathbf{b}$

- Net_D – сеть прямого распространения, которая учится по A и b выдавать z*
- NetGA – сеть, которая по A, b, z', z'' учится выдавать лучший z
- GA – генетические алгоритмы с обычным скрещиванием
- BS – Brute force search



Частичная специализация действительно нужна

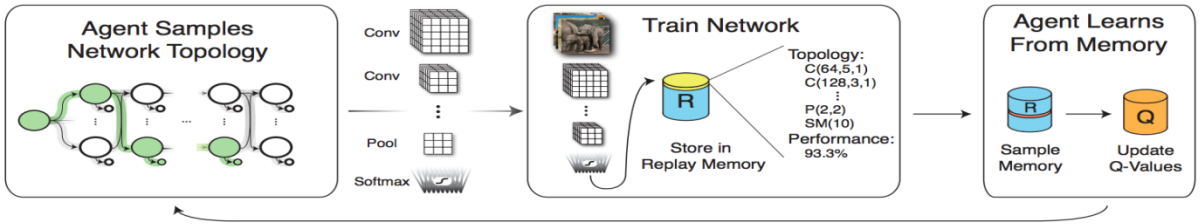
Potapov A. Rodionov S. Genetic Algorithms with DNN-Based Trainable Crossover as an Example of Partial Specialization of General Search // Proc. Artificial General Intelligence, AGI'17. P. 101-111.

Связь с метаобучением

- Рассмотрим задачу обучения нейросети:
  - это задача вывода/поиска
    - *градиентный спуск*
    - *MCMC*
    - *и т.д.*
- Можно поставить задачу на специализацию этого вывода/поиска
  - Строим мета-сеть, которая учится находить сеть, решающую задачи конкретного класса
    - Полная специализация: прямое отображение параметров задачи в веса сети
    - Частичная специализация: метаобучение

Метаобучение с глубокими сетями

- A neural network that embeds its own meta-levels
- Learning to learn using gradient descent
- Learning to learn by gradient descent by gradient descent
- Learning to reinforcement learn
- RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning
- Meta-Learning with Memory-Augmented Neural Networks
- Designing Neural Network Architectures using Reinforcement Learning
- …



https://arxiv.org/pdf/1611.02167.pdf

Обсуждение

- Глубокое обучение
  - Генеративные модели
  - Представление алгоритмов
  - Метаобучение

Обсуждение

- Глубокое обучение
- Вероятностные модели
  - Вероятностное программирование
  - Вариационные приближения
  - Специализация

Обсуждение

- Глубокое обучение
- Вероятностные модели
- Метавычисления
- Универсальная индукция
- Когнитивные архитектуры
- Инкрементная самооптимизация
- …
- Общий ИИ

Спасибо за внимание!

potapov@aideus.com