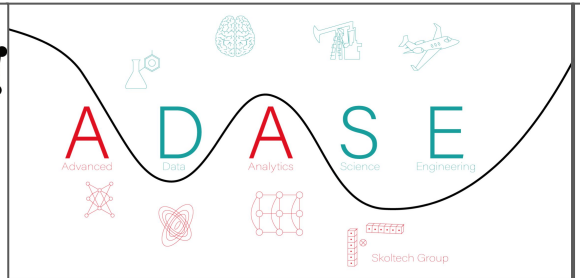
Глубокие Генеративные (Вероятностные) Модели

- Александр Нотченко
Группа ADASE, Сколтех, ИППИ

Open Data Science



Skolkovo Tech

Skolkovo Institute of Science and Technology

Мотивация

- Мир утопает в данных
- Извлечь ценность из данных можно множеством способов
- Одни способ - познать и предсказывать процессы/явления через данные, используя математику и вычисления
- Строить модели, описывающие процессы, которые можно применить к конкретным данным для решения конкретных задач

Для решения задач Data Science и Machine learning есть много методов

Дискримитативные vs Генеративные модели

Q: Есть предложение на неизвестном языке, задача определить язык

A: Генеративный подход - выучить множество языков и определить, на каком языке оно написано, потому что мы можем сгенерировать любое предложение и сравнить

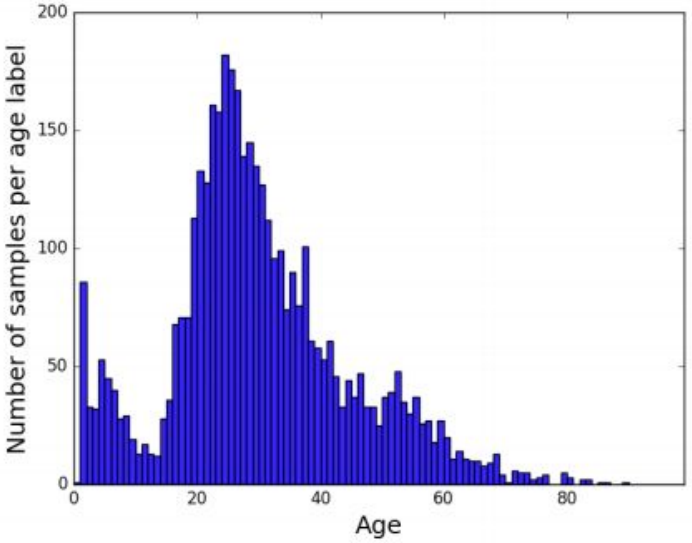
A: дискримитативный подход - определить лингвистические различия между языками без выучивания языков

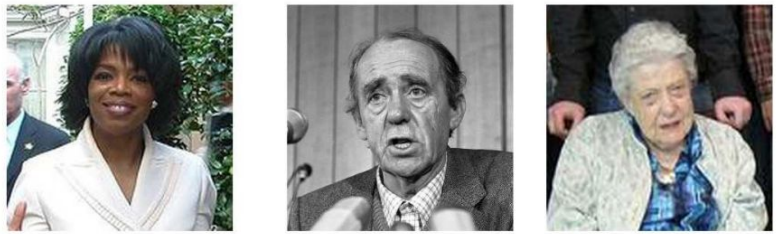
Неопределенность в моделях и данных

- Неполная информация
- Смешивания разных датасетов вместе
- Значение дисперсии для оценки предсказания алгоритмов машинного обучения
- Шум измерения
- Необходимость вычисления оценки уверенности для принятия решений

Пример: Определить по фото возраст человека

- Много изменчивости в представлении
- Много шума на изображениях
- Люди справляются средне в этой задаче



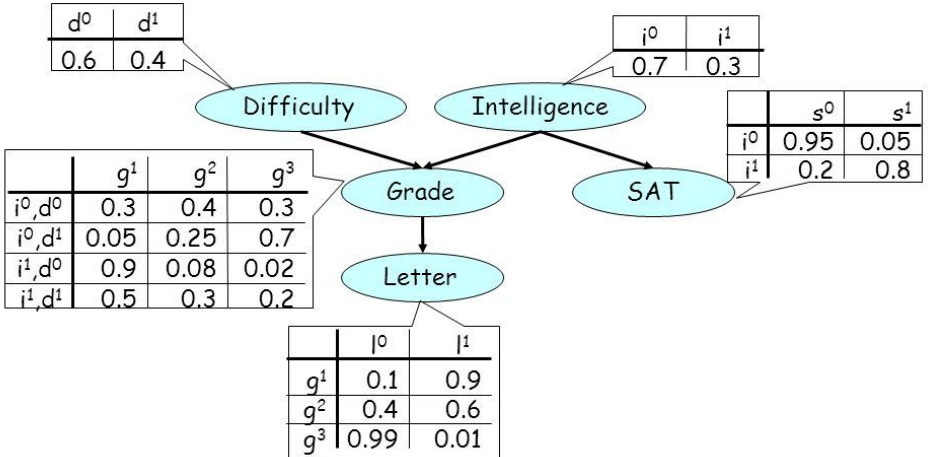


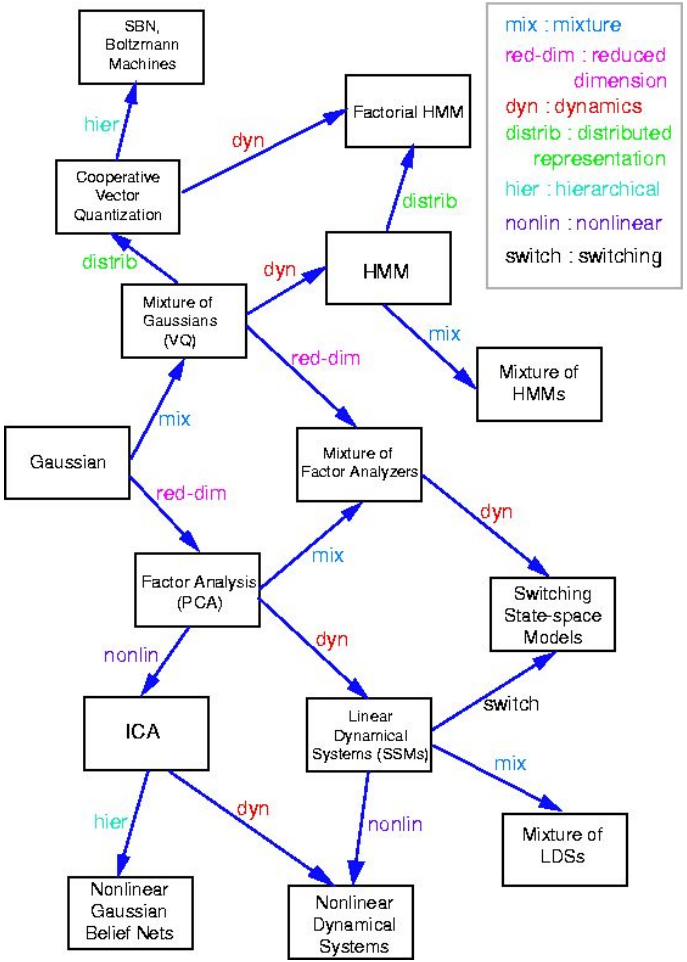
ChaLearn LAP and FotW Challenge and Workshop @ CVPR2016

Face Analysis Workshop and Challenge

Генеративные (вероятностные) модели

- Вероятность - язык коммуникации моделей
- Графические модели передают информацию через изменение неопределенности
- Вычисления с случайными переменными разных типов задают мета-априорные распределения





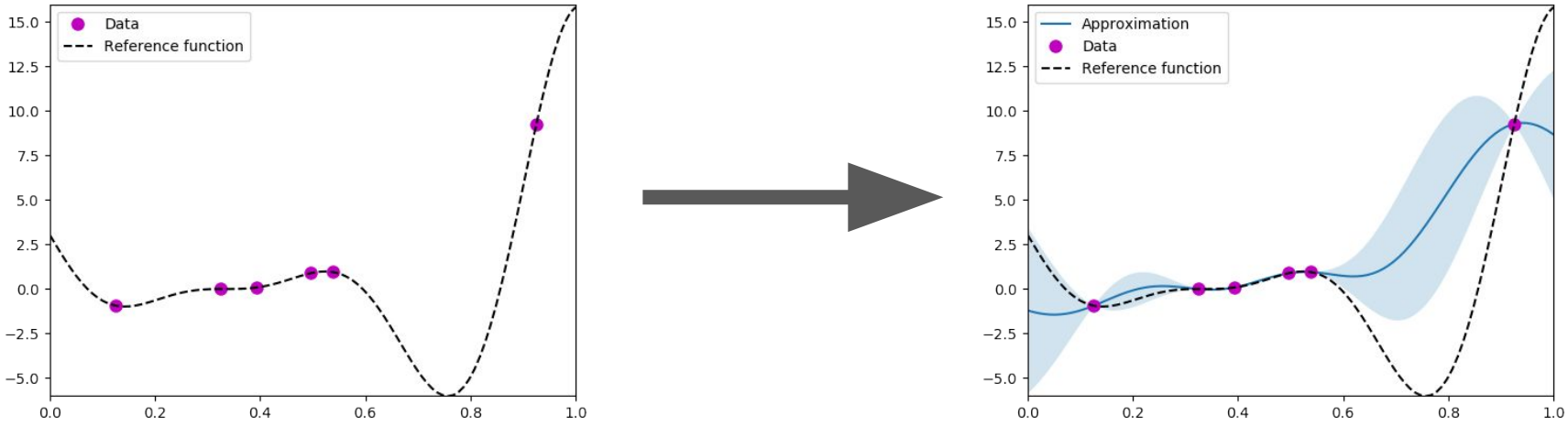
Гауссовские процессы

- Искать в пространстве случайных функций

Задача регрессии

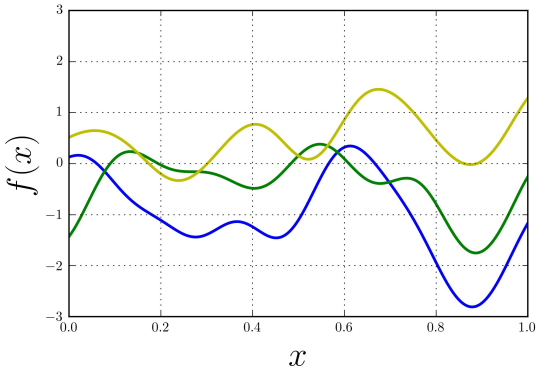
По обучающей выборке построить:

- Аппроксимацию функции
- Доверительный интервал



Гауссовские процессы

- Вероятностная модель
- Моделирует распределение функций:
  - Распределение в каждой точке $f(x) \sim Normal(m(x), k(x, x))$
  - Распределение всей функции: функцию можно приближенно представить как большой вектор ее значений в разных точках $(f(x_1), \ldots, f(x_N)) \sim Normal(\boldsymbol{m}, \boldsymbol{K})$, где $\boldsymbol{m} = (m(x_1), \ldots, m(x_N))$, $\boldsymbol{K} = |k(x_i, x_j)|$, $i, j = 1, \ldots, N$.
- Аппроксимация $f(x)$ имеет вид:



$f(x) = \Sigma\, \alpha_i\, k(x, x_i)$ - математическое ожидание значения функции в точке x при условии того, что заданы значения в точках обучающей выборки

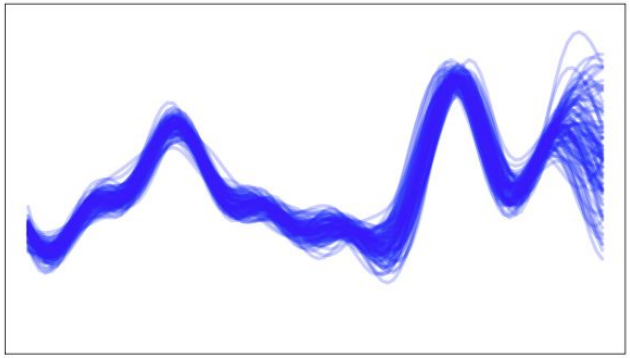
Глубокие гауссовские процессы

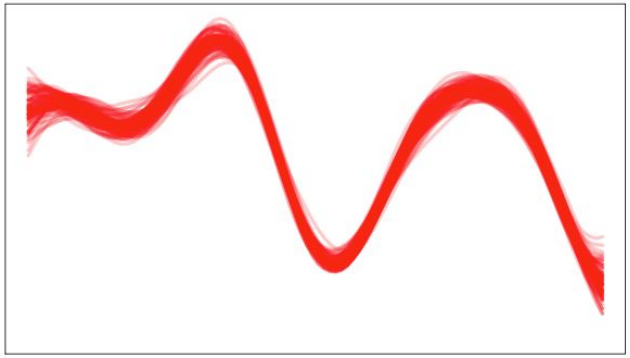
f и *g* - гауссовские процессы (случайные функции)

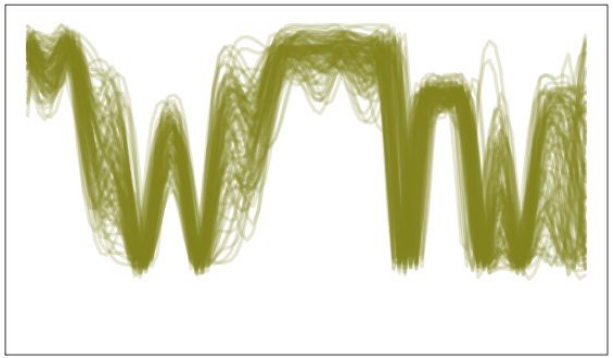
Можем ли мы получить их композицию?

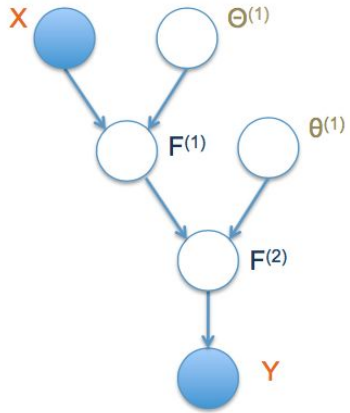
$$(f \circ g)(x)??$$

Да! Результат - Многослойная сеть с бесконечным числом нейронов









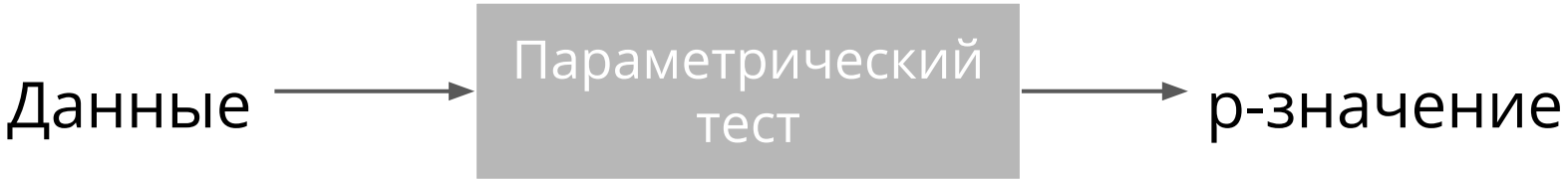
Вероятностное программирование

Новый способ описывать модели и производить оценку вероятностей

Машинное обучение как BlackBox

- Учим классификатор различать разные распределения
- Классификация плохо подходит для познания сложных процессов
- **BlackBox**: Сложно либо невозможно интерпретировать модель
- Сложно извлечь иную пользу кроме простого присваивания классов

Классическая статистика



- “Серая коробка” - **неявные предположения**:
  - Нормальность переменных (не работает для большинства случаев)
- Тяжело изменить предположения.
- p-значение не очень полезно для сложных случаев с сложными гипотезами.

Вероятностное программирование

- Конструируем **собственную модель** в **коде** (“открытый ящик”)
- Производим **автоматический вывод** с помощью MCMC
- **Байесовский подход** - вероятности на всех этапах, априорные распределения

Пример

- Оценить параметры нормального распределения по данным
- Обращаем процесс генерации данных с помощью байесова вывода

$$p(H|D) = \frac{p(D|H)p(H)}{p(D)}$$

D - Данные

H - гипотеза

p(H | D) - Апостериорное распределение

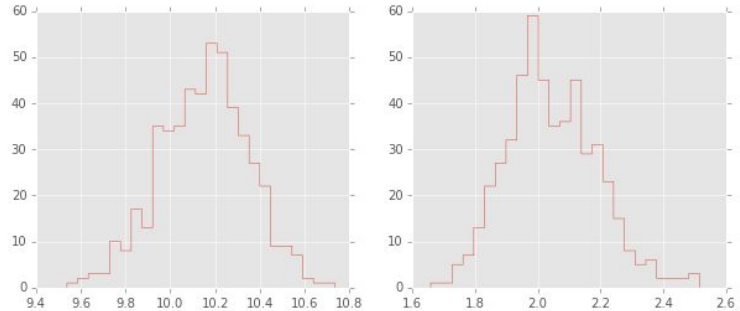
p(H) - Априорное распределение

p(D|H) - Правдоподобие

p(D) - Доказательство

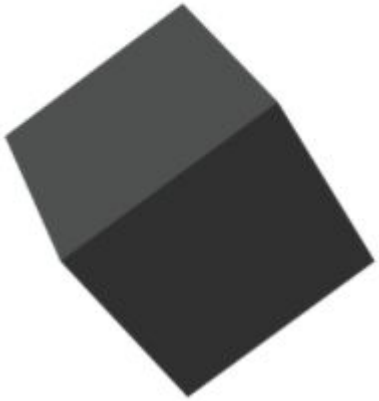
```
# generate observed data
N = 100
_mu = np.array([10])
_sigma = np.array([2])
y = np.random.normal(_mu, _sigma, N)

niter = 1000
with pm.Model() as model:
    # define priors
    mu = pm.Uniform('mu', lower=0, upper=100, shape=_mu.shape)
    sigma = pm.Uniform('sigma', lower=0, upper=10, shape=_sigma.shape)

    # define likelihood
    y_obs = pm.Normal('Y_obs', mu=mu, sd=sigma, observed=y)

    # inference
    start = pm.find_MAP()
    step = pm.Slice()
    trace = pm.sample(niter, step, start, random_seed=123, progressbar=True)
```

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1);
plt.hist(trace['mu'][-niter/2:,0], 25, histtype='step');
plt.subplot(1,2,2);
plt.hist(trace['sigma'][-niter/2:,0], 25, histtype='step');
```

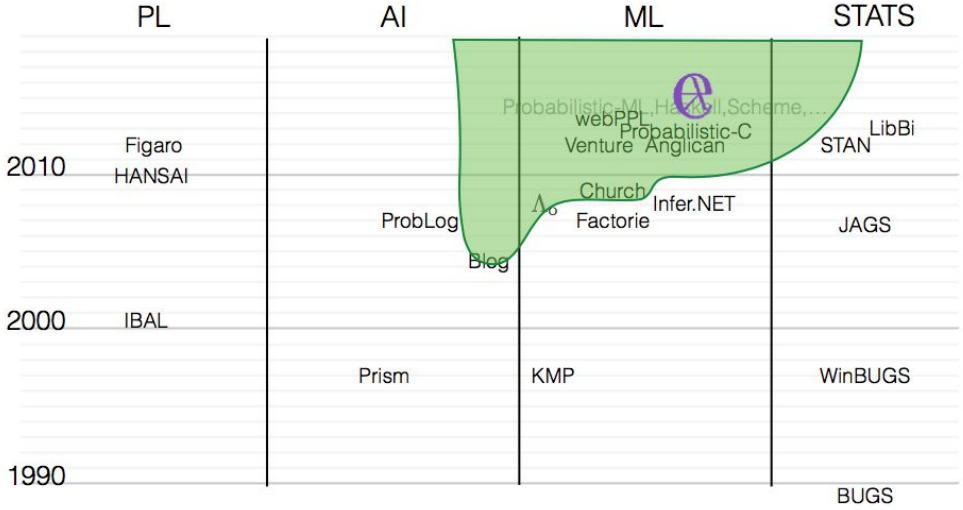


Множество проектов в этой области

Edward



Высокоуровневые языки
вероятностного программирования

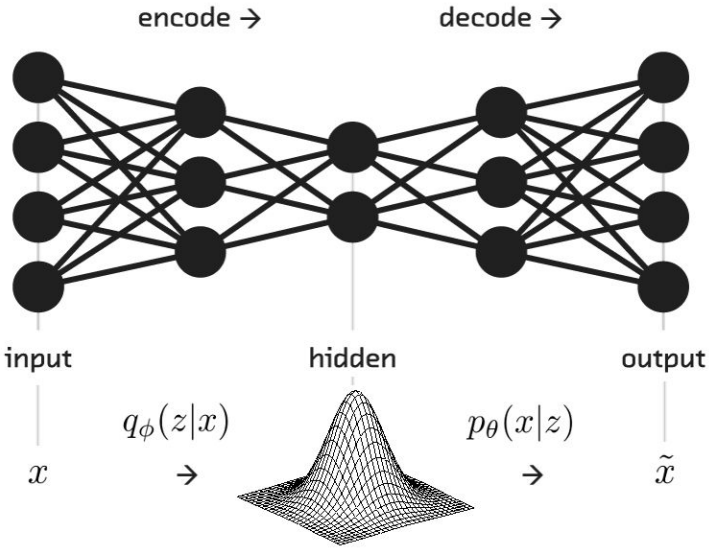


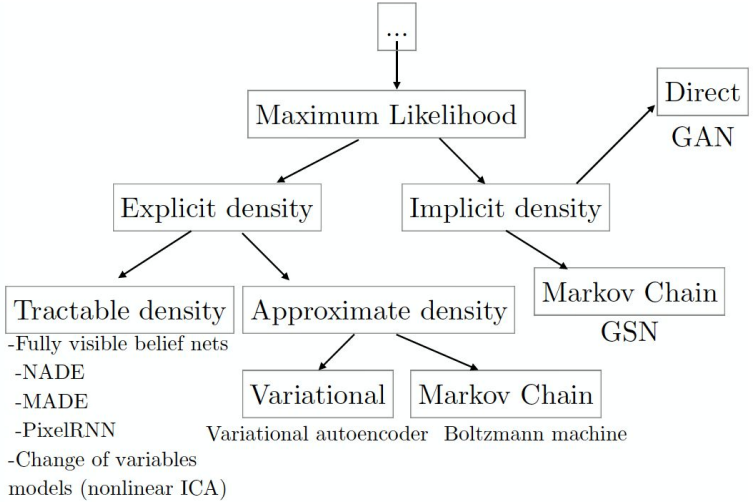
Глубокие генеративные модели

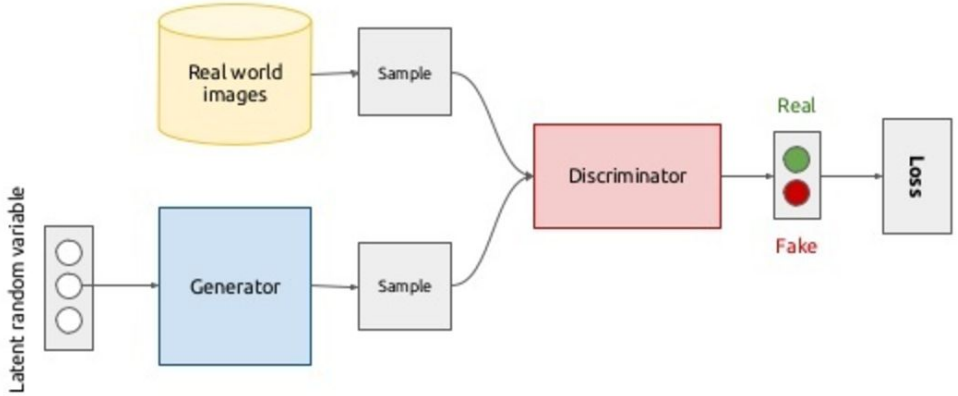
- Лучшее 2-х миров

Выученные генераторы (распределения)

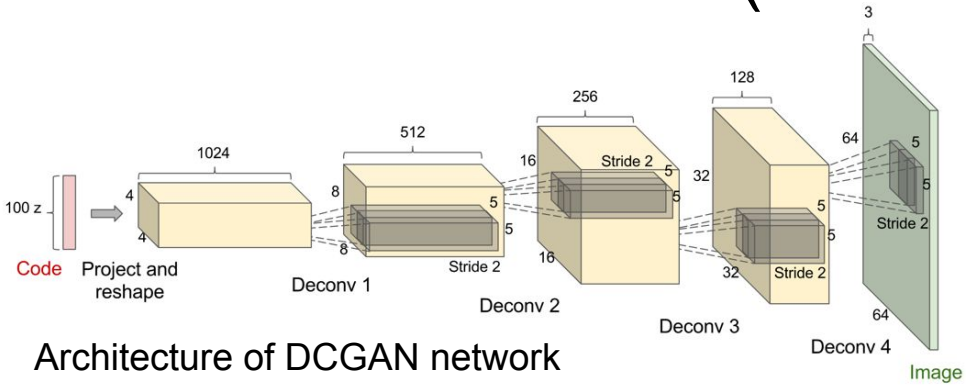
- Генеративные Адверсальные Сети (GAN)
- Вариационные авто-кодировщики (VAE)
- Авторегрессионные модели (PixelCNN, e.t.c)







Generative Networks (DCGAN)



Architecture of DCGAN network

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

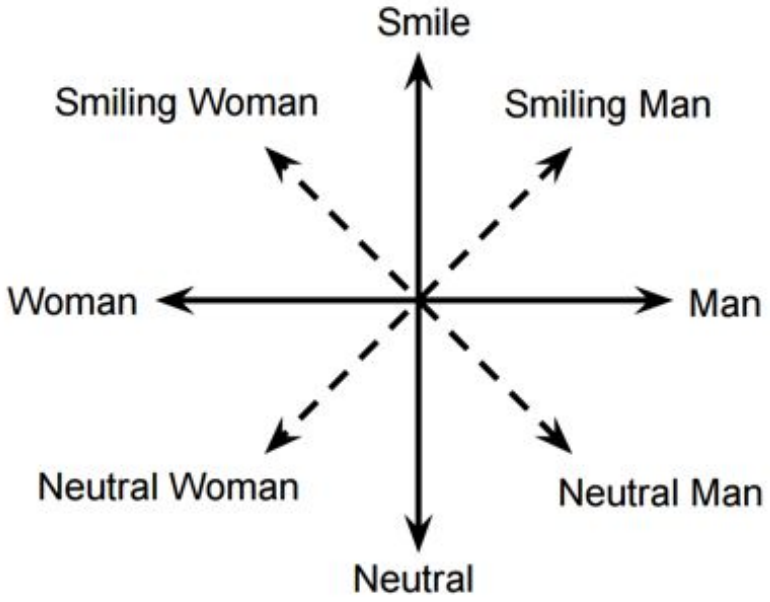
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

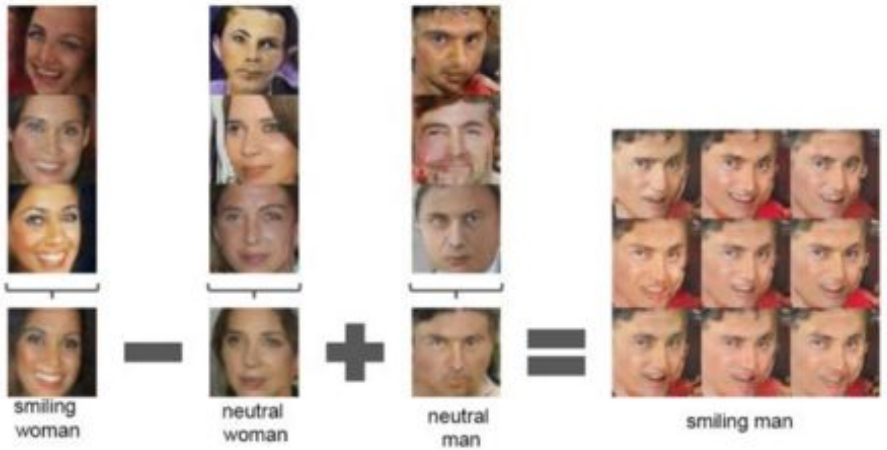
Generated samples (trained on ImageNet)

From: https://openai.com/blog/generative-models/

Interpretation of Input Parameters

- In the DCGAN paper, it is suggested that the input parameters could use a semantic structure as in the following example.





http://arxiv.org/abs/1511.06434

Преимущества и недостатки

Плюсы:

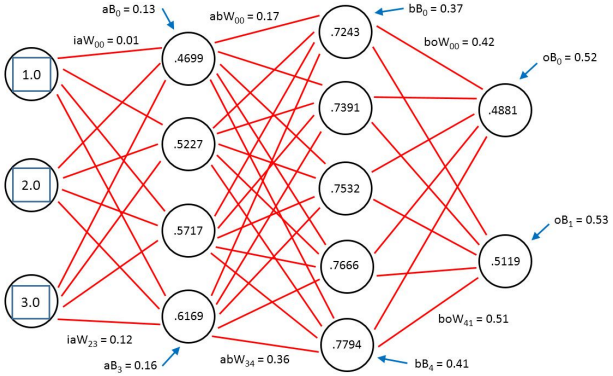
- Способны представлять очень сложные распределения:

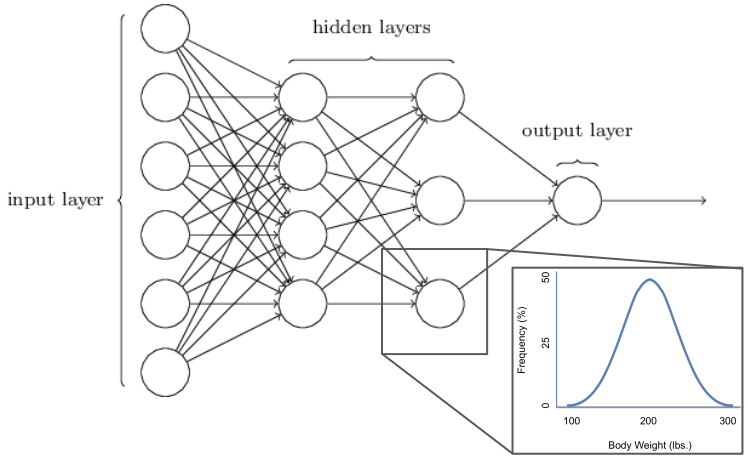
  $p(z|X)$ - нейронная сеть
- Скрытые представления - интерпретируемые
- Можно интегрировать с другими моделями

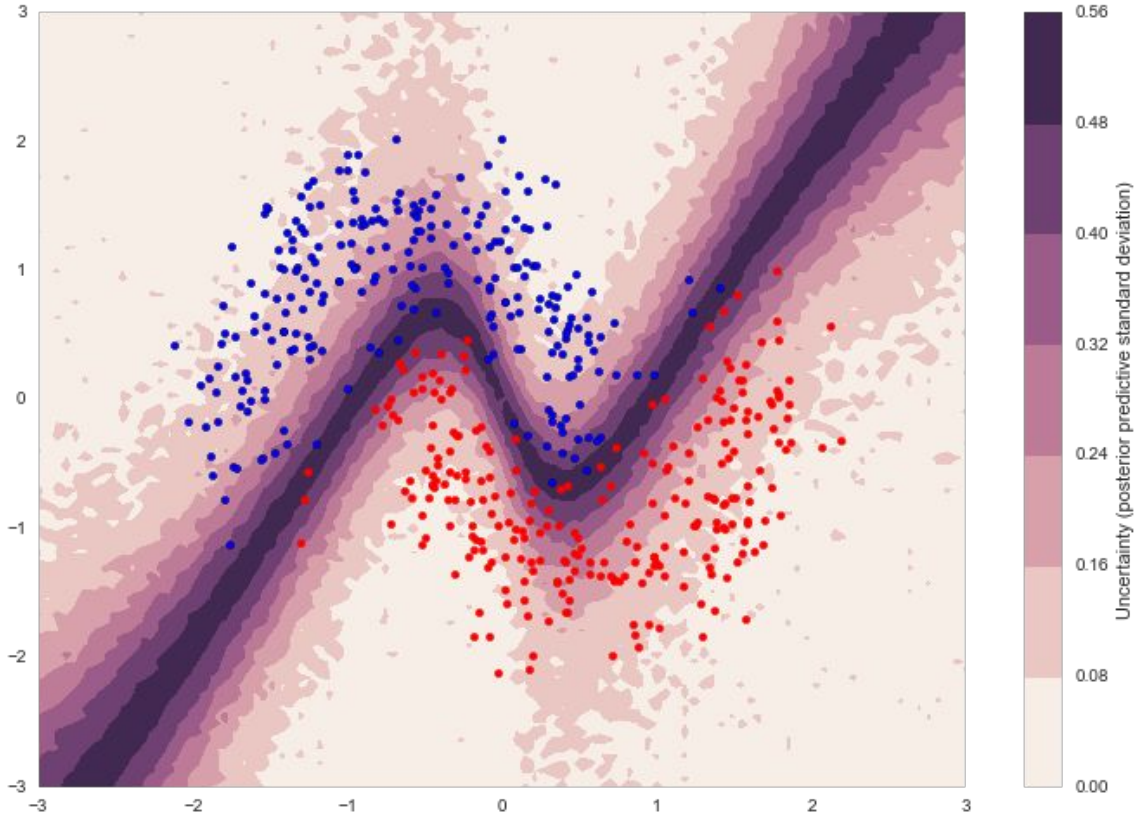
Минусы:

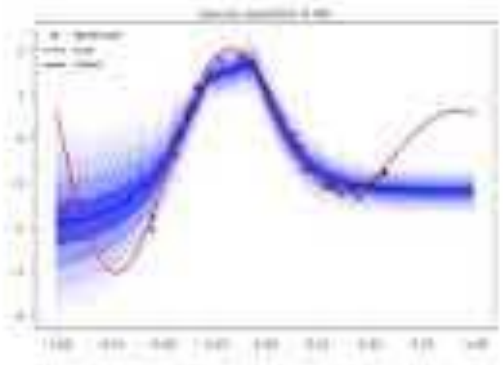
- Сложно обучать по сравнению с обычными DNN
- Требуют много данных
- Нужно знать как интегрировать в пайплайн обработки данных

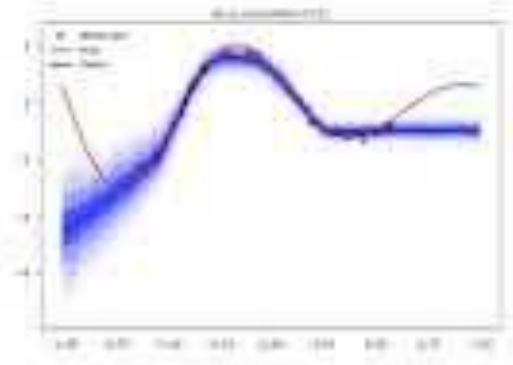
Нейро-Байесовский подход

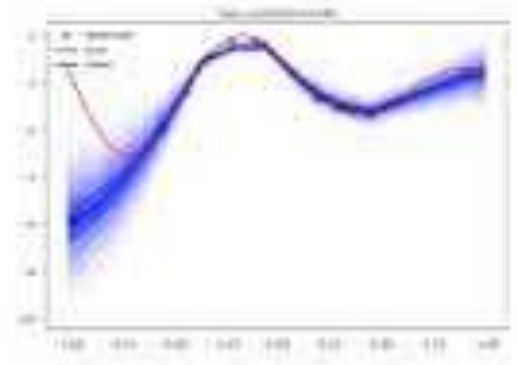


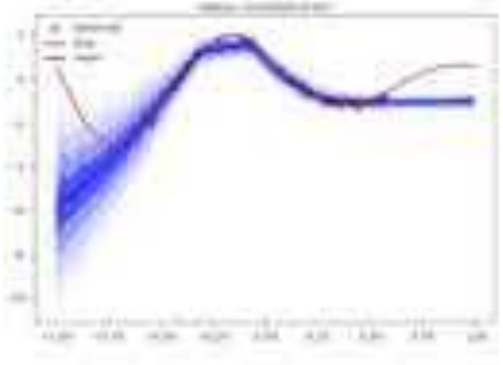


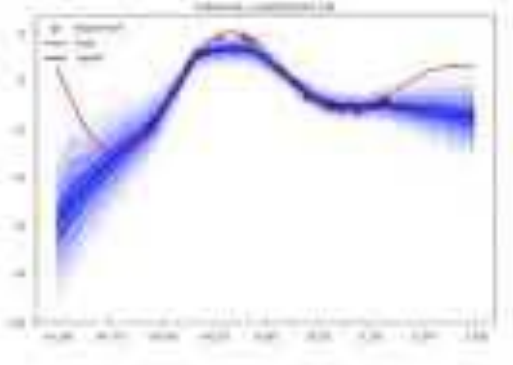


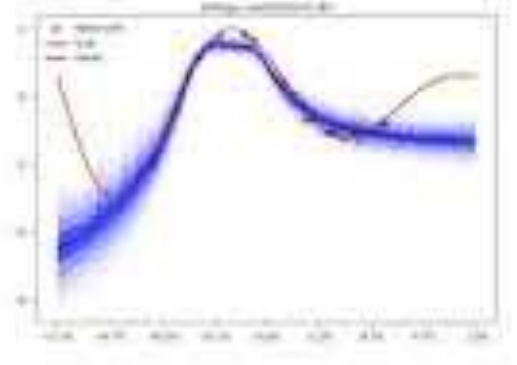






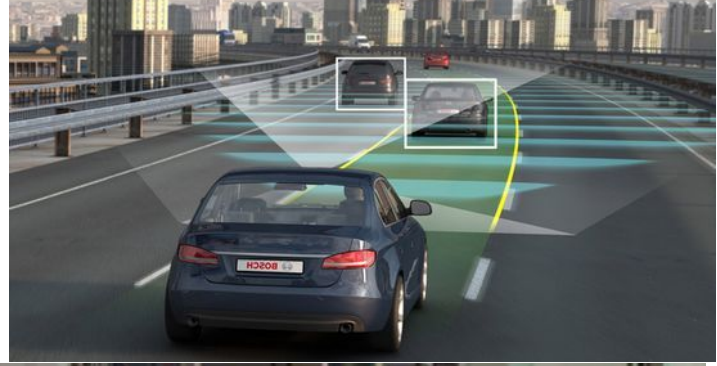


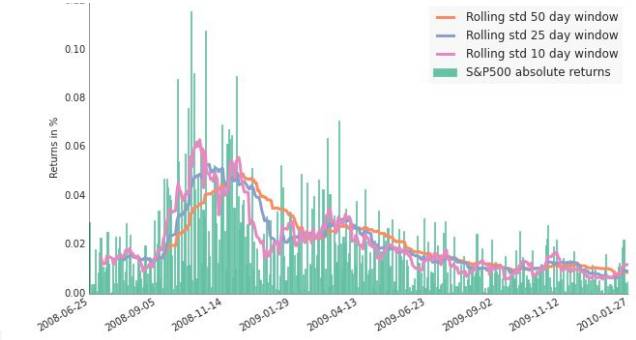


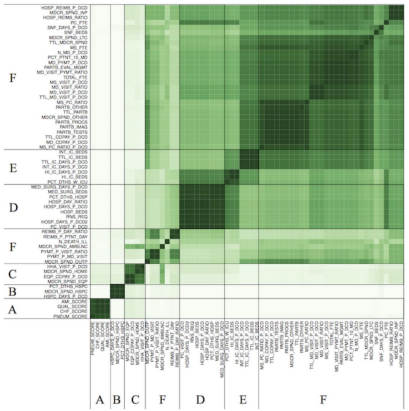


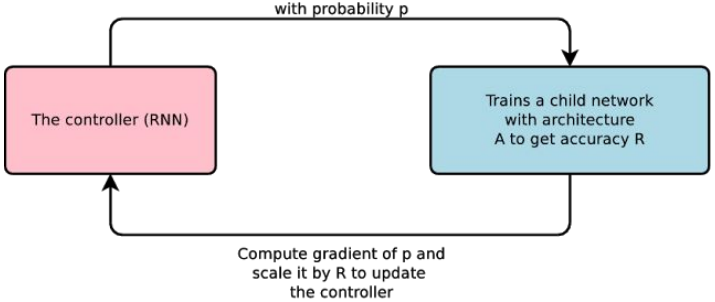
Основные применения сейчас и в будущем

- Autonomous driving (беспилотные автомобили)
- Бытовая робототехника
- Предсказания в высоко шумных средах (трейдинг, онлайн реклама)
- Банковские и Страховые услуги
- Автоматический Анализ данных (AutoML)









WIKIPEDIA
The Free Encyclopedia

ain page
ontents
eatured content

Right to explanation

From Wikipedia, the free encyclopedia

In the regulation of algorithms, particularly artificial intelligence and its subfield of machine learning, a **right to explanation** (or **right to *an* explanation**) is a right to be given an explanation for an output of the algorithm. Such rights primarily refer to individual rights to be given

Спасибо за внимание!

Сессия вопросов и ответов