

# Машинное обучение (Machine Learning)

Глубокие порождающие модели: вариационный  
автокодер, соперничающие сети (Deep Generative  
Learning: VAE, GAN)

Уткин Л.В.

Санкт-Петербургский политехнический университет Петра Великого



# Содержание

- 1 Порождающие и разделяющие модели
- 2 Вариационный автокодер (VAE)
- 3 Соперничающие сети
- 4 Соперничающие автокодеры

Презентация является компиляцией и заимствованием материалов из замечательных презентаций и материалов по машинному обучению:

*Oliver Duerr, Carl Doersch, Bohyung Han, Eric Jang*

# Порождающие модели

# Мотивация

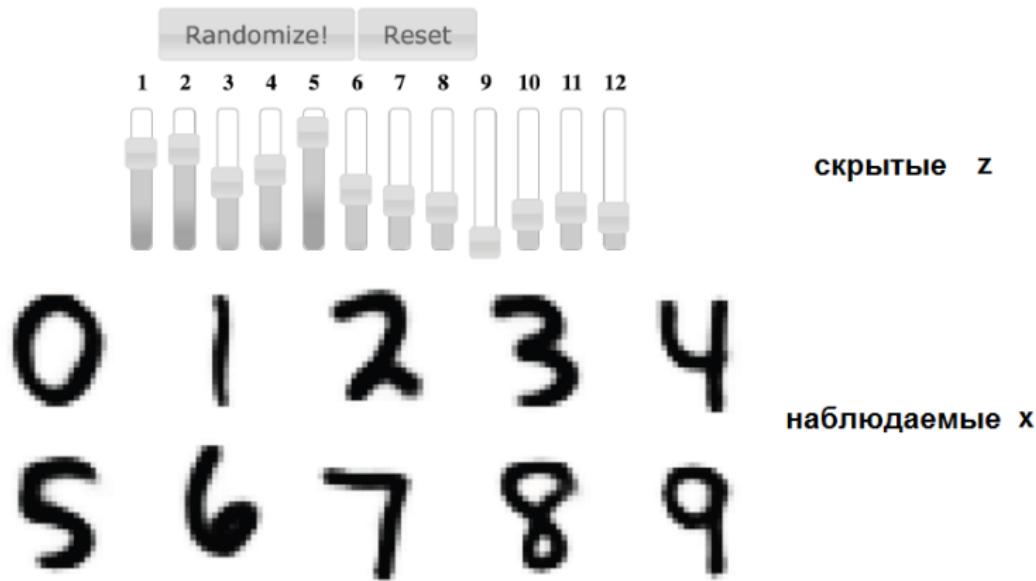
<https://www.youtube.com/watch?v=XNZIN7Jh3Sg>



← Это не часть  
обучающей  
выборки

# Мотивация

[http://www.dpkingma.com/sgvb\\_mnist\\_demo/demo.html](http://www.dpkingma.com/sgvb_mnist_demo/demo.html)



# Порождающие и разделяющие модели

- ① **Разделяющие модели (дискриптивные):** Оценка параметров на основе максимизации правдоподобия обучающей выборки по отношению к вероятности класса, а затем классификатор относит объект к его наиболее вероятному классу.
- ② **Порождающие модели:** максимизация правдоподобия совместного распределения объектов и классов, а затем использование формулы Байеса для нахождения вероятности отношения объекта к классу.

# Порождающие и разделяющие модели

- Пусть  $(x, y)$  - входные данные
- Результат обучения порождающей модели - совместное распределение вероятностей  $p(x, y)$
- Результат обучения разделяющей модели - условное распределение  $p(y|x)$ .
- Рассмотрим 4 точки данных:  
 $(x, y) \rightarrow \{(0, 0), (0, 0), (1, 0), (1, 1)\}$

$p(x, y)$		
	$y = 0$	$y = 1$
$x = 0$	1/2	0
$x = 1$	1/4	1/4

$p(x y)$		
	$y = 0$	$y = 1$
$x = 0$	1	0
$x = 1$	1/2	1/2

# Порождающие модели

*Порождающая модель позволяет оценить совместное распределение вероятностей  $p(\mathbf{x}, \mathbf{y})$ . Это означает, что можно сгенерировать  $\mathbf{x}, \mathbf{y}$  в соответствии с  $p(\mathbf{x}, \mathbf{y})$ .*

# Порождающие и разделяющие модели

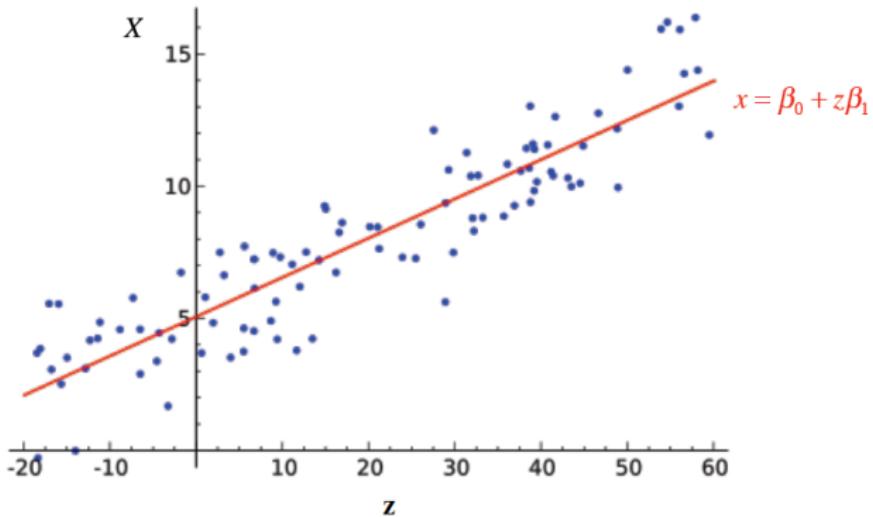
- ➊ Разделяющий алгоритм пытается найти  $p(y|x)$  из данных и затем классифицировать их или пытается найти разделяющую функцию между классами.  
Порождающий алгоритм пытается найти  $p(x,y)$ , которое затем трансформируется в  $p(y|x)$ .
- ➋ Порождающий алгоритм может использовать  $p(x,y)$ , чтобы генерировать новые данные, аналогичные уже существующим.  
Разделяющий алгоритм в общем имеет лучше характеристики в задачах классификации.
- ➌ Порождающий: Наивный Байес  
Разделяющий: Логистич. регрессия, SVM, Нейронные сети

# Порождающие модели на основе нейронных сетей

- ① Вариационный автокодер (**Variational AutoEncoder (VAE)**)
- ② Порождающие конкурирующие сети (**Generative Adversarial Networks (GANs)**)
- ③ Глубокая машина Больцмана (DBM)
- ④ Глубокая сеть доверия (DBN)

# Вспомним регрессию

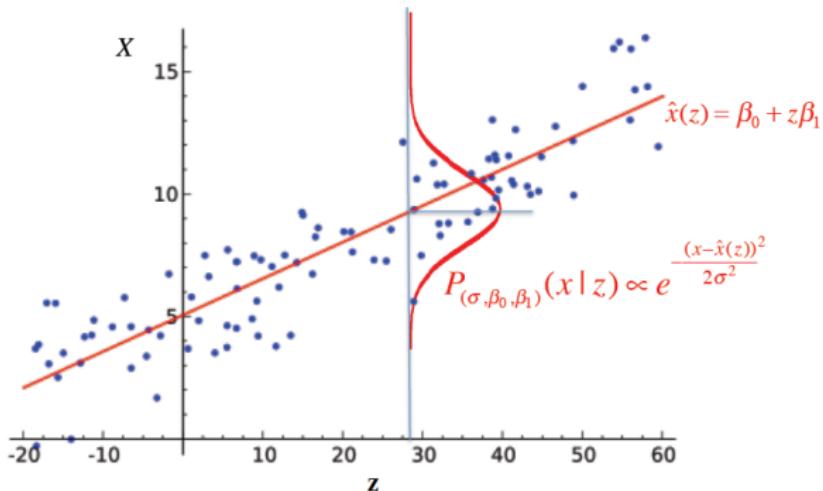
Часть людей представляют регрессию как множество точек и аппроксимирующую прямую



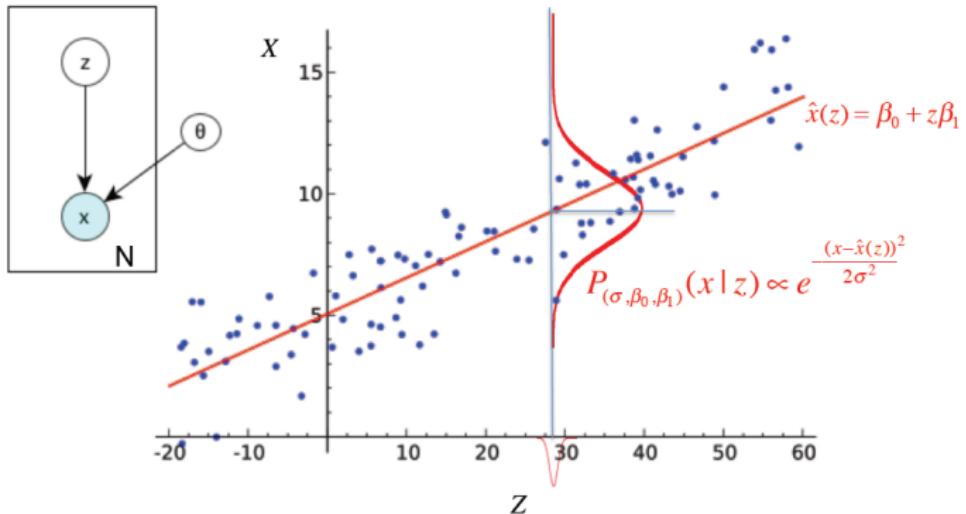
# Вспомним регрессию еще

Статистики добавляют  $P_\theta(X|Z)$ : плюсы добавления модели ошибок:

- Какова вероятность точки данных
- Доверительные границы
- Сравниваемость моделей

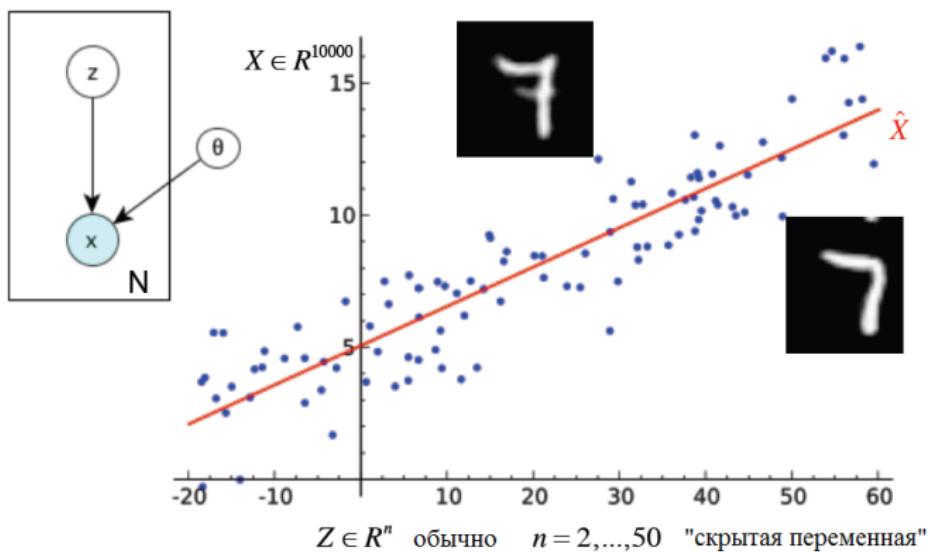


# Вспомним регрессию с графическими моделями



# Переход от малой размерности к большой

Переход от  $\mathbb{R}^1$  к  $\mathbb{R}^{10000}$



# Скрытые переменные

- Рассмотрим задачу генерации изображений цифр от 0 до 9.
- Если левая половина цифры - левая половина 5, то правая половина не может быть левой половиной 0. Это помогает принимать решение какую цифру генерировать.
- Такое решение отражается в скрытой переменной  $z$  (latent variable).
- Перед тем, как модель нарисует что-нибудь, она сначала сгенерирует число  $z$  из  $\{0, \dots, 9\}$ , затем убеждается, все ли штрихи совпадают с этой цифрой.
- $z$  называется “скрытой”, так как при данной цифре, порожденной моделью, мы не знаем точно, какие параметры  $z$  сгенеририровали цифру.

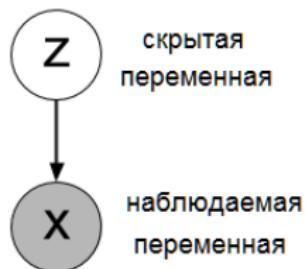
# Скрытые переменные

**Неформально:** Чтобы говорить о качестве модели, необходимо удостовериться, что для каждой точки  $\mathbf{x}$  в обучающей выборке  $S$ , есть “окружение” скрытой переменной, которое заставляет модель генерировать что-то очень похожее на  $\mathbf{x}$ .

**Формально:**

- Дано: вектор скрытых переменных  $\mathbf{z} \sim p(\mathbf{z})$ ;  
семейство функций  $f(\mathbf{z}; \theta)$ ,  $\theta$  - вектор параметров
- Нужно оптимизировать  $\theta$  так, что  $f(\mathbf{z}; \theta)$  производит выборку подобно  $\mathbf{x}$  с высокой вероятностью для каждого  $\mathbf{x} \in S$ , когда  $\mathbf{z}$  генерируется из  $p(\mathbf{z})$

# Некоторое отступление



Пусть  $x$  представляет “исходные значения пикселей изображения”, а  $z$  - двоичная переменная такая, что  $z = 1$ , если “ $x$  - изображение кота”.

# Некоторое отступление

$X =$



$P(z) = 1$  (точно кот)

$X =$



$P(z) = 0$  (точно не кот)

$X =$



$P(z) = 0.1$  (напоминает кота)

## Некоторое отступление

- Теорема Байеса устанавливает связь между переменными:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

- $p(z|x)$  - постериорная вероятность: “дано изображение, какова вероятность, что это кот?” Если можно сгенерировать  $z \sim p(z|x)$ , то значения можно использовать для построения классификатора, который скажет, является ли данное изображение котом или нет.

# Некоторое отступление

- $p(\mathbf{x}|\mathbf{z})$  - правдоподобие: “дано значение  $\mathbf{z}$ , как вероятно, что изображение  $\mathbf{x}$  определенной категории { кот / не кот}”. Если можно сгенерировать  $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$ , то можно сгенерировать изображения котов и изображения не котов также просто, как генерацию случайных чисел.
- $p(\mathbf{z})$  - априорная вероятность (любая информация о  $\mathbf{z}$ ), например, если известно, что 1/3 всех изображений - коты, то  $p(\mathbf{z} = 1) = 1/3$  и  $p(\mathbf{z} = 0) = 2/3$ .

# А это важно - здесь суть подхода

*Скрытые переменные можно интерпретировать в рамках байесовского подхода как априорные доверия, связанные с наблюдаемыми переменными.*

*Например, если мы полагаем, что  $x$  имеет многомерное нормальное распределение, то скрытая переменная  $z$  может представлять среднее и дисперсию нормального распределения. Распределение  $p(z)$ , определенное на параметрах, является априорным для  $p(x)$ .*

*Генерируя  $z$  (различные параметры), мы можем генерировать различные  $x$ !*

# В чем проблема?

- Для сложных задач мы не знаем, как генерировать из  $p(z|x)$  или как вычислить  $p(x|z)$ .
- Мы знаем форму  $p(z|x)$ , но соответствующее вычисление настолько является сложным, что мы не можем оценить его за приемлемое время.
- Можно попытаться использовать какой-нибудь известный подход для этого, но большинство из них медленно сходится.

# Идея вариационного вывода

Вместо сложного распределения  $p(\mathbf{z}|\mathbf{x})$ , используем простое параметрическое распределение  $q_\phi(\mathbf{z}|\mathbf{x})$ , например, нормальное, для которого известно, как получить апостериорное распределение. При этом мы подбираем параметры  $\phi$  таким образом, чтобы  $q_\phi$  было как можно ближе к  $p(\mathbf{z}|\mathbf{x})$ .

Близость определяется, например, при помощи дивергенции Кульбака-Лейблера.

# Иллюстрация близких распределений



# Совсем формально

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z}; \theta) p(\mathbf{z}) d\mathbf{z} = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} p(\mathbf{x}|\mathbf{z}; \theta) \rightarrow \max_{\theta}$$

- $f(\mathbf{z}; \theta)$  заменена распределением  $p(\mathbf{x}|\mathbf{z}; \theta)$
- Если модель способна воспроизвести обучающие примеры, то она также способна с большой вероятностью породить аналогичные и с малой вероятностью неподобные примеры.

# Забегая вперед

- В вариационном автокодере, если  $\mathbf{x} \in \mathbb{R}$ , то  $p(\mathbf{x}|\mathbf{z}; \theta) = \mathcal{N}(\mathbf{x}|f(\mathbf{z}; \theta), \sigma^2 \cdot I)$  - нормальное распределение со средним  $f(\mathbf{z}; \theta)$  и дисперсией  $\sigma^2 \cdot I$
- Тогда можно увеличивать постепенно  $p(\mathbf{x}|\mathbf{z}; \theta)$  делая  $f(\mathbf{z}; \theta)$  близким к  $\mathbf{x}$  для некоторого  $\mathbf{z}$
- Если  $\mathbf{x}$  - двоичное, то then  $p(\mathbf{x}|\mathbf{z}; \theta)$  - распределение Бернулли с параметром  $f(\mathbf{z}; \theta)$

## Забегая еще далее

Для решения задачи с  $p(x)$  имеются 2 проблемы:

- ① как определить скрытые переменные  $z$ , т.е. решить, какую информацию они представляют?
- ② что делать с интегрированием по  $z$ ?

Вариационный автокодер позволяет ответить на эти вопросы

# Как определить скрытые переменные

Мы хотим в идеале избежать:

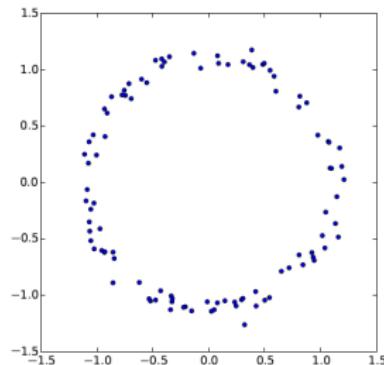
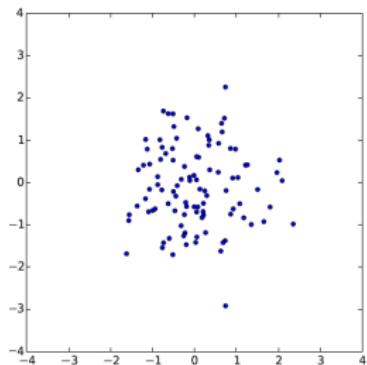
- принятия решения “в ручную”, какую информацию каждая координата вектора  $z$  кодирует
- в явном виде описывать зависимости (скрытую структуру) между координатами  $z$

# Вариационный автокодер (VAE)

# Подход VAE

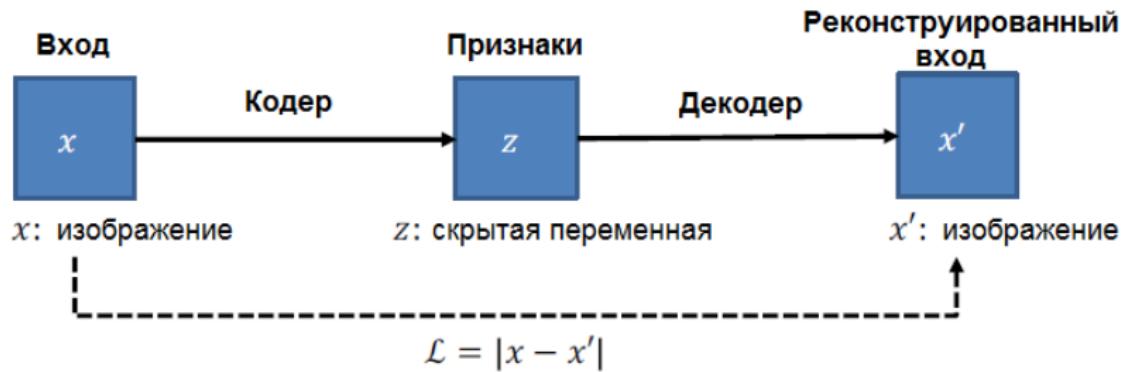
- Подход VAE для решения: он предполагает, что нет простой интерпретации координат  $\mathbf{z}$ , и вместо этого предлагает, что выборку  $\mathbf{z}$  можно сгенерировать из простого распределения, например, из  $\mathcal{N}(0, I)$  !
- Ключевая идея: любое распределение размерности  $d$  может быть сгенерировано, взяв множество  $d$  нормально распределенных переменных и отобразить их, используя достаточно сложную функцию

# Подход VAE

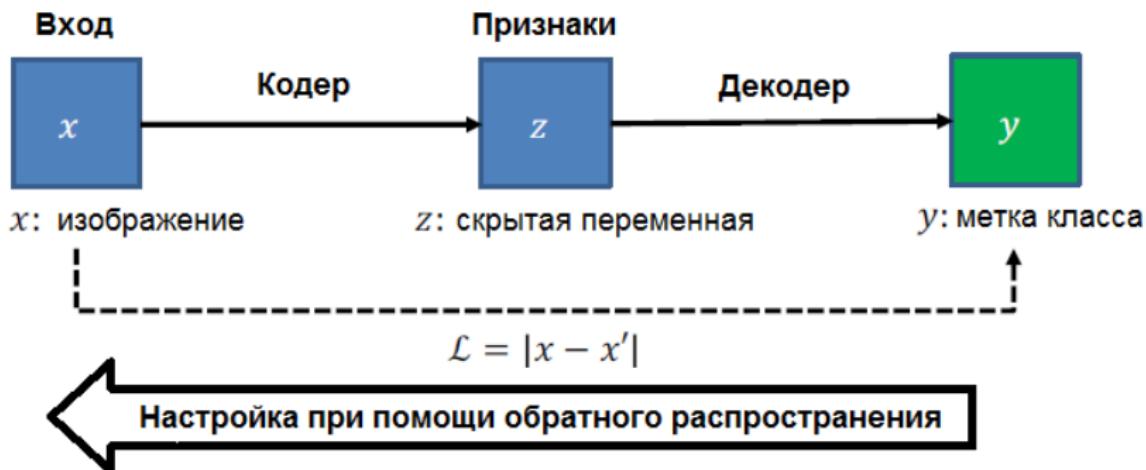


- Слева: выборка  $Z$  из нормального распределения
- Справа: та же выборка, но с  $g(z) = z/10 + z/\|z\|$  образует кольцо
- Детерминированная функция  $g$  “обучается” из данных

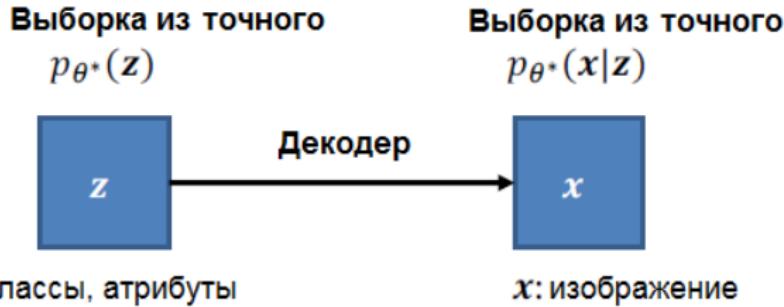
# Стандартный автокодер для представления признаков



# Стандартный автокодер для классификации



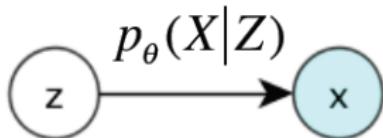
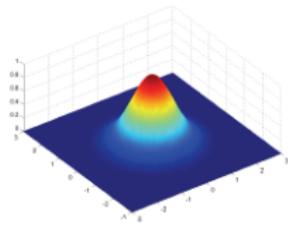
# Вариационный автокодер



- Значение генерируется из априорного распределения  $p_{\theta^*}(z)$
- Значение  $x^{(i)}$  генерируется из условного распределения  $p_{\theta^*}(x|z)$
- Точное значение параметра  $\theta^*$  и значения скрытых переменных  $z^{(i)}$  не известны

# Вариационный автокодер (кодирование)

Дано:  $\mathbf{z} \sim \mathcal{N}(0, I)$ ,  $\mathbf{x}|\mathbf{z} \sim p_{\theta^*}(\mathbf{x}|\mathbf{z})$



Один пример

Хотим получить (обучиться)  $\theta$  из  $N$  обучающих наблюдений  $\mathbf{x}^{(i)}$ ,  $i = 1, \dots, N$

# Вариационный автокодер

Выборка из точного

$$p_{\theta^*}(z)$$



Декодер

Выборка из точного

$$p_{\theta^*}(x|z)$$



$z$ : классы, атрибуты

$x$ : изображение

- Трудно оценить  $p_{\theta^*}(z)$  и  $p_{\theta^*}(x|z)$  на практике
- Необходимо аппроксимировать эти распределения

# Вариационный автокодер - основная идея

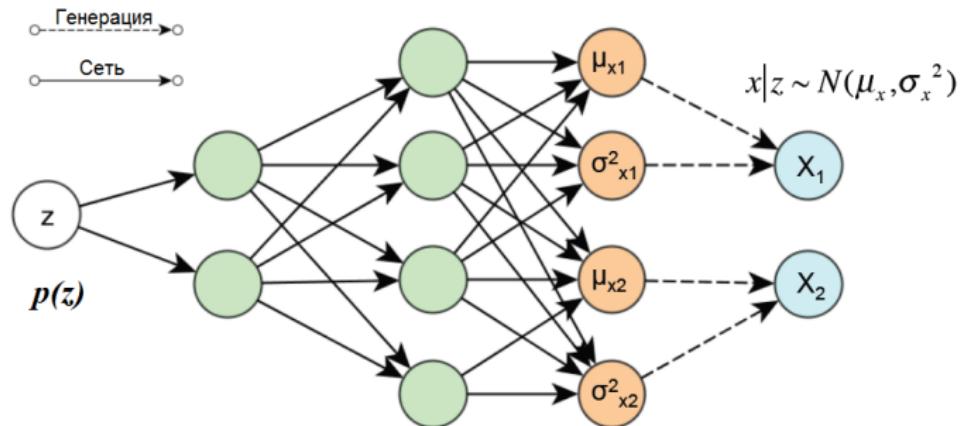


- Предполагаем, что  $p_\theta(z)$  имеет нормальное распределение
- Предполагаем, что  $p_\theta(x|z)$  имеет диагональное нормальное распределение
- Декодер оценивает среднее и дисперсию  $p_\theta(x|z)$

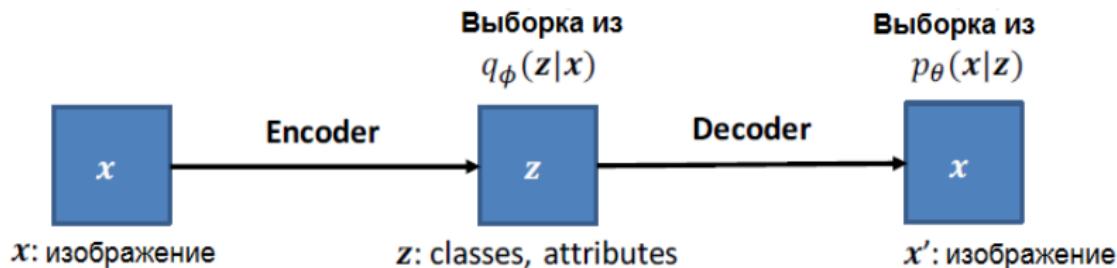
# Декодирование

Пусть  $z \in \mathbb{R}$  и  $x \in \mathbb{R}^2$

Идея: Нейр.сеть + Норм. распр (или Бернулли) с диагональной ковариацией  $\Sigma$



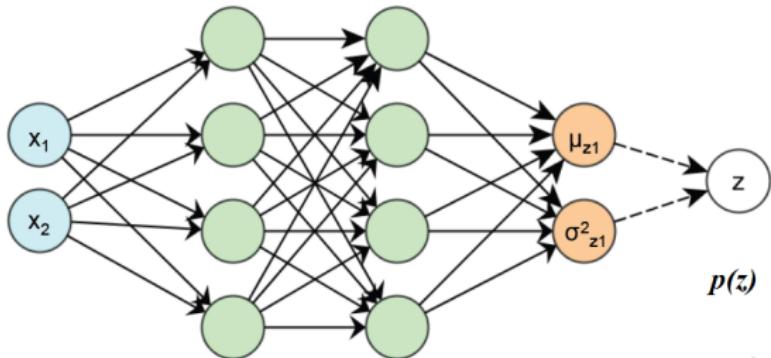
# Вариационный автокодер - основная идея



- Кодер оценивает среднее и дисперсию  $q_\phi(\mathbf{x}|\mathbf{z})$
- Декодер оценивает среднее и дисперсию  $p_\theta(\mathbf{x}|\mathbf{z})$
- Максимизируем нижнюю границу маргинального правдоподобия  $p_\theta(\mathbf{x}|\mathbf{z})$

# Декодирование

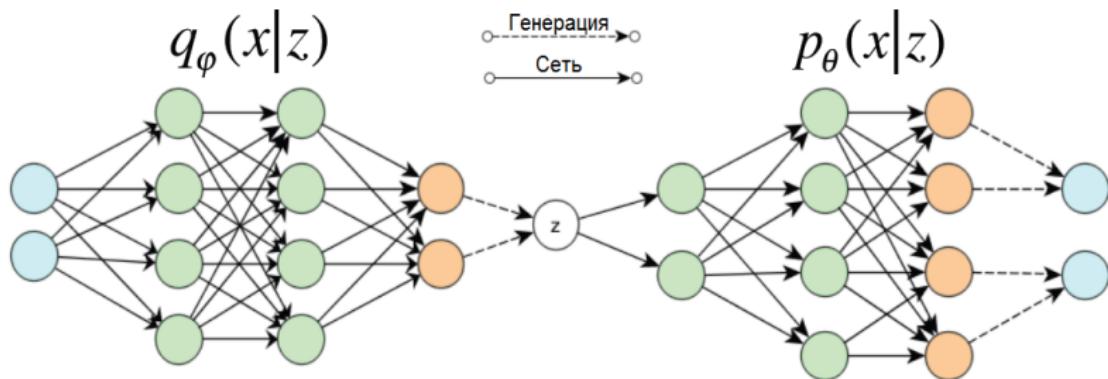
НН прямого распространения + Gaussian:

$$q_{\phi}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mu_{\mathbf{z}}(\mathbf{x}), \sigma_{\mathbf{z}}(\mathbf{x}))$$


## Таким образом

Если генерация  $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$  является “кодированием”,  
которое конвертирует наблюдение  $\mathbf{x}$  в скрытый код  $\mathbf{z}$ , то  
генерация  $\mathbf{x} \sim q(\mathbf{x}|\mathbf{z})$  - “декодирование”, которое  
реконструирует наблюдения из  $\mathbf{z}$ .

# Весь автокодер



- Параметры  $\varphi$  и  $\theta$  обучаются при помощи обратного распространения
- Главное определить функцию потерь

# Выбор функции потерь

- Какая техника в статистике является одной из лучших?
- Метод максимального правдоподобия: настройка  $\varphi$  и  $\theta$ , чтобы максимизировать функцию правдоподобия
- Максимизируем логарифм правдоподобия для заданного “изображения”  $x^{(i)}$  обучающего множества
- Суммируем по всем обучающим примерам

# Нижняя граница функции правдоподобия

$$\begin{aligned}L &= \log p(\mathbf{x}) = \sum_{\mathbf{z}} q(\mathbf{x}|\mathbf{z}) \log p(\mathbf{x}) = \sum_{\mathbf{z}} q(\mathbf{x}|\mathbf{z}) \log \left( \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \right) = \\&= \sum_{\mathbf{z}} q(\mathbf{x}|\mathbf{z}) \log \left( \frac{p(\mathbf{z}, \mathbf{x})}{q(\mathbf{z}|\mathbf{x})} \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \right) = \\&= \sum_{\mathbf{z}} q(\mathbf{x}|\mathbf{z}) \log \left( \frac{p(\mathbf{z}, \mathbf{x})}{q(\mathbf{z}|\mathbf{x})} \right) + \sum_{\mathbf{z}} q(\mathbf{x}|\mathbf{z}) \log \left( \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \right) = \\&= L^\nu + KL(q(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})) \geq L^\nu\end{aligned}$$

- $KL$  определяет, насколько  $q(\mathbf{z}|\mathbf{x})$  близко к  $p(\mathbf{z}|\mathbf{x})$
- $L^\nu$  - нижняя граница для правдоподобия;  $L^\nu = L$  при  $q(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$

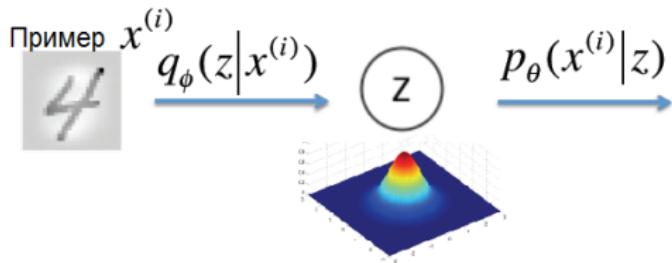
# Приближенный вывод

$$\begin{aligned} L^v &= \sum_z q(\mathbf{x}|\mathbf{z}) \log \left( \frac{p(\mathbf{z}, \mathbf{x})}{q(\mathbf{z}|\mathbf{x})} \right) = \\ &= \sum_z q(\mathbf{x}|\mathbf{z}) \log \left( \frac{p(\mathbf{x}|\mathbf{z}) \cdot p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right) = \\ &= \sum_z q(\mathbf{x}|\mathbf{z}) \log \left( \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right) + \sum_z q(\mathbf{x}|\mathbf{z}) \log (p(\mathbf{x}|\mathbf{z})) = \\ &= -KL(q(\mathbf{z}|\mathbf{x}^{(i)}) || p(\mathbf{z})) + \mathbb{E}_{q(\mathbf{z}|\mathbf{x}^{(i)})} [p(\mathbf{x}^{(i)}|\mathbf{z})] \end{aligned}$$

# Приближенный вывод

$$L^v = -KL(q(z|x^{(i)})||p(z)) + \mathbb{E}_{q(z|x^{(i)})} [p(x^{(i)}|z)]$$

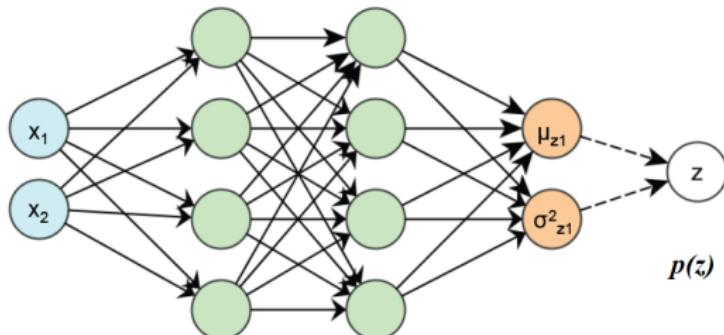
- $-KL(q(z|x^{(i)})||p(z))$  - **регуляризация**  $p(z)$  - обычно  $\mathcal{N}(z; 0, 1)$
- $\mathbb{E}_{q(z|x^{(i)})} [\log p(x^{(i)}|z)]$  - **качество реконструкции**,  $\log(1)$ , если  $x^{(i)}$  реконструируется идеально ( $z$  образует  $x^{(i)}$ )



# Вычисление регуляризации

- Используем  $\mathcal{N}(\mathbf{z}; 0, 1)$  как априорное для  $p(\mathbf{z})$
- $q(\mathbf{z}|\mathbf{x}^{(i)})$  - норм. с параметрами  $\mu_{\mathbf{z}}^{(i)}, \sigma_{\mathbf{z}}^{(i)}$ , определяемыми NN

$$-KL(q(\mathbf{z}|\mathbf{x}^{(i)})||p(\mathbf{z})) = \frac{1}{2} \sum_{j=1}^J \left( 1 + \log \left( \sigma_{\mathbf{z}_j}^{(i)2} \right) - \mu_{\mathbf{z}_j}^{(i)2} - \sigma_{\mathbf{z}_j}^{(i)2} \right)$$



# Вычисление качества реконструкции

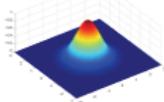
Приближенное вычисление  $\mathbb{E}_{q(\mathbf{z}|\mathbf{x}^{(i)})}$  генерацией  
 $\mathbf{z}^{(i,l)} \sim q(\mathbf{z}|\mathbf{x}^{(i)})$ ,  $l = 1, \dots, M$ :

$$\mathbb{E}_{q(\mathbf{z}|\mathbf{x}^{(i)})} [\log p(\mathbf{x}^{(i)}|\mathbf{z})] \approx \frac{1}{M} \sum_{l=1}^M \log (p(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}))$$

Пример  $\mathcal{X}^{(i)}$



$$q_\phi(z|x^{(i)})$$



$$\log(p_\theta(x^{(i)}|z^{(i,1)})), z^{(i,1)} \sim N(\mu_z^{(i)}, \sigma_z^{2(i)})$$

...

$$\log(p_\theta(x^{(i)}|z^{(i,M)})), z^{(i,M)} \sim N(\mu_z^{(i)}, \sigma_z^{2(i)})$$

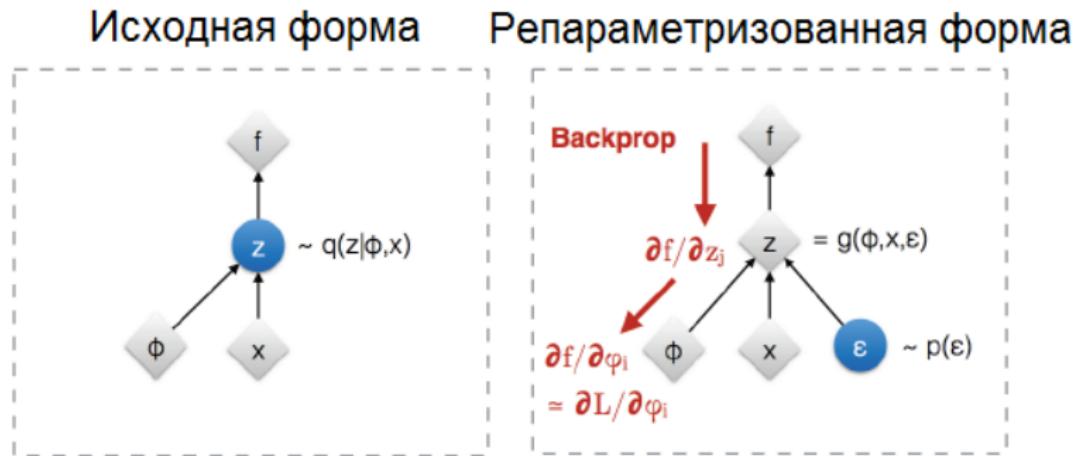
# Репараметризация

- Обратное распространение невозможно при случайной генерации
- Используется прием репараметризации

$$\mathbf{z}^{(i,l)} \sim \mathcal{N}(\mu_{\mathbf{z}}^{(i)}, \sigma_{\mathbf{z}}^{(i)}) \rightarrow \mathbf{z}^{(i,l)} = \mu_{\mathbf{z}}^{(i)} + \sigma_{\mathbf{z}}^{(i)} \odot \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, 1)$$

- $\mathbf{z}$  имеет то же распределение, но теперь возможно обратное распространение, т.к. есть детерминированная часть и шум

# Иллюстрация репараметризации



: Детерминированная вершина



: Случайная вершина

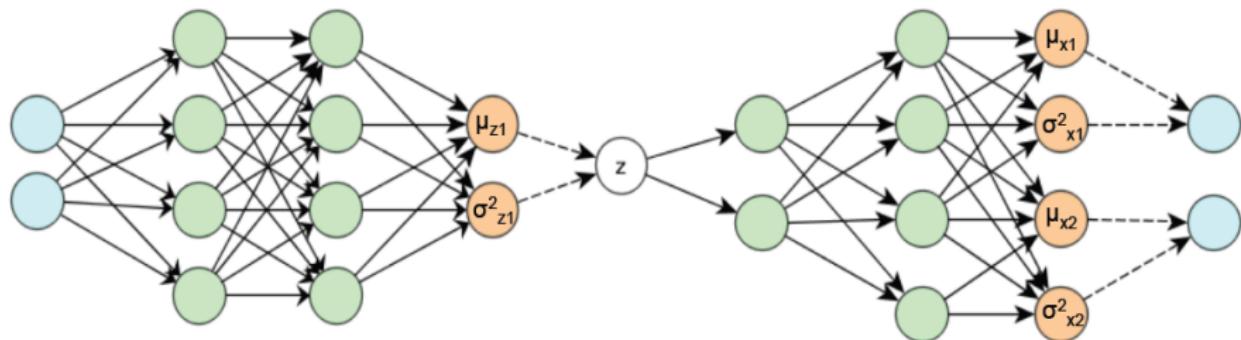
[Kingma, 2013]

[Bengio, 2013]

[Kingma and Welling 2014]

[Rezende et al 2014]

# Объединяем все вместе



# Объединяем все вместе

- **Регуляризация:** град. спуск, чтобы оптимизировать по  $\mathbf{x}^{(i)}$

$$-KL(q(\mathbf{z}|\mathbf{x}^{(i)})||p(\mathbf{z})) = \frac{1}{2} \sum_{j=1}^J \left( 1 + \log \left( \sigma_{\mathbf{z}_j}^{(i)2} \right) - \mu_{\mathbf{z}_j}^{(i)2} - \sigma_{\mathbf{z}_j}^{(i)2} \right)$$

- **Репродукция:** метод наим. квад. для постоянной дисперсии

$$-\log p(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}) = \sum_{i=1}^M \frac{1}{2} \log \left( \sigma_{\mathbf{x}_i}^2 \right) + \frac{\left( \mathbf{x}_j^{(i)} - \mu_{\mathbf{x}_j} \right)^2}{2\sigma_{\mathbf{x}_i}^2}$$

# Вариационные методы и Deep Learning

- Deep learning - эффективный инструментарий для оптимизации, например, методом градиентного спуска, когда имеется огромное количество параметров и данных.
- Вариационные байесовские методы являются аппаратом, при помощи которого можно “переписать” задачи статистического вывода в виде задач оптимизации.
- Комбинация вариационных байесовских методов и Deep learning позволяет реализовать вывод на очень сложных апостериорных распределениях вероятностей.

# Ресурсы и software

- Variational Autoencoder в TensorFlow:  
<https://jmetzen.github.io/2015-11-27/vae.html>
- Demo:  
[http://www.dpkingma.com/sgvb\\_mnist\\_demo/demo.html](http://www.dpkingma.com/sgvb_mnist_demo/demo.html)

# Соперничающие сети Generative adversarial networks (GAN)

# Соперничающие сети - Generative adversarial networks (GAN)

## Мотивация:

- Порождающие модели в основном обучаются при помощи метода максимума правдоподобия, который может быть неприменимым благодаря нормализации.
- Порождающая соперничающая сеть - это метод для обучения порождающих моделей при помощи нейронных сетей, обученных при помощи стохастического градиентного спуска вместо метода максимума правдоподобия.

# Соперничающие сети

Две задачи в машинном обучении:

- ① дискриминативная (*discriminative*): классифицирует входные данные
- ② порождающая (*generative*): пытается создать модель, которая может генерировать данные, похожие на обучающие данные

# Соперничающие сети

- Две сети:
  - ➊ порождающая (generative):  $y_G = G(z)$  - на входе  $z$  (шум), на выходе  $y_G$
  - ➋ дискриминативная (discriminative):  $y_D = D(x)$  - на входе  $x$  (данные), на выходе  $y_D$
- G-сеть должна научиться генерировать такие образцы  $y_G$ , что D-сеть  $D$  не сможет их отличить от эталонных образцов
- D-сеть должна научиться отличать эти сгенерированные образцы от настоящих

# Данные и шум



$x$



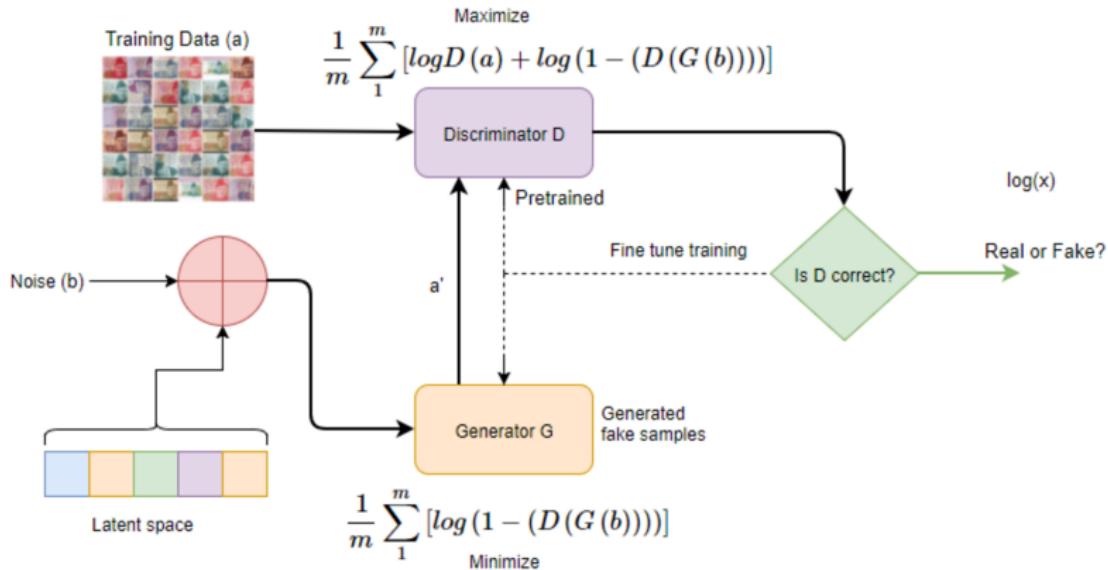
$G(z)$

<http://www.lab41.org/lab41-reading-group-generative-adversarial-nets/>

# Соперничающие сети - две модели

- Две модели: порождающая (фальшивомонетчик) и дискриминативная (банкир)
- Фальшивомонетчик пытается построить на выходе подделку на настоящие деньги, а банкир — отличить подделку от оригинала (обе модели начинают с рандомных условий, и в начале выдают в качестве результатов шум).
- Цель фальшивомонетчика - сделать такой продукт, который банкир не мог бы отличить от настоящего.
- Цель банкира — максимально эффективно отличать подделки от оригиналлов.
- Обе модели начинают игру друг против друга, где останется только один.

# Соперничающие сети



# Соперничающие сети - обучение G-сети

- G-сеть: максимизация функционала  $D(G(z))$ , т.е. G-сеть максимизирует результат работы D-сети.
- G-сеть должна научиться для любого значения  $z$  на входе сгенерировать на выходе такое значение  $y_G$ , подав которое на вход D-сети, получим максимальное значение на ее выходе  $y_D$  (сделать так, чтобы банкир был уверен, что подделки — настоящие.)

# Соперничающие сети - обучение G-сети

- Если на выходе D-сети стоит сигмоид, то она возвращает вероятность  $D(x)$  того, что на вход сети подано “правильное” значение, т.е. G-сеть стремится максимизировать вероятность того, что D-сеть не отличит результат работы порождающей сети от “эталонных” образцов (а это означает, что порождающая сеть порождает правильные образцы).
- G-сеть учится по градиенту результата работы D-сети.

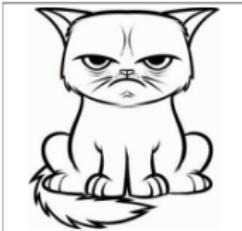
# Соперничающие сети - обучение D-сети

- D-сеть учится два раза за один шаг обучения: первый раз - на вход подается эталонный образец, а второй раз - результат G-сети.
- D-сеть: максимизация функционала  $D(x)(1 - D(G(z)))$ . Цель банкира - одновременно положительно опознавать оригиналы  $D(x)$ , и отрицательно - подделки  $(1 - D(G(z)))$ .
- D-сеть ничего не знает о том, что ей подано на вход: эталон или подделка. Об этом “знает” только функционал. Однако сеть учится в сторону градиента этого функционала.

# Соперничающие сети - обучение D-сети

D-сеть: возвращает 1 для реального образа, 0 - для шума

$$D(\text{}) = 1$$

$$D(\text{}) = 0$$

<http://www.lab41.org/lab41-reading-group-generative-adversarial-nets/>

# Соперничающие сети - процесс обучения

- ① На вход G-сети на каждом шаге подаются какие-то значения, например, совершенно случайные числа.
- ② На вход D-сети на каждом шаге подается очередной эталонный образец и очередной результат работы G-сети.

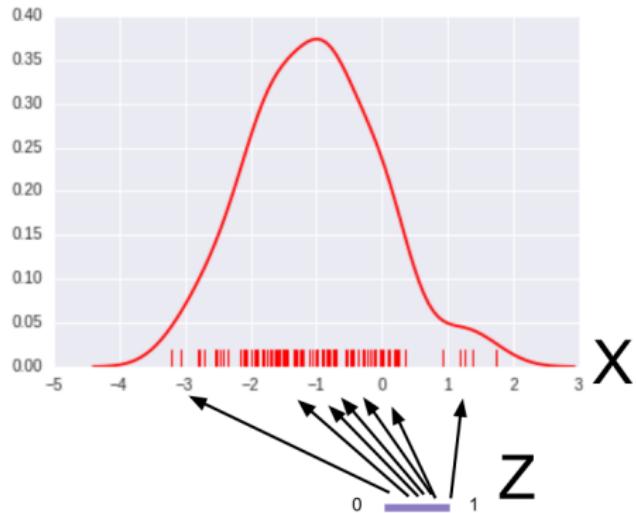
## Результат:

- ① G-сеть должна научиться генерировать образцы как можно более близкие к эталонным.
- ② D-сеть должна научиться отличать эталонные образцы от порожденных.

# Возвращаясь к порождающим моделям (практика)

- Нейронная сеть обучается генерированием простых нормально распределенных  $\mathcal{N}(-1, 1)$  сл. величин.
- На входе  $G$  одиночное случайное число из равномерного распределения шума:  $z \sim \text{uniform}(0, 1)$ .
- Мы хотим, чтобы  $G$  отобразила точки  $z_1, \dots, z_M$  в  $x_1, \dots, x_M$  так, чтобы густота точек  $x_i = G(z_i)$  соответствовала “сжатости” функции  $p_{data}(X)$ .
- Таким образом,  $G$  берет  $z$  и генерирует поддельные  $x'$ .

# Возвращаясь к порождающим моделям (практика)

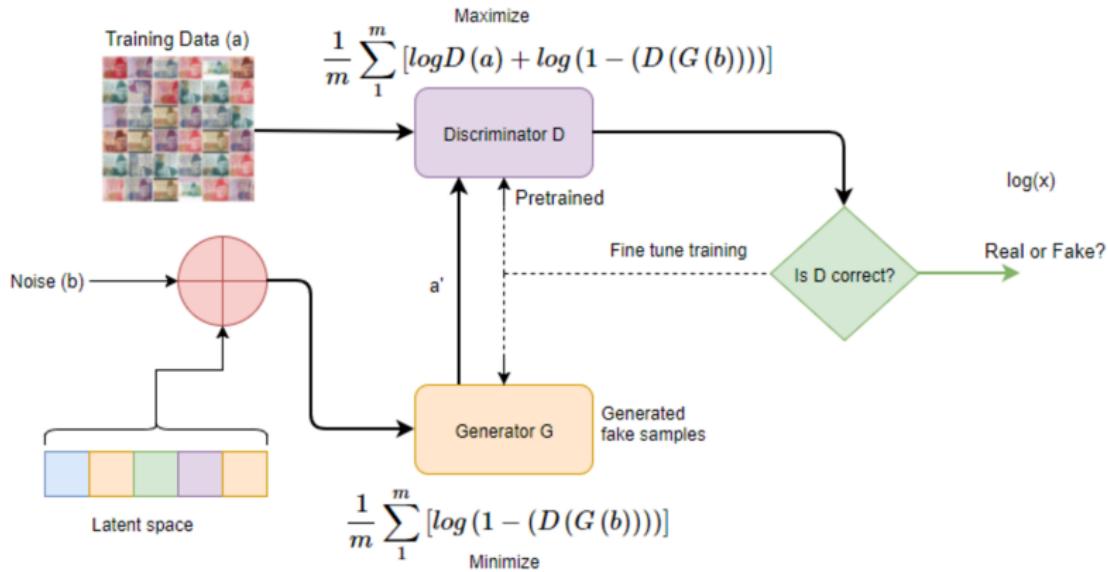


[http://blog.evjang.com/2016\\_06\\_01\\_archive.html](http://blog.evjang.com/2016_06_01_archive.html)

# Возвращаясь к порождающим моделям (практика)

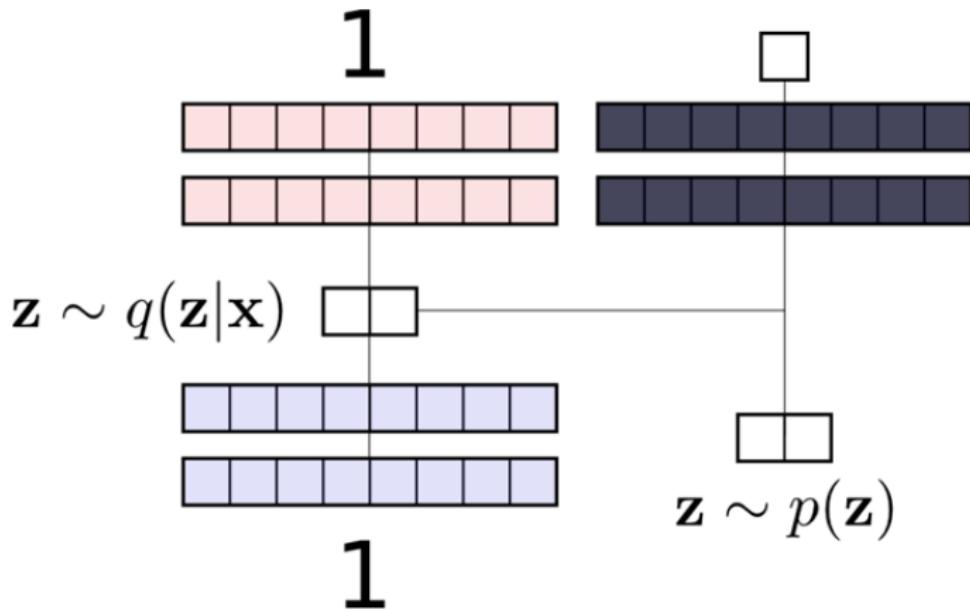
Тем временем на вход дискриминатора  $D$  подается  $x$  и на выходе - вероятность того, что вход принадлежит  $p_{data}$ . Пусть  $D_1$  и  $D_2$  копии  $D$  (они разделяют одни и те же параметры  $D_1(x) = D_2(x)$ ). Вход  $D_1$  - один сгенерированный вектор из “истинного” распределения  $x \sim p_{data}$  так, чтобы максимизировать  $D_1(x)$ . На выходе  $D_2$  - вектор  $x'$  (поддельный вектор, сгенерированный сетью  $G$ ) так, что, для оптимизации  $D$  мы минимизируем  $D_2(x')$ .

# Соперничающие сети

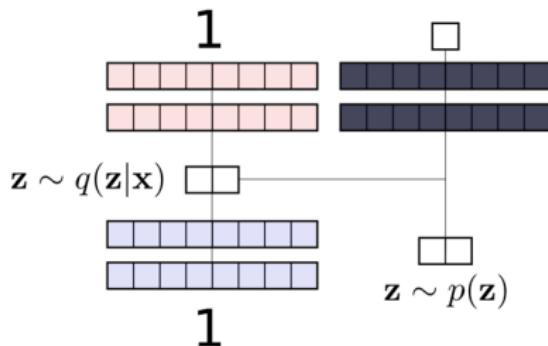


# Соперничающие автокодеры

# Соперничающие автокодеры - Adversarial autoencoders

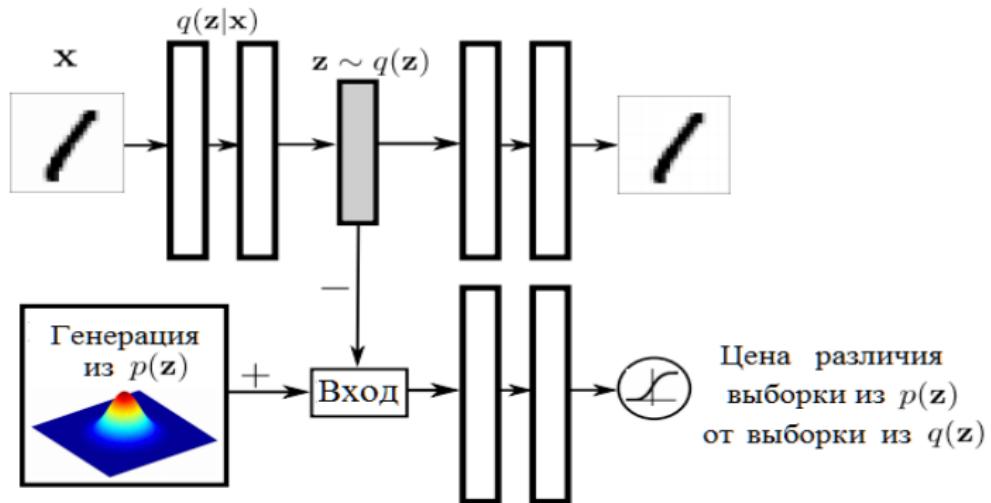


# Соперничающие автокодеры

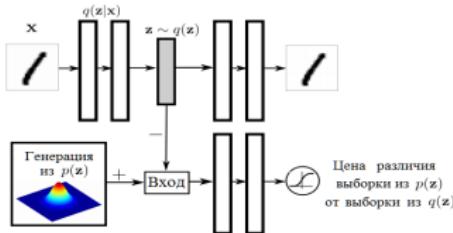


- Слева: входной вектор  $x$  (цифра “1”) преобразуется в код  $z$  кодером и поступает на декодер.
- Справа: выборочный вектор  $z$  генерируется в соответствии с априорным распределением  $p(z)$ . Дискриминатор оптимизируется, чтобы разделить выборки из  $p(z)$  и из  $q(z|x)$ .

# Соперничающие автокодеры

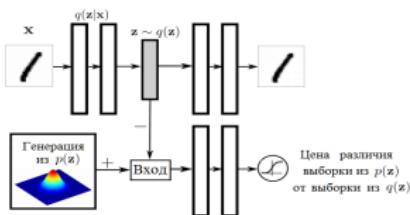


# Соперничающие автокодеры



- Верхняя часть рисунка - вероятностный автокодер. Для данного входа  $x$  генерируется скрытый код  $z$  из кодирующего распределения  $q(z|x)$  (моделируется как выход глубокой нейронной сети).
- В обычном автокодере этот кодер - детерминированный. Здесь он может быть вероятностным.
- Декодирующая сеть затем обучается, чтобы декодировать  $z$  и реконструировать исходный входной вектор  $x$ .
- Цель обучения - минимизация ошибки реконструкции (обычно квадратичная норма).

# Соперничающие автокодеры



- Ошибка реконструкции обеспечивает то, что процесс кодирования сохраняет информацию о входном изображении, но она не осуществляет ничего другого, соответствующего скрытым векторам  $z$ .
- В общем, их распределение описывается как агрегированное апостериорное распределение  $q(z) = \mathbb{E}_x q(z|x)$ .

# Соперничающие автокодеры

- Мы хотим, чтобы  $q(z)$  совпадало с априорным  $p(z)$ . Это достигается введением дополнительного слагаемого в функцию потерь автокодера, который измеряет дивергенцию между  $q$  и  $p$ .
- Это можно сделать при помощи соперничающего обучения: обучаем разделяющую сеть, которая постоянно обучается, чтобы различать реальные кодовые векторы  $Z$ , образованные кодированием реальных данных, от случайных кодовых векторов, сгенерированных из  $p$ . Если  $q$  почти полностью совпадает с  $p$ , то оптимальная разделяющая сеть должна иметь большую ошибку классификации.

# Соперничающие автокодеры - события

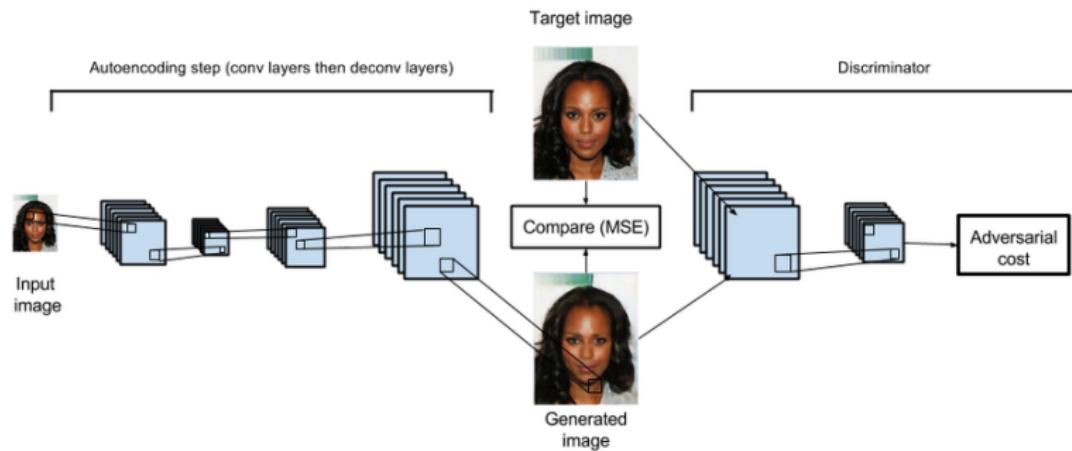
Для каждого блока данных есть три события:

- ① Блок входных векторов кодируется и декодируется, после чего сеть модифицируется на основе стандартной ошибки реконструирования.
- ② Блок входных векторов преобразуется кодером, после чего он соединяется с вектором, сгенерированным из априорного распределения  $p(z)$ . Дискриминатор затем модифицируется, используя двоичный кросс-энтропийный функционал потерь, основанный его способности разделять эти два вектора.

## Соперничающие автокодеры - события (продолжение)

3. Блок входных векторов преобразуется кодером, источник этих данных прогнозируется дискриминатором, и генератор (кодер) модифицируется, используя двоичный кросс-энтропийный функционал потерь, основанный на способности “обмануть” дискриминатор в предположении, что данные - из априорного распределения.

# Соперничающие автокодеры и сверточная сеть



[https://swarbrickjones.wordpress.com/2016/01/24/generative-adversarial-  
autoencoders-in-theano/](https://swarbrickjones.wordpress.com/2016/01/24/generative-adversarial-autoencoders-in-theano/)

# Вопросы

?