Генеративные модели

Лекция 2: Нормализующие потоки

Постановка задачи

Пусть у нас есть набор данных $\{\mathbf{x}_i\}_{i=1}^{n}$,

$\mathbf{x}_i \in \mathbb{R}^m$ – вектор, представляющий объект

$p_{data}(\mathbf{x})$ описывает как объекты распределены в пространстве $\mathbb{R}^m$

Цель: найти неизвестное распределение $p_{data}(\mathbf{x})$

Зная $p_{data}(\mathbf{x})$ мы сможем оценивать вероятность новых объектов и генерировать новые

Проблема высокой размерности

Объекты находятся в пространстве высокой размерности

Прямая оценка $p_{data}(\mathbf{x})$ в таких пространствах затруднена

Найти истинное распределение $p_{data}(\mathbf{x})$ почти невозможно

Будем искать аппроксимацию внутри некоторого заранее заданного ***параметрического семейства распределений*** $p_{\boldsymbol{\theta}}(\mathbf{x})$

$\boldsymbol{\theta}$ – параметры модели

Ищем $\boldsymbol{\theta}^*$, которые максимизируют правдоподобие:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

Связь правдоподобия и KL-дивергенции

Наша задача – найти такие параметры $\boldsymbol{\theta}$, при которых $p_{\boldsymbol{\theta}}(\mathbf{x})$ будет максимально близко к $p_{data}(\mathbf{x})$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p_{data}(\mathbf{x})$$

Будем искать такие параметры $\boldsymbol{\theta}$, которые минимизируют дивергенцию между истинным и модельным распределениями:

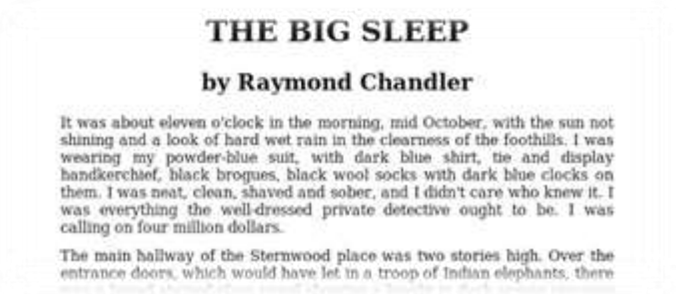
$$\min_{\boldsymbol{\theta}} D(p_{data}(\mathbf{x}) \mid p_{\boldsymbol{\theta}}(\mathbf{x}))$$

Минимизация прямой KL –дивергенции эквивалента максимизации правдоподобия

Обучение авторегресионных моделей

Объекты в реальном мире – многомерные случайные величины

Прямое моделирование $p_{\boldsymbol{\theta}}(\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{D})$ – сложная задача из-за проклятия размерности

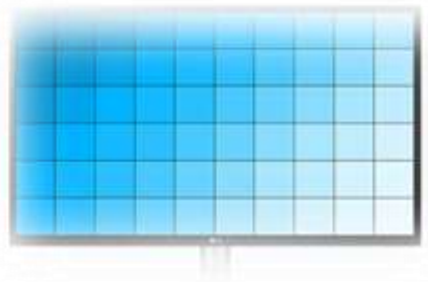


THE BIG SLEEP

by Raymond Chandler

It was about eleven o'clock in the morning, mid October, with the sun not shining and a look of hard wet rain in the clearness of the foothills. I was wearing my powder-blue suit, with dark blue shirt, tie and display handkerchief, black brogues, black wool socks with dark blue clocks on them. I was neat, clean, shaved and sober, and I didn't care who knew it. I was everything the well-dressed private detective ought to be. I was calling on four million dollars.

The main hallway of the Sternwood place was two stories high. Over the entrance doors, which would have let in a troop of Indian elephants, there



Разложение многомерной плотности

Разложим совместную вероятность на произведение условных:

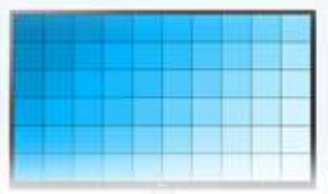
$$p_{\boldsymbol{\theta}}(\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{D}) = p_{\boldsymbol{\theta}}(\mathrm{x}_1) \cdot p_{\boldsymbol{\theta}}(\mathrm{x}_2|\mathrm{x}_1) \cdot p_{\boldsymbol{\theta}}(\mathrm{x}_3|\mathrm{x}_1, \mathrm{x}_2) \cdot \ldots \cdot p_{\boldsymbol{\theta}}(\mathrm{x}_\mathrm{D}|\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_{\mathrm{D}-1})$$

$$= \prod_{j}^{D} p_{\boldsymbol{\theta}}(\mathrm{x}_\mathrm{i}|\mathbf{x}_{1:j-1})$$

$\mathbf{x}_{1:j-1}$ – вектор всех компонент объекта $\mathbf{x}$ до позиции j

Теперь задача максимизации правдоподобия принимает вид:

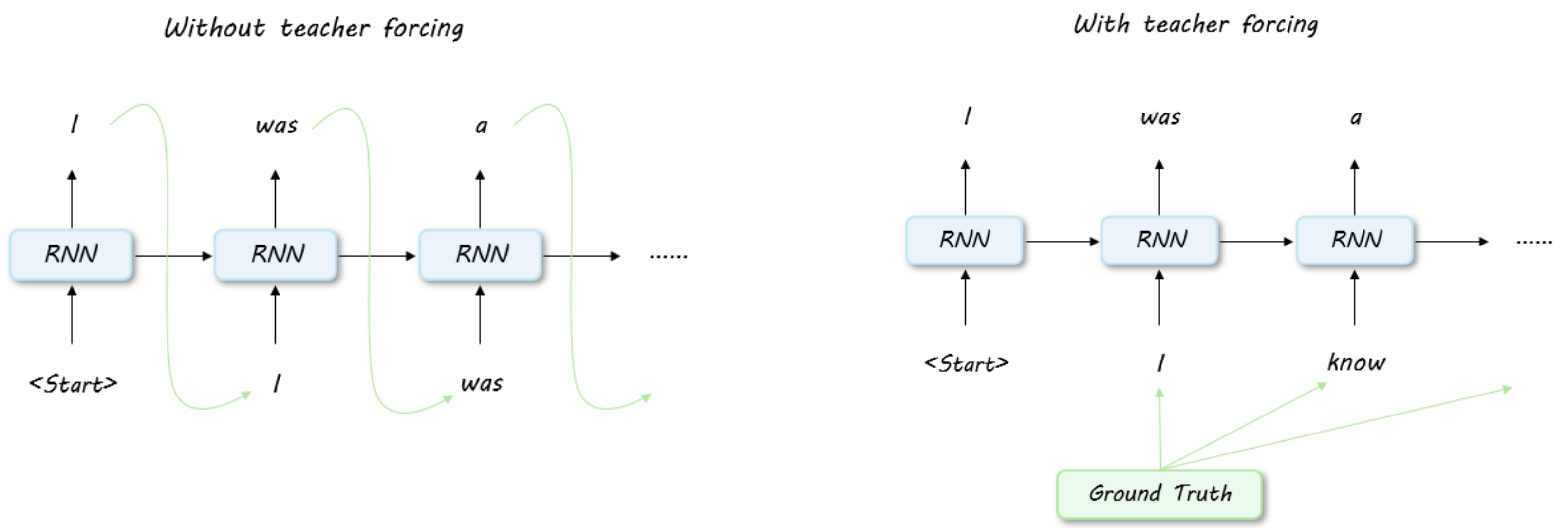
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{argmax} \sum_{i=1}^{n} \sum_{j}^{D} \log p_{\boldsymbol{\theta}}(x_{ij}|\mathbf{x}_{i,1:j-1})$$



Обучение авторегресионных моделей

В начале обучения модель часто совершает ошибки, которые с каждым шагом накапливаются

Решение:
Принудительно подавать на вход модели правильные данные (***teacher forcing***)



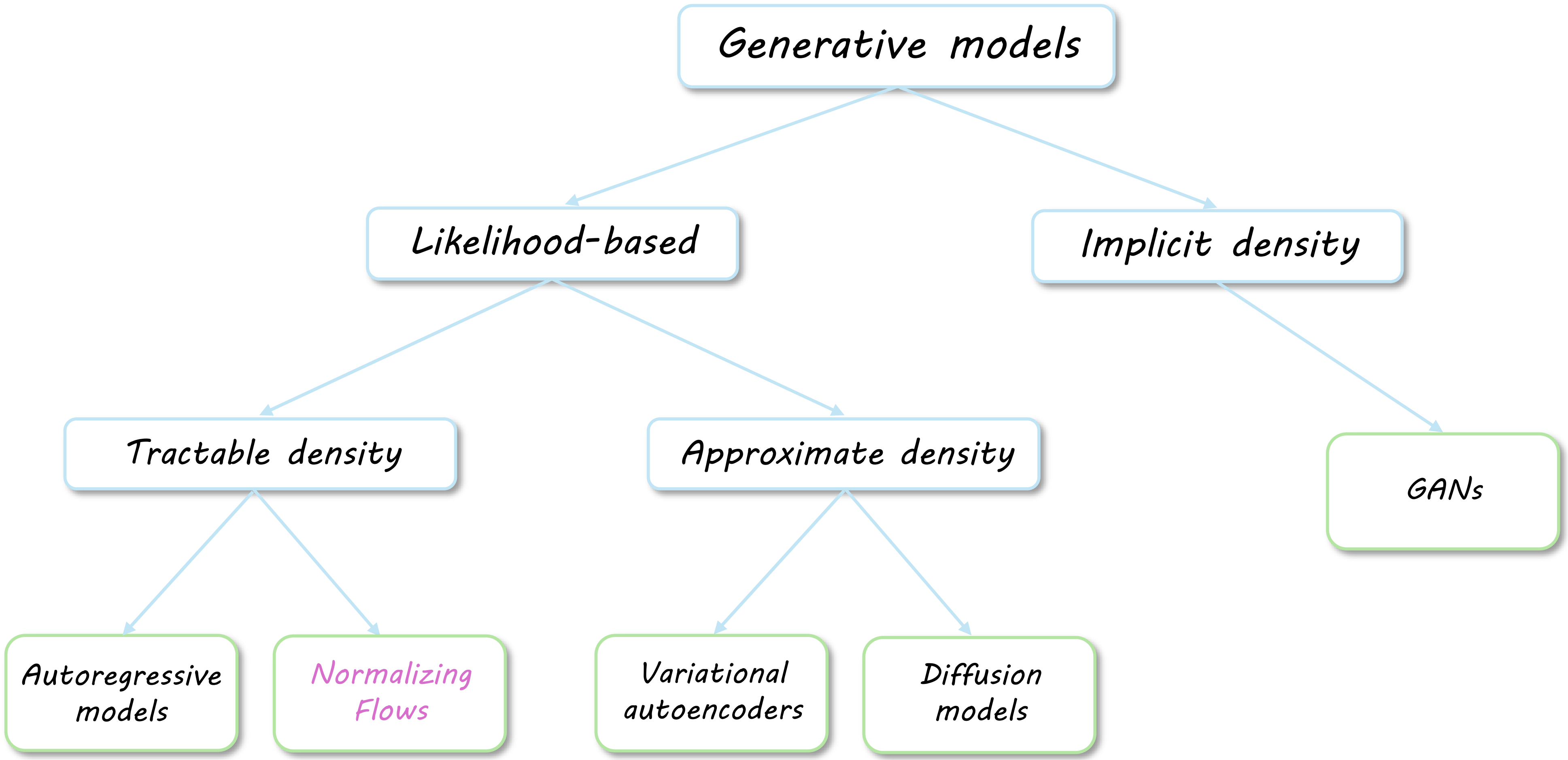
При генерации модель использует собственные предсказания

План

- Идея нормализующих потоков
- Авторегрессионные потоки
- Линейные потоки
- *NICE* и *RealNVP*
- *Glow*

Нормализующие потоки

Зоопарк генеративных моделей



Разложение многомерной плотности

Авторегрессионные модели:

⊕ Точно вычисляют правдоподобие

⊖ Медленная генерация

Вариационные автокодировщики:

⊕ Быстрая генерация из латентного пространства

⊖ Не могут точно вычислить правдоподобие

Нормализующие потоки:

⊕ Точно вычисляют правдоподобие

⊕ Используют латентное пространство для быстрой генерации

Нормализующие потоки берут лучшее от двух миров

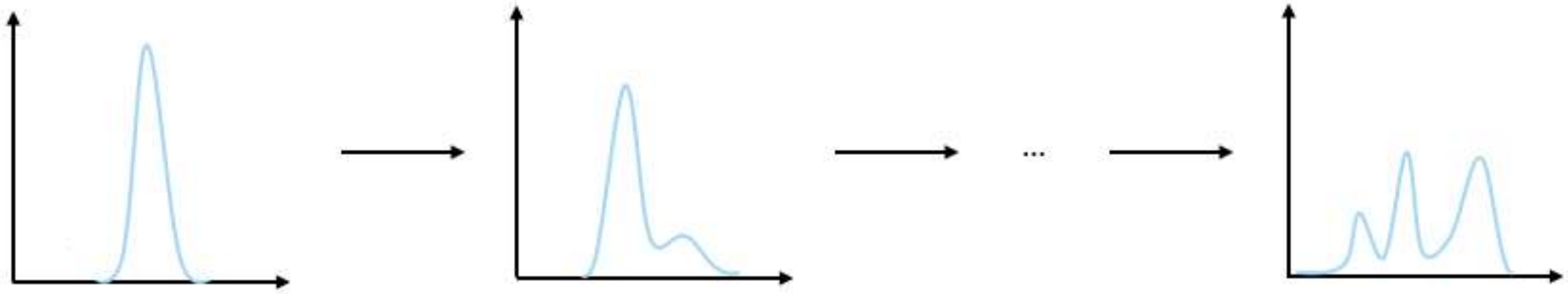
Идея нормализующих потоков

Идея:

Возьмём простое, хорошо изученное распределение:

$$\boldsymbol{z} \sim \mathcal{N}(0, \mathbf{I})$$

Шаг за шагом будем усложнять это распределение, пока оно не станет похоже на наши данные $\mathbf{x}$



Идея нормализующих потоков

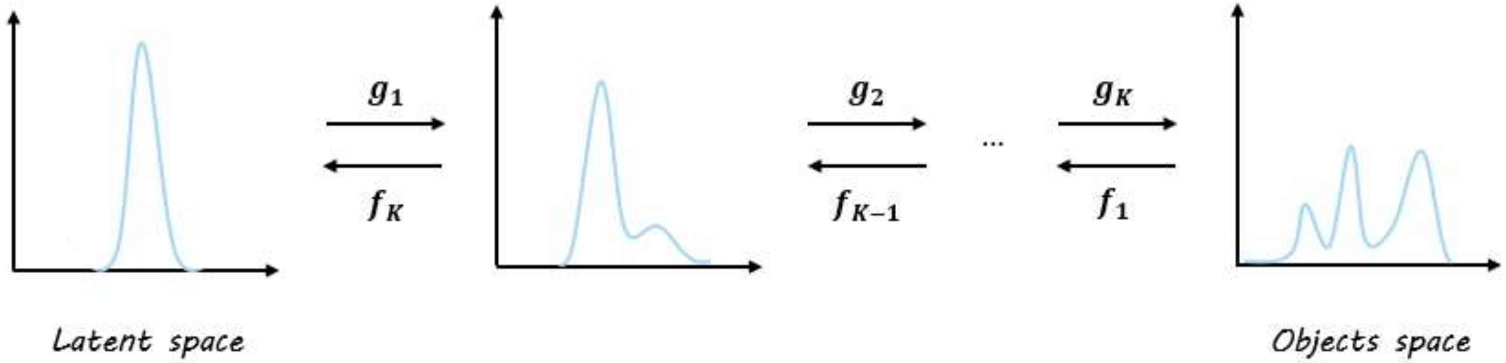
Особенность потоков – наличие двух направлений:

- ***Нормализующее направление*** (***для обучения***) $\mathbf{x} \rightarrow \boldsymbol{z}$:

  Берём реальный объект $\mathbf{x}$ и с помощью нормализующей функции $\boldsymbol{f}$ превращаем его в простой $\boldsymbol{z}$

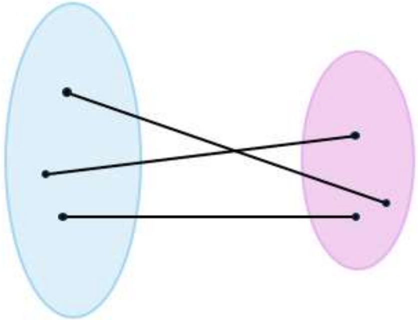
- **Генеративное направление** $\mathbf{z} \rightarrow \mathbf{x}$:

  Берём простой объект $\boldsymbol{z}$ и с помощью генеративной функции $\boldsymbol{g} = \boldsymbol{f}^{-1}$ создаём объект $\mathbf{x}$



Идея нормализующих потоков

Функции взаимно обратные → для каждой точки в пространстве **x** существует ровно одна из пространства **z**



Размерности исходного и преобразованного пространств должны совпадать:

$$dim\,\mathbf{x} = dim\,\mathbf{z}$$

Теорема о замене переменных

Наша цель – найти распределение $p(\mathbf{x})$

Связать неизвестную плотность $p(\mathbf{x})$ с известной плотностью простого распределения $p(\mathbf{z})$ нам позволяет ***формула замены переменных***:

$$p(\mathbf{x}) = p(\mathbf{z})\left|\det\left(\boldsymbol{J_f}\right)\right| = p(\mathbf{z})\left|\det\left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right)\right| = p(\boldsymbol{f}(\mathbf{x}))\left|\det\left(\frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

Здесь $\boldsymbol{J_f}(\mathbf{x})$ – матрица Якоби*:*

$$\boldsymbol{J_f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_D}{\partial x_1} & \cdots & \frac{\partial f_D}{\partial x_D} \end{pmatrix}$$

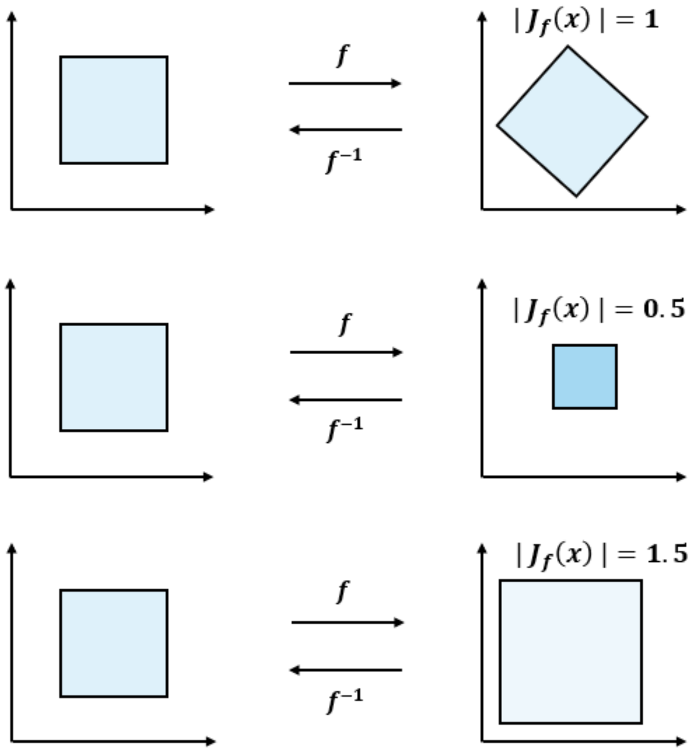
Якобиан

- Интуитивно определитель Якобиана можно понимать как коэффициент изменения объема
- Он показывает, насколько $\boldsymbol{f}$ сжимает или растягивает пространство

- Если объем не изменился $\left|\det \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})\right| = 1$, плотность в этой точке не изменится

- Если пространство сжалось $\left|\det \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})\right| < 1$, плотность в этой точке должна увеличиться

- Если пространство растянулось $\left|\det \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})\right| > 1$, плотность в этой точке должна уменьшиться



Пример

Пусть есть одномерная случайная величина $z \sim \mathcal{N}(0,1)$:

$$\pi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

Хотим получить новую переменную:

$$x = g(z) = 5z + 1$$

Найдем нормализующую функцию f, которая является обратной к g:

$$z = f(x) = g^{-1}(x) = \frac{x-1}{5}$$

Пример

Якобиан в одномерном случае:

$$\left|\frac{df(x)}{dx}\right| = \left|\frac{d}{dx}\left(\frac{x-1}{5}\right)\right| = \left|\frac{1}{5}\right| = \frac{1}{5}$$

Подставляем и получаем распределение для x:

$$p(x) = \frac{1}{5}\pi\left(\frac{x-1}{5}\right) = \frac{1}{\sqrt{2\pi\cdot 5^2}}e^{-\frac{(x-1)^2}{2\cdot 5^2}} = \mathcal{N}(1, 25)$$

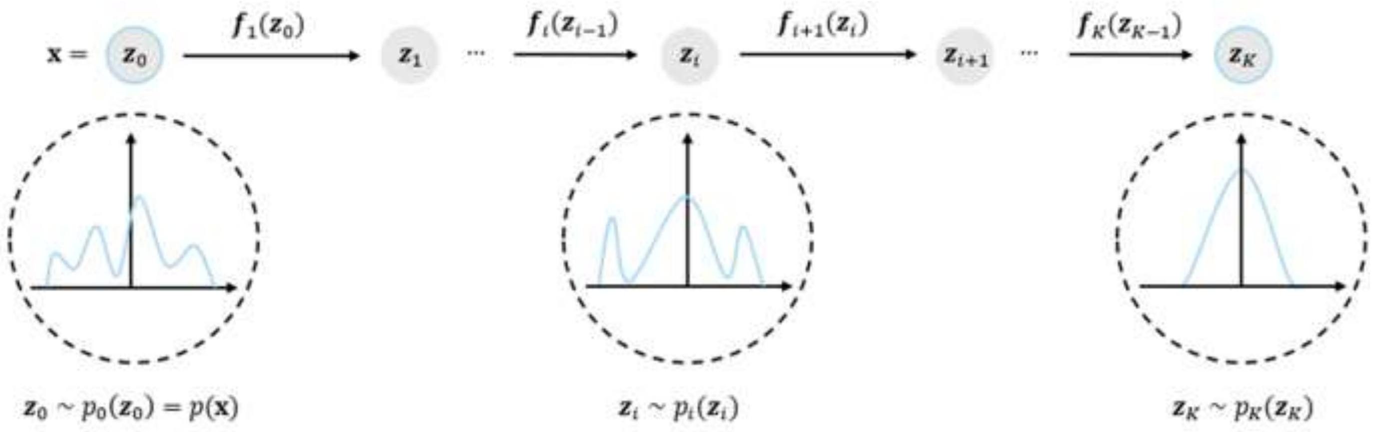
Композиция преобразований

- Одной функции $\boldsymbol{f}$ недостаточно, чтобы из $\mathcal{N}(0,1)$ получить сложное распределение наших данных

- Потоки используют последовательность из K простых и обратимых преобразований:

$$\mathbf{x} = \boldsymbol{z}_0 \overset{\boldsymbol{f}_1}{\rightarrow} \boldsymbol{z}_1 \overset{\boldsymbol{f}_2}{\rightarrow} \dots \overset{\boldsymbol{f}_K}{\rightarrow} \boldsymbol{z}_K$$

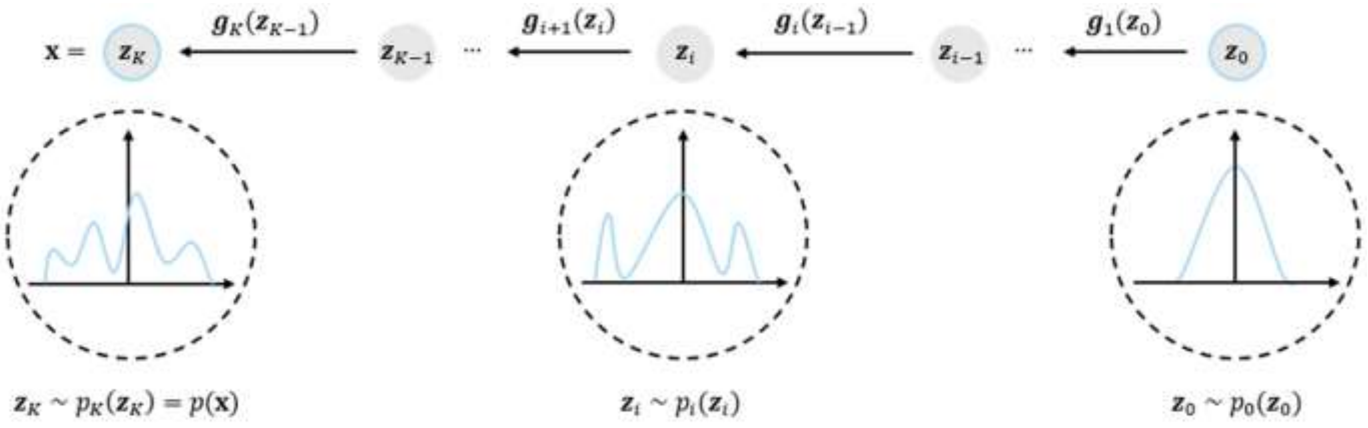
- Во время обучения мы используем нормализующее направление $\mathbf{x} \rightarrow \mathbf{z}$:



Композиция преобразований

- Генеративное направление $\boldsymbol{z} \to \mathbf{x}$ используется для создания новых объектов:

$$\boldsymbol{z}_0 \overset{g_1}{\to} \boldsymbol{z}_1 \overset{g_2}{\to} \dots \overset{g_K}{\to} \boldsymbol{z}_K = \mathbf{x}$$



Композиция преобразований

Нужно найти определитель якобиана всей композиции

Два математических факта:

- Матрица Якоби для композиций функций равна их произведению:

$$\frac{\partial \boldsymbol{f}}{\partial \mathbf{x}} = \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{z}_{K-1}} \cdot \frac{\partial \boldsymbol{f}_{K-1}}{\partial \boldsymbol{z}_{K-2}} \cdot \ldots \cdot \frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{z}_0}$$

- Определитель произведения матриц равен произведению определителей:

$$\det\left(\boldsymbol{J}_{\boldsymbol{f}}\right) = \det\left(\boldsymbol{J}_{\boldsymbol{f}_K}\right) \cdot \det\left(\boldsymbol{J}_{\boldsymbol{f}_{K-1}}\right) \cdot \ldots \cdot \det\left(\boldsymbol{J}_{\boldsymbol{f}_1}\right)$$

Будем считать определители для каждого шага $\boldsymbol{f}_k$ и перемножать их:

$$\det\left(\frac{\partial \boldsymbol{f}}{\partial \mathbf{x}}\right) = \prod_{k=1}^{K} \det\left(\frac{\partial \boldsymbol{f}_k}{\partial \boldsymbol{z}_{k-1}}\right)$$

Обучение нормализующих потоков

Чтобы сделать эту композицию обучаемой, представим $\boldsymbol{f}_k$ как нейросеть с параметрами $\boldsymbol{\theta}$

Плотность, которую моделирует наша модель:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = p(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x})) \left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}}}{\partial \mathbf{x}}\right)\right| = p(\boldsymbol{z}_K) \left|\prod_{k=1}^{K} \det\left(\frac{\partial \boldsymbol{f}_{k,\boldsymbol{\theta}}}{\partial \boldsymbol{z}_{k-1}}\right)\right|$$

$\boldsymbol{z}_K$ – финальный вектор в латентном пространстве

Максимизируем правдоподобие:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log p(\boldsymbol{z}_K) + \sum_{k=1}^{K} \log \left|\det\left(\frac{\partial \boldsymbol{f}_{k,\boldsymbol{\theta}}}{\partial \boldsymbol{z}_{k-1}}\right)\right|$$

- $\log p(\boldsymbol{z}_K)$ отвечает за то, чтобы выход потока соответствовал базовому распределению
- $\sum_{k=1}^{K} \log \left|\det\left(\frac{\partial \boldsymbol{f}_{k,\boldsymbol{\theta}}}{\partial \boldsymbol{z}_{k-1}}\right)\right|$ отслеживает суммарное изменение объема после всех преобразований

Функция потерь

Функция потерь:

$$\mathcal{L}_\theta(\mathbf{x}) = -\log p_{\boldsymbol{\theta}}(\mathbf{x}) = -\left(\log p(\boldsymbol{z}_K) + \sum_{k=1}^{K} \log\left|\det\left(\frac{\partial \boldsymbol{f}_{k,\boldsymbol{\theta}}}{\partial \boldsymbol{z}_{k-1}}\right)\right|\right)$$

Требования к преобразованиям $\boldsymbol{f}_{k,\boldsymbol{\theta}}$:

- Наличие обратной функции $\boldsymbol{g}_{\boldsymbol{\theta}} = \boldsymbol{f}_{\boldsymbol{\theta}}^{-1}$
- Быстрое вычисление определителя $\det(\boldsymbol{J}_{\boldsymbol{f}})$ (в идеале за $\mathcal{O}(D)$)

Будем искать функции, которые удовлетворяют этим условиям

Авторегрессионные потоки

Авторегрессионные потоки

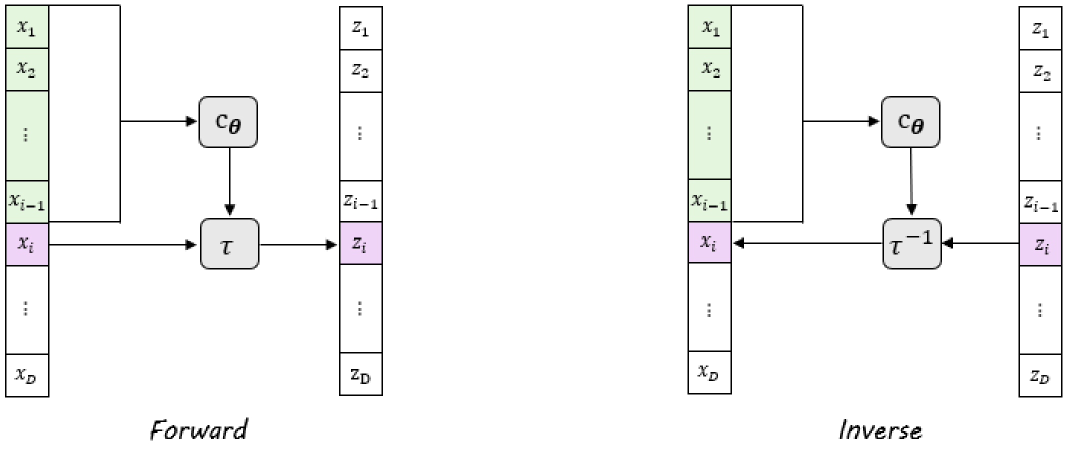
Авторегрессионные потоки вдохновлены идеей классических *AR* моделей:

$$z_i = \tau(x_i; c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1}))$$

τ – ***transformer***, обратимая функция, которая преобразует x_i в z_i
$c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1})$ – ***conditioner***, модель, которая генерирует параметры для τ на основе $\mathbf{x}_{1:i-1}$

Если τ обратим, то и весь поток обратим:

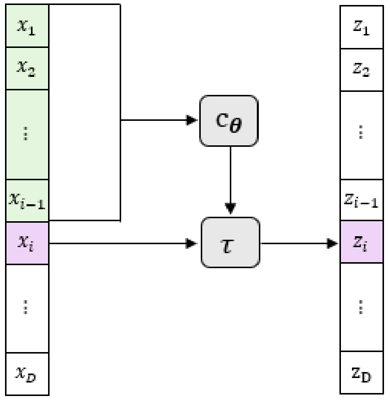
$$x_i = \tau^{-1}(z_i; c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1}))$$



Авторегрессионные потоки

- При прямом проходе $\mathbf{x}$ полностью известен и мы можем параллельно вычислить все z_i

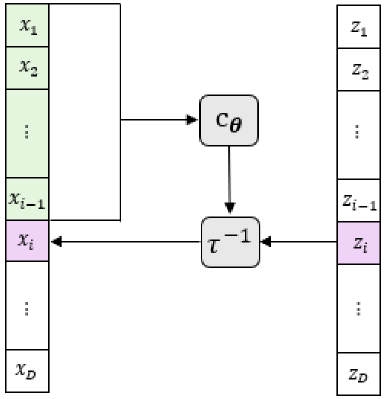
$$z_i = \tau(x_i; c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1}))$$



Forward

- При обратном проходе $\boldsymbol{z}$ полностью известен, но для генерации x_i нам нужен контекст $\mathbf{x}_{1:i-1}$

$$x_i = \tau^{-1}(z_i; c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1}))$$



Inverse

Авторегрессионные потоки

Якобиан имеет треугольный вид, поскольку $\frac{\partial z_i}{\partial x_j} = 0$ при $j > i$:

$$\boldsymbol{J}_{\boldsymbol{f_\theta}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\partial z_D}{\partial x_1} & \cdots & \frac{\partial z_D}{\partial z_D} \end{pmatrix}$$

Определитель треугольной матрицы считается за $\mathcal{O}(D)$:

$$\log|\det \boldsymbol{J}_{\boldsymbol{f_\theta}}(\mathbf{x})| = \sum_{i=1}^{D} \log \left| \frac{\partial \tau(x_i; c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1}))}{\partial x_i} \right|$$

$Affine\ Transformers$

Аффинное преобразование:

$$z_i = \alpha_i \cdot x_i + \beta_i$$

α_i и β_i – параметры масштаба и сдвига

Такой поток всегда обратим (если $\alpha_i \neq 0$):

$$x_i = \frac{z_i - \beta_i}{\alpha_i}$$

Частная производная $\frac{\partial z_i}{\partial x_j} = \alpha_i$

Логарифм определителя якобиана для всего преобразования:

$$\log|\det \boldsymbol{J}_{\boldsymbol{f_\theta}}(\mathbf{x})| = \sum_i^D |\alpha_i|$$

Masked Conditioners

Хотим посчитать параметры $c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1})$ для генерации z_i:

$$z_i = \tau(x_i; c_{\boldsymbol{\theta}}(\mathbf{x}_{1:i-1}))$$

Будем использовать единую нейросеть для вычисления всех параметров

Будем использовать маски, чтобы сохранить авторегрессию в этой сети:

- В полносвязных сетях на веса накладываются бинарные маски (***MADE***)
- В свёрточных сетях используются ***causal convolution*** (***PixelCNN***)

Либо на обучении, либо на генерации будет медленная скорость

Есть 2 популярные архитектуры:

- ***MAF*** (***Masked Autoregressive Flow***) – создана для быстрого обучения
- ***IAF*** (***Inverse Autoregressive Flow***) – та же архитектура, но развернута для быстрой генерации

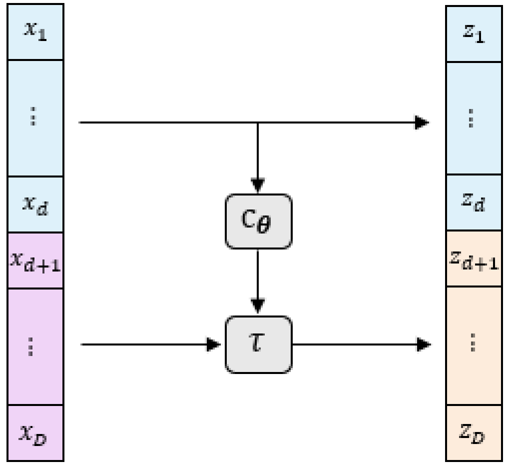
Coupling Layers

Чтобы решить проблему вычислительной асимметрии были предложены ***слои связи*** (***coupling layers***)

Делим входной вектор $\mathbf{x}$ на две части:

$$\mathbf{x} = [\mathbf{x}_{1:\mathrm{d}}; \mathbf{x}_{d+1:D}]$$

- Первая часть $\mathbf{x}_{1:\mathrm{d}}$ преобразуется напрямую без использования $c_{\boldsymbol{\theta}}$
- Вторая часть $\mathbf{x}_{d+1:D}$ преобразуется с помощью τ



Обычно преобразование первой части тождественное:

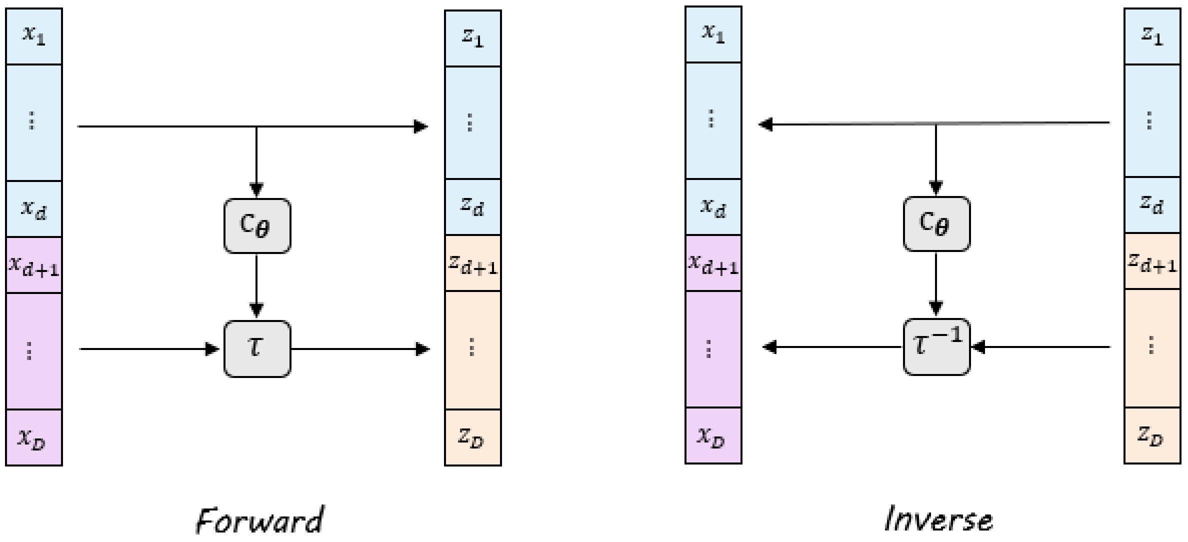
$$\begin{cases} \mathbf{z}_{1:\mathrm{d}} = \mathbf{x}_{1:\mathrm{d}} \\ \mathbf{z}_{\mathrm{d}+1:D} = \tau(\mathbf{x}_{\mathrm{d}+1:D};\ c_\theta(\mathbf{x}_{1:\mathrm{d}})) \end{cases}$$

Свойства слоёв связи

Симметричные вычисления:

- Прямой проход быстр, так как $\mathbf{x}$ известен
- Обратный проход тоже быстр, так как мы знаем первую половину $\mathbf{x}_{1:\mathrm{d}}$:

$$\begin{cases} \mathbf{x}_{1:\mathrm{d}} = \mathbf{z}_{1:\mathrm{d}} \\ \mathbf{x}_{\mathrm{d}+1:D} = \tau^{-\mathbf{1}}(\mathbf{z}_{\mathrm{d}+1:D};\, c_\theta(\mathbf{x}_{1:\mathrm{d}})) = \tau^{-\mathbf{1}}(\mathbf{z}_{\mathrm{d}+1:D};\, c_\theta(\mathbf{z}_{1:\mathrm{d}})) \end{cases}$$



Свойства слоёв связи

Блочный нижнетреугольный якобиан:

$$\boldsymbol{J}_f = \begin{pmatrix} \dfrac{\partial \boldsymbol{z}_{1:d}}{\partial \mathbf{x}_{1:d}} & \dfrac{\partial \boldsymbol{z}_{1:d}}{\partial \mathbf{x}_{d+1:D}} \\ \dfrac{\partial \boldsymbol{z}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \dfrac{\partial \boldsymbol{z}_{d+1:D}}{\partial \mathbf{x}_{d+1:D}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \\ \boldsymbol{A} & \boldsymbol{D} \end{pmatrix}$$

Определитель равен произведению определителей диагонального блока:

$$\det\left(\boldsymbol{J}_f\right) = \det\left(\boldsymbol{I}\right) \cdot \det\left(\boldsymbol{D}\right) = \det\left(\boldsymbol{D}\right)$$

Линейные потоки

Линейные потоки

- Авторегрессионные потоки и слои связи чувствительны к порядку переменных
- Обычное ***перемешивание*** (***permutation***) – не обучаемая операция

Решение:

- Вставить между нелинейными слоями обучаемый линейный поток:

$$\boldsymbol{z} = \mathbf{W}\mathbf{x} + \boldsymbol{b}$$

- Это позволит модели самой находить оптимальное перемешивание

- Линейные потоки ограничены в выразительности
- Используются в основном для перестановок

Линейные потоки

Проблема:

- Определитель якобиана считается за $\mathcal{O}(D^3)$

Решние:

- Не будем обучать матрицу напрямую, параметризуем её через матричные разложения

Самый популярный способ – использовать ***LU-разложение***:

$$\boldsymbol{W} = \boldsymbol{PLU}$$

$\boldsymbol{P} -$ фиксированная матрица перестановок (не обучается)
$\boldsymbol{L} -$ нижнетреугольная матрица
$\boldsymbol{U} -$ верхнетреугольная матрица

Линейные потоки

- Матрица $\boldsymbol{P}$ не обучается

- Позднее были работы, которые использовали ***QR-разложение:***

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{R}$$

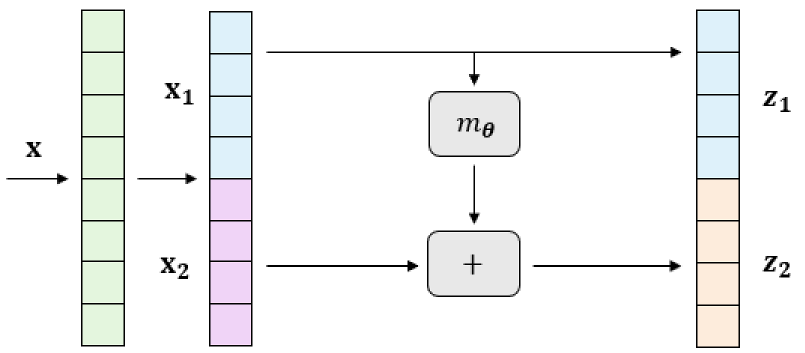
Для изображений обычно применяется ***обратимая*** $\mathbf{1} \times \mathbf{1}$ ***свертка***

- Это линейный поток, который перемешивает информацию между каналами

$NICE$ и $RealNVP$

NICE: *Additive Coupling Layers*

Разделяем вектор $\mathbf{x}$ на 2 части $\mathbf{x}_{1:\mathrm{d}}$ и $\mathbf{x}_{\mathrm{d}+1:D}$:



$$\begin{cases} \mathbf{z}_{1:\mathrm{d}} = \mathbf{x}_{1:\mathrm{d}} \\ \mathbf{z}_{\mathrm{d}+1:D} = \mathbf{x}_{\mathrm{d}+1:D} + m_\theta(\mathbf{x}_{1:\mathrm{d}}) \end{cases}$$

m_θ – произвольная нейросеть

Такая архитектура легко обратима:

$$\begin{cases} \mathbf{x}_{1:\mathrm{d}} = \mathbf{z}_{1:\mathrm{d}} \\ \mathbf{x}_{\mathrm{d}+1:D} = \mathbf{z}_{\mathrm{d}+1:D} - m_\theta(\mathbf{z}_{1:\mathrm{d}}) \end{cases}$$

Якобиан имеет блочный нижнетреугольный вид:

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{I}_d & \mathbf{0} \\ \dfrac{\partial \mathbf{z}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \boldsymbol{I}_{D-d} \end{pmatrix}$$

$NICE$: Проблемы

Определитель якобиана всегда равен единице, поэтому такое преобразование сохраняет объем

Даже композиция из нескольких аддитивных слоёв все равно сохраняет объем

Авторы $\boldsymbol{NICE}$ предложили в конце добавить слой масштабирования:

$$\boldsymbol{z} = \boldsymbol{s} \odot \boldsymbol{h}$$

$\boldsymbol{h}$ – выход последнего связующего слоя
$\boldsymbol{s}$ – обучаемый вектор масштаба
$\odot$ – поэлементное умножение

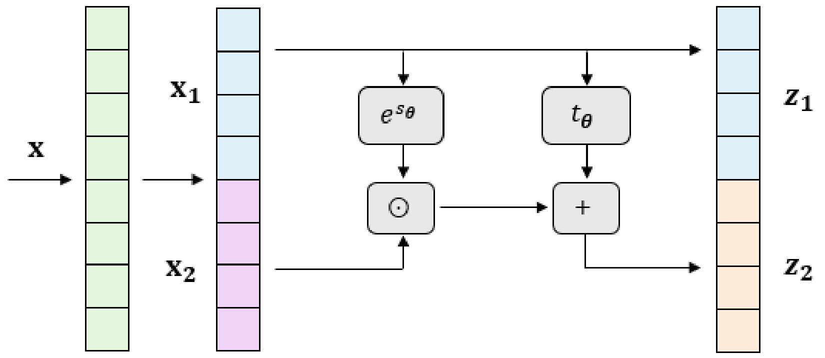
Определитель якобиана:

$$\det(\boldsymbol{J}) = \prod_{i=1}^{D} \boldsymbol{s}_i$$

$RealNVP$: $Affine\ Coupling\ Layers$

Разделяем вектор $\mathbf{x}$ на 2 части $\mathbf{x}_{1:\mathrm{d}}$ и $\mathbf{x}_{\mathrm{d}+1:D}$:

$$\begin{cases} \mathbf{z}_{1:\mathrm{d}} = \mathbf{x}_{1:\mathrm{d}} \\ \mathbf{z}_{\mathrm{d}+1:D} = \mathbf{x}_{\mathrm{d}+1:D} \odot e^{s_{\theta}(\mathbf{x}_{1:\mathrm{d}})} + t_{\boldsymbol{\theta}}(\mathbf{x}_{1:\mathrm{d}}) \end{cases}$$



$s_{\theta},\ t_{\theta}$ – произвольные нейросети

Такая архитектура всё равно легко обратима:

$$\begin{cases} \mathbf{x}_{1:\mathrm{d}} = \mathbf{z}_{1:\mathrm{d}} \\ \mathbf{x}_{\mathrm{d}+1:D} = \left(\mathbf{z}_{\mathrm{d}+1:D} - t_{\theta}(\mathbf{z}_{1:\mathrm{d}})\right) \odot e^{-s_{\theta}(\mathbf{x}_{1:\mathrm{d}})} \end{cases}$$

И якобиан имеет блочный нижнетреугольный вид:

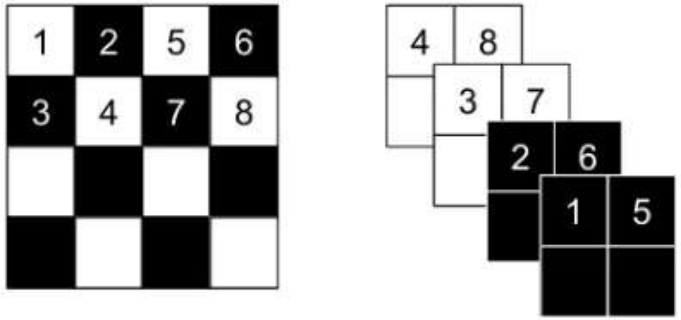
$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{I}_d & \mathbf{0} \\ \dfrac{\partial \mathbf{z}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & diag(e^{s_{\theta}(\mathbf{x}_{1:\mathrm{d}})}) \end{pmatrix}$$

RealNVP

- Для эффективной работы с изображениями в ***RealNVP*** используют иерархическую (***Multi – Scale***) архитектуру

- Это позволяет переходить от анализа мелких деталей к более крупным

Типичный блок устроен следующим образом:



- Сначала применяются несколько связующих слоёв с ***шахматными масками***

- Затем сжимаем $h \times w \times c \to \frac{h}{2} \times \frac{w}{2} \times 4c$

- Применяем связующие слои с ***канальными масками***

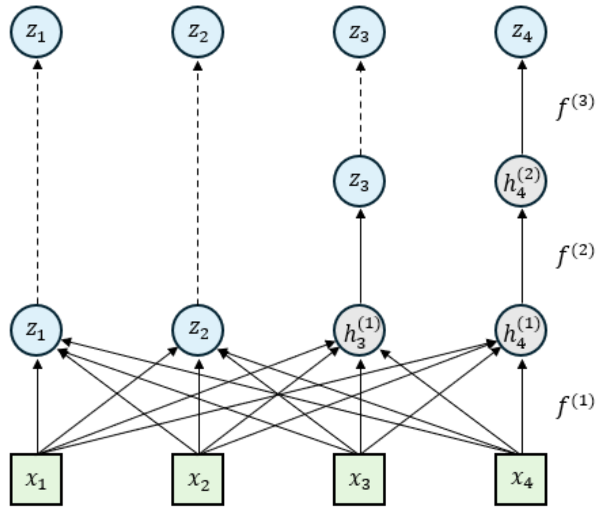
RealNVP

Проблема:

Прогонять весь входной вектор через все слои довольно затратно

Решение:

После каждого блока будем выносить половину переменных из потока сразу в финальный вектор $\boldsymbol{z}$



$RealNVP: Samples$

$Glow$

$Glow: ActNorm$

$\boldsymbol{Glow}$ – идейный наследник $\boldsymbol{RealNVP}$, использующий $\boldsymbol{Affine\ Copling\ Layers}$ и $\boldsymbol{Multi-Scale}$ архитектуру

Авторы решили заменить эвристики $\boldsymbol{RealNVP}$ на более гибкие, обучаемые аналоги

- $\boldsymbol{BatchNorm}$ плохо работает с малыми размерами батчей (шумные статистики)

- Чтобы не зависеть от статистик батча, авторы предложили использовать $\boldsymbol{ActNorm}$ ($\boldsymbol{Activation\ Normalization}$) слой

$Glow$: Invertible 1×1 $Conv$

Проблема:

Фиксированные перестановки в ***RealNVP*** не обучаются и не гарантируют оптимальное смешивание каналов

Решение:

Будем обучать обратимую свёртку 1×1 – линейный поток, который позволит модели самой найти оптимальный способ перемешивания каналов

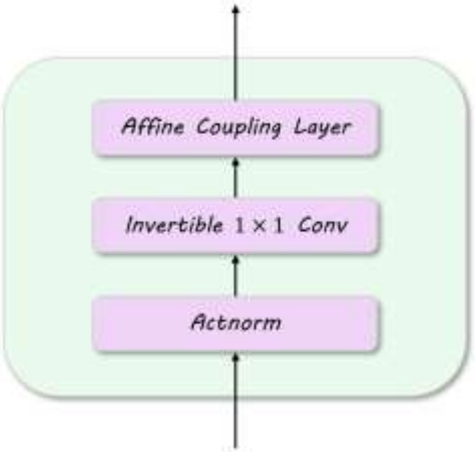
Трюк:

Чтобы быстро считать определитель матрицы свёртки $\boldsymbol{W}$, используется ***LU-разложение***

$Glow$: $Flow\ Step$

$\boldsymbol{Glow}$ объединяет все 3 операции в один обратимый блок ($\boldsymbol{Flow\ Step}$), который многократно применяется в модели:

$$\boldsymbol{ActNorm \rightarrow 1 \times 1\ Conv \rightarrow Affine\ Coupling\ Layer}$$



$\boldsymbol{Multi - Scale}$ $\boldsymbol{архитектура}$:

- Сжимаем $h \times w \times c \rightarrow \frac{h}{2} \times \frac{w}{2} \times 4c$
- Делаем K шагов потока
- Половину каналов выносим сразу в финальный вектор $\boldsymbol{z}$

Glow: *Samples*

Спасибо за внимание!