

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ И ИНФОРМАТИКИ

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

**«НЕЙРОСЕТЕВАЯ РЕКОНСТРУКЦИЯ ПОРИСТЫХ ТЕЛ С СОХРАНЕНИЕМ
ТОПОЛОГИЧЕСКИХ И СТАТИСТИЧЕСКИХ СВОЙСТВ ОБРАЗЦА»**

Выполнил студент
235М группы:
Будакян Я. С.

Научный руководитель:
к.т.н., доц. Грачев Е. А.

Допущена к защите
Зав. кафедрой _____

Москва

2019

Содержание

ВВЕДЕНИЕ	2
1 Нейронные сети	4
1.1 Математическая модель нейрона	4
1.2 Метод обратного распространения ошибки	5
1.3 Сверточные нейронные сети	6
1.4 Генеративные состязательные сети	7
1.4.1 Общая структура	7
1.4.2 Обучение GAN	9
1.4.3 Модификация “pix2pix GAN”	10
2 Методы стохастической оптимизации	12
2.1 Стохастический градиентный спуск	12
2.2 RMSprop	12
2.3 Adam	13
3 Верификация	14
3.1 Функционалы Минковского	14
3.2 Двухточечная корреляционная функция	14
4 Результаты вычислительных экспериментов	15
4.1 Выборки с реализацией одного тренда	15
4.1.1 Выборка 1	15
4.1.2 Выборка 2	16
4.1.3 Выборка 3	17
4.2 Выборки с реализацией различных трендов	19
4.2.1 Выборка 4	19
ВЫВОДЫ	22
ЗАКЛЮЧЕНИЕ	23
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	24
ПРИЛОЖЕНИЕ	26

ВВЕДЕНИЕ

ВВЕДЕНИЕ

1 Нейронные сети

В общем смысле искусственная нейронная сеть - это математическая модель, построенная по принципу организации и функционирования биологических нейронных сетей. Она представляет из себя систему соединенных простых блоков - искусственных нейронов, каждый из которых имеет входы и выходы для взаимодействия с другими нейронами. Главное преимущество нейронных сетей перед традиционными алгоритмами в том, что они обучаются на некотором наборе данных, а не программируются в классическом смысле этого понятия. Процесс обучения заключается в нахождении оптимальных весовых коэффициентов между нейронами. С математической точки зрения, процесс обучения - это задача многопараметрической нелинейной оптимизации.

1.1 Математическая модель нейрона

Одиночный нейрон обычно представляет собой взвешенный сумматор с нелинейной функцией активации на выходе:

$$x_{out} = \phi(\vec{w} \cdot \vec{x}_{in}),$$

где \vec{w} - вектор весовых коэффициентов связей, \vec{x}_{in} - входной вектор, ϕ - нелинейная функция активации (Рис. 1).

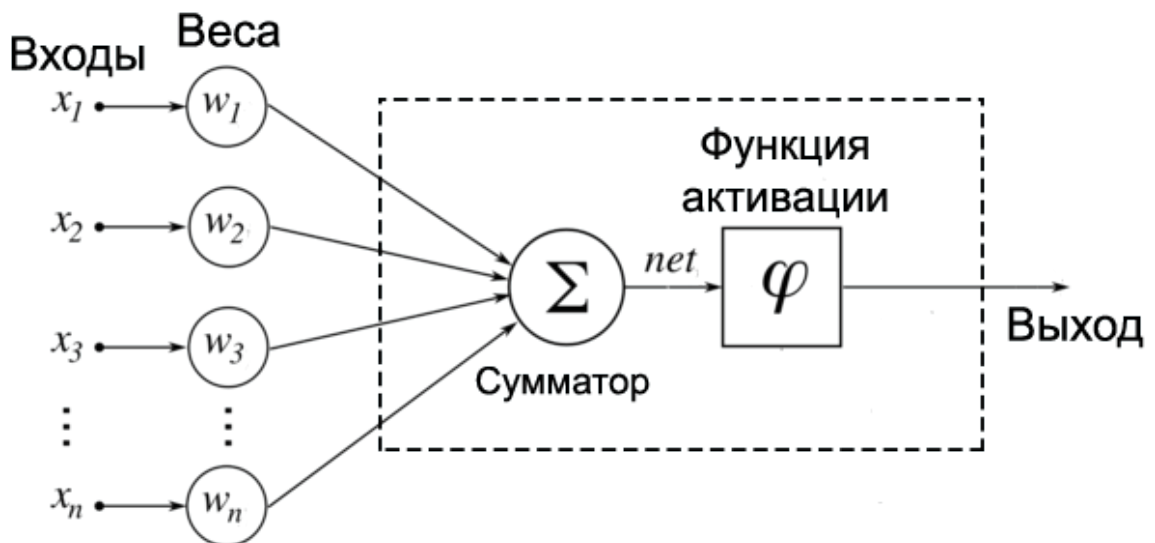


Рис. 1: Математическая модель нейрона

Функция активации может выбираться разной в зависимости от задачи. Наиболее часто используемые функции:

- Сигмоида (логистическая функция)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Гиперболический тангенс

- ReLU

$$ReLU(x) = \max(0, x)$$

- softmax

$$\sigma(\vec{x})_j = \frac{e^{x_j}}{\sum_{k=1}^N e^{z_k}}$$

Множество таких нейронов объединяется в сеть и обучается каким-либо методом оптимизации.

1.2 Метод обратного распространения ошибки

Метод обратного распространения ошибки (backpropagation) - самый широко используемый и успешный алгоритм обучения глубоких (многослойных) нейронных сетей. Суть этого метода заключается в распространении сигналов ошибки от выходов сети к ее входам в обратном к распространению сигнала в сети направлении. Это позволяет вычислить производные ошибки по весам сети, которые потом можно использовать в любом градиентном алгоритме оптимизации (например, в стохастическом градиентном спуске).

Обозначим множество входов сети как $\{x_1, \dots, x_n\}$, множество выходов - O , w_{ij} - вес, присвоенный ребру, соединяющему i -й и j -й узлы, y_k - известные (правильные) ответы, o_i - выход i -го узла. Введем функцию ошибки (например, сумма квадратов расстояний):

$$L(\vec{x}, W) = \frac{1}{2} \sum_{k \in O} (y_k - o_k)^2,$$

где $W = \{w_{ij}\}$ - матрица весовых коэффициентов

Рассмотрим сначала нейроны последнего слоя. Весовой коэффициент w_{ij} влияет на выход сети как часть суммы $S_j = \sum_i w_{ij}x_i$. Соответственно,

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial S_j} \frac{\partial S_j}{\partial w_{ij}} = x_i \frac{\partial L}{\partial S_j}$$

Аналогично, S_j влияет на общую ошибку только в рамках выхода j -го узла o_j , поэтому

$$\frac{\partial L}{\partial S_j} = \frac{\partial L}{\partial o_j} \frac{\partial o_j}{\partial S_j} = \left(\frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in Out} (y_k - o_k)^2 \right) \left(\frac{\partial \phi(S)}{\partial S} \Big|_{S=S_j} \right)$$

Если узел j не находится на последнем слое, то у него есть набор связей с нейронами следующего слоя. Обозначим их множество как K_j . Тогда

$$\frac{\partial L}{\partial S_j} = \sum_{k \in K_j} \frac{\partial L}{\partial S_k} \frac{\partial S_k}{\partial S_j}$$

$$\frac{\partial S_k}{\partial S_j} = \frac{\partial S_k}{\partial o_j} \frac{\partial o_j}{\partial S_j} = w_{jk} \frac{\partial o_j}{\partial S_j}$$

$\frac{\partial L}{\partial S_k}$ - аналогичная поправка, но для нейрона следующего слоя. В итоге, получены выражения для производных ошибки по весам для нейронов выходного слоя, а аналогичные производные для нейронов внутренних слоев выражены через нейроны следующих слоев. Это и есть процесс обратного распространения ошибки - градиенты ошибки по весам вычисляются последовательно, начиная с выходного слоя и заканчивая первым.

1.3 Сверточные нейронные сети

Сверточные нейронные сети (CNN - convolutional neural networks) - это специальная архитектура нейронной сети, нацеленная на эффективное распознавание изображений, впервые предложенная Яном Лекуном [9]. Структура такой сети имеет некоторое сходство со строением зрительной коры головного мозга. Свое название CNN получили из-за наличия сверточных слоев, в которых каждый фрагмент изображения умножается на ядро свертки, полученный результат суммируется и записывается в аналогичную позицию выходного изображения. Одно отдельное ядро свертки обычно интерпретируют как

кодирование какого-либо признака изображения. При этом сами ядра выучиваются сетью самостоятельно, а не закладываются человеком. В CNN чередуются сверточные и субдискретизирующие слои, таким образом более глубокие сверточные слои могут выделять абстрактные детали изображения, вплоть до общих понятий, таких как “кошка”, “собака”, и т.п. На данный момент CNN считаются базовым нейросетевым подходом при работе с изображениями.

1.4 Генеративные состязательные сети

Архитектура нейронной сети, получившая название генеративной состязательной сети (generative adversarial network - GAN), впервые была описана в 2014 году [4]. За последнее время сети такого типа добились больших успехов в задачах синтеза объектов из сложных распределений. Этим объясняется мотивация попытки применения данной архитектуры для решения поставленной задачи.

1.4.1 Общая структура

Переформулируем изначальную задачу нахождения такой процедуры синтеза X' , что $P_{X'} \approx P_X$:

$$\rho(P_{X'}, P_X) \longrightarrow \min_{P_{X'}}$$

Введем параметризованную процедуру генерации:

$$X' = g_{\theta}(\cdot)$$

Получаем:

$$\rho(P_{X'}, P_X) \longrightarrow \min_{P_{X'}}$$

$$\rho(g_{\theta}(\cdot), P_X) \longrightarrow \min_{g_{\theta}(\cdot)}$$

$$\rho(g_{\theta}(\cdot), P_X) \longrightarrow \min_{\theta}$$

Возникает вопрос: что использовать в качестве метрики похожести двух распределений ρ , где одно из распределений задано обучающей выборкой. В качестве такой метрики можно использовать функцию потерь обученного классификатора, потому что естественно предположить, что чем чаще ошибается

обученный классификатор, тем больше одно распределение похоже на другое. Тогда задача примет вид:

$$\rho(P_{X'}, P_X) \longrightarrow \min \Leftrightarrow L \longrightarrow \max,$$

где L - функция потерь обученного классификатора. Соответственно, можно ввести две нейросети:

- $d_\zeta(x)$ - классификатор для измерения расстояния, “дискриминатор”
- $g_\theta(x)$ - сеть, трансформирующая шум в элементы множества X' , “генератор”

Суть использования двух сетей состоит в том, что они обучаются совместно, конкурируя друг с другом: генератор пытается имитировать целевое распределение, а дискриминатор пытается классифицировать поступающие от генератора и из обучающей выборки изображения на 2 класса: реальные (из изначального распределения P_X) и ложные (из $P_{X'}$, т.е. синтезированные генератором). Для дальнейшего рассмотрения введем функцию потерь дискриминатора (например, logloss):

$$l_1 = l(d_\zeta(x), 1)$$

$$l_2 = l(d_\zeta(x'), 0)$$

$$\begin{aligned} L(X, X') &= \frac{1}{2} \mathbb{E}_X l_1 + \frac{1}{2} \mathbb{E}_{X'} l_2 = -\frac{1}{2} (\mathbb{E}_X \log d_\zeta(x) + \mathbb{E}_{X'} \log(1 - d_\zeta(x'))) = \\ &= -\frac{1}{2} (\mathbb{E}_X \log d_\zeta(x) + \mathbb{E}_V \log(1 - d_\zeta(g_\theta(v)))) = L(\zeta, \theta). \end{aligned}$$

Функция потерь обученного классификатора:

$$L^*(\theta) = \min_{\zeta} L(\zeta, \theta)$$

Соответственно,

$$\begin{aligned} \min_{\zeta} L(\zeta, \theta) &\longrightarrow \max_{\theta} \\ \theta^* &= \arg \max_{\theta} \left[\min_{\zeta} L(\zeta, \theta) \right] \end{aligned}$$

Определим оптимальный дискриминатор:

$$d_{\theta}^* = d_{\zeta^*(\theta)}$$

$$\zeta^*(\theta) = \arg \min_{\zeta} L(\zeta, \theta)$$

1.4.2 Обучение GAN

Итак, задача обучения GAN свелась к нахождению

$$\theta^* = \arg \max_{\theta} \left[\min_{\zeta} L(\zeta, \theta) \right]$$

В итоге, процесс обучения принимает следующий вид:

- Обучаем дискриминатор при фиксированном генераторе
- Обучаем генератор при фиксированном дискриминаторе
- Повторяем до сходимости параметров обеих моделей

Описанный процесс схематично изображен на (Рис. 2).

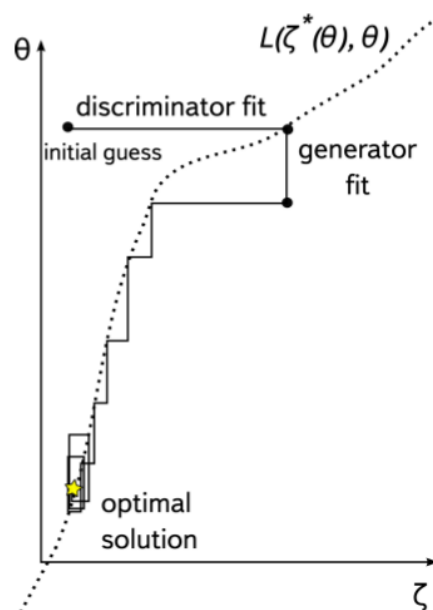


Рис. 2: Схематическое изображение процесса обучения GAN.

1.4.3 Модификация “pix2pix GAN”

Для решения задачи была опробована модификация обычной структуры GAN под названием “pix2pix GAN” [10, 11]. Ее отличие от схемы GAN, введенной выше, состоит в том, что вместо шума на вход генератору приходят другие изображения, на которых он основывается при синтезе. Схематически ее устройство изображено на (Рис. 3).

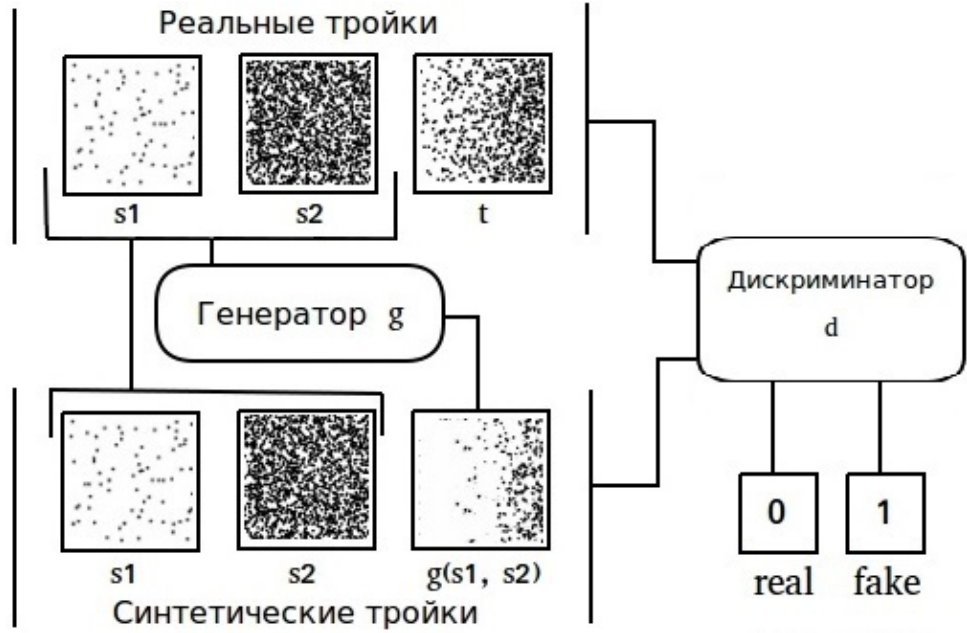


Рис. 3: Схематическое устройство сети pix2pix GAN.

Для pix2pix сети общий функционал потерь выглядит следующим образом:

$$L(G, D) = L_{adv}(G, D) + \eta L1$$

$$L1 = \mathbb{E}_{p_{data}(s_1, s_2, r)}(\| r - G(s_1, s_2) \|_1)$$

$$L_{adv}(G, D) = \mathbb{E}_{p_{data}(s_1, s_2, r)} \log D(s_1, s_2, r) + \mathbb{E}_{p_{data}(s_1, s_2)} \log(1 - D(s_1, s_2, G(s_1, s_2)))$$

где G, D - генератор и дискриминатор, (s_1, s_2, r) - тройка изображений (интенсивность слева, справа и реальное изображение с трендом), $\mathbb{E}_{p_{data}(s_1, s_2, r)}$ - мат. ожидание логарифмического правдоподобия того, что тройка изображений (s_1, s_2, r) принадлежит вероятностному распределению реальных троек $p_{data}(s_1, s_2, r)$, а $p_{data}(s_1, s_2)$ соответствует распределению реальных изображений s_1, s_2 .

В качестве генератора в [10, 11] использовалась сеть “U-Net” [12]. Основное отличие сети “U-Net” от обычной сети архитектуры “encoder-decoder”

заключается в наличии прямых связей между сверточными и разверточными слоями. Использование такого типа генератора позволяло увеличить качество синтезируемых изображений. Схемы сетей типа “U-Net” и “Encoder-decoder” приведены на (Рис. 4).

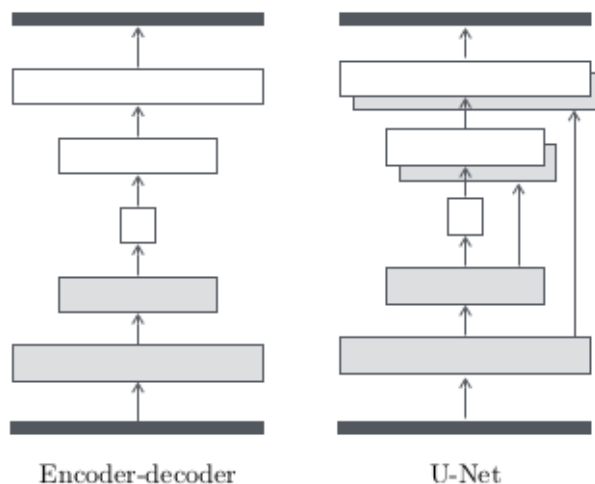


Рис. 4: Схематическое изображение нейросети-генератора.

Разработанные и использованные в экспериментах архитектуры сетей генератора и дискриминатора были такими:

Генератор:

$C[nf]-C[nf*2]-C[nf*4]-C[nf*8]-C[nf*8]-C[nf*8]-C[nf*8]-C[nf*8]-DC[nf*8]-DC[nf*8]-DC[nf*8]-DC[nf*8]-DC[nf*4]-DC[nf*2]-DC[nf]-DC[1]$

Под $C[nf]$ или $DC[nf]$ здесь подразумеваются блоки, состоящие из сверточного или разверточного слоя с указанным числом фильтров, батч-нормализации и функции активации LeakyRELU с коэффициентом 0.2.

Дискриминатор:

$C[nf]-C[nf*2]-C[nf*4]-C[nf*8]-C[1]$

В дискриминаторе батч-нормализация не применялась.

При использовании “U-Net”, к указанной выше архитектуре генератора добавлялись дополнительные сквозные связи между соответствующими сверточными и разверточными слоями.

2 Градиентные методы оптимизации

Для обучения глубоких нейросетей применяется метод обратного распространения ошибки, который позволяет рассчитать градиенты весов, и различные алгоритмы стохастической оптимизации. В данной работе для обучения моделей применялось два алгоритма оптимизации под названиями “RMSprop” и “Adam”, являющихся некоторыми модификациями стохастического градиентного спуска (SGD).

2.1 Градиентный спуск

вав

2.2 Стохастический градиентный спуск

Описание стохастического градиентного спуска есть в [13]. Стохастический градиентный спуск обновляет каждый параметр путем вычитания градиента целевой функции по соответствующему параметру и умножая его на гиперпараметр λ - шаг обучения. Градиент целевой функции подсчитывается не на всем наборе данных, а на его случайном подмножестве.

$$i \sim \mathcal{U}\{1, 2, \dots, n\}$$

$$\theta_{t+1} = \theta_t - \lambda \nabla L_i(\theta_t),$$

где L_i - целевая функция, вычисленная на i -ой части данных (mini-batch), индекс i выбирается случайно.

2.3 RMSprop

RMSprop (root mean square propagation) описан в [14]. Идея заключается в том, что для каждого параметра градиент перемасштабируется, учитывая прошлые значения градиентов для этого параметра. Производится это путем деления градиента на нормировочный множитель, который является корнем из среднего квадрата градиентов.

$$g_{t+1} = \gamma g_t + (1 - \gamma)(\nabla L_i(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\lambda \nabla L_i(\theta_t)}{\sqrt{g_{t+1} + \epsilon}}$$

ϵ - это небольшая константа, введенная для численной стабильности.

2.4 Adam

Adam расшифровывается как adaptive moment estimation. Описание этого алгоритма дано в [15]. Этот алгоритм, помимо перемасштабирования, использует инерцию градиента, что позволяет смягчить быстрое изменение градиента, присущее стохастическому градиентному спуску.

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla L_i(\theta_t)$$

$$v_{t+1} = \beta_2 g_t + (1 - \beta_2) (\nabla L_i(\theta_t))^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\lambda \hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

Авторы статьи [15] пишут, что этот алгоритм достаточно устойчив к неоптимальному выбору параметров, поэтому во многих статьях в начале для обучения пробуют применить именно Adam.

3 Верификация

Верификация синтеза

3.1 Функционалы Минковского

bla bla bla

3.2 Двухточечная корреляционная функция

another bla

4 Результаты вычислительных экспериментов

Архитектуры, описанные в пункте 1.4.3, были реализованы в виде компьютерных программ на языке Python с помощью библиотеки для построения искусственных нейронных сетей Keras [16], которая, в свою очередь, использует для расчетов пакет Tensorflow [17]. Используемые версии программных пакетов указаны в Приложении 1. Обучение проводилось на синтетических данных, сгенерированных самостоятельно реализованным генератором.

4.1 Выборки с реализацией одного тренда

4.1.1 Выборка 1

Выборка состояла из 3000 обучающих троек, 500 валидационных и 50 тестовых. Частицами среды были круги диаметром 7 пикселей. Все изображения содержали в себе различные случайные реализации одного тренда интенсивности

$$\lambda(x) = 0.2 + 0.01875x : \lambda_i = 0.2, \quad \lambda_f = 5$$

Отличия в параметрах нейросетей указаны в (Таб. 1). Для всех обученных моделей параметр η , отвечающий за баланс между L_{adv} и $L1$, был равен 100.

Сеть	Число фильтров на первом слое	Сеть-генератор
nf8	8	U-Net
nf16	16	U-Net
nf16woU	16	Encoder-decoder
nf32	32	U-Net

Таблица 1: Отличия в параметрах моделей (Выборка 1)

Примеры синтеза, полученные с помощью нейросетей с различными параметрами, приведены в (Таб. 2).

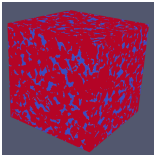
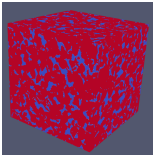
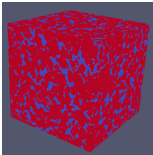
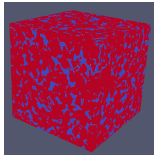
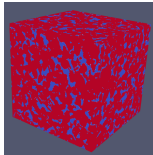
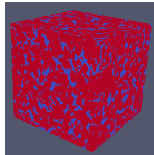
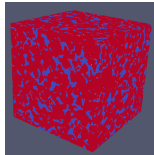
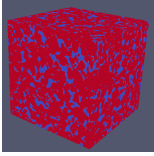
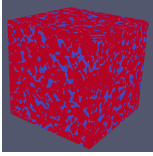
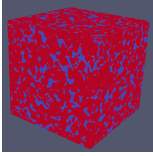
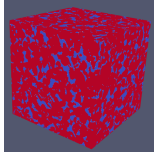
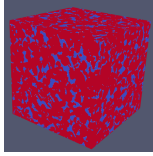
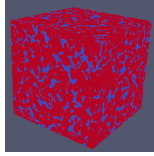
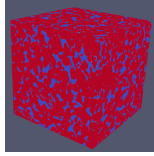
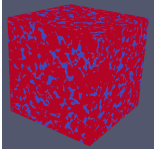
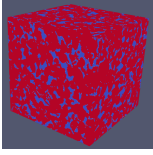
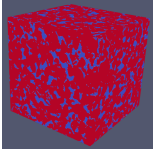
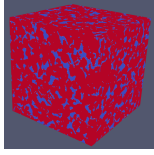
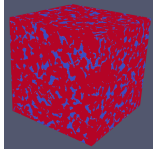
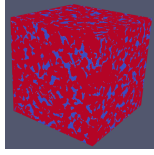
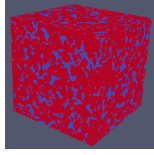
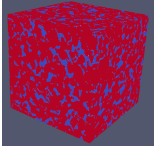
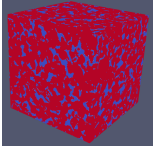
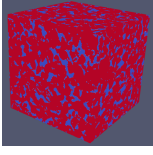
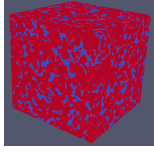
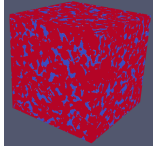
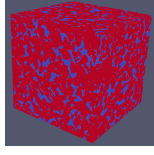
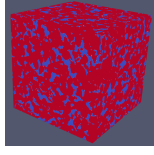
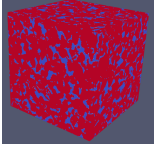
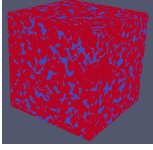
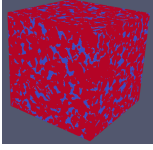
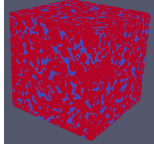
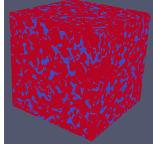
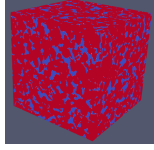
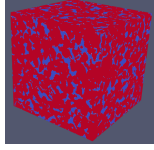
Вход 1	Вход 2	Тренд	Сеть nf8	nf16	nf16woU	nf32
						
						
						
						
						

Таблица 2: Примеры синтеза (Выборка 1)

График приближения тренда различными сетями показан на (Рис. 5).

Рис. 5: Аппроксимация тренда различными сетями (Выборка 1)

Значения введенной метрики для различных сетей указаны в (Таб. 3).

Сеть	Метрика
nf16woU	0.24048
nf8	0.22511
nf16	0.18844
nf32	0.14589

Таблица 3: Значения метрики для разных сетей (меньше - лучше)

Лучшей по введенной метрике оказалась сеть nf32.

4.1.2 Выборка 2

Выборка состояла из 6000 обучающих троек, 316 валидационных и 50 тестовых. Частицами среды были квадраты стороной 3 пикселя. Все изображения содержали в себе различные случайные реализации одного тренда интенсивности

$$\lambda(x) = 1 + 0.0546875x : \lambda_i = 1, \quad \lambda_f = 15$$

Отличия в параметрах обученных нейросетей указаны в (Таб. 4). Во всех обученных моделях в качестве сети-генератора использовалась сеть “U-Net”.

Сеть	Число фильтров на первом слое	η	Метод оптимизации
nf32e5	32	5	Adam
nf64e1	64	1	RMSprop
nf64e5	64	5	RMSprop
nf64e10	65	10	RMSprop

Таблица 4: Отличия в параметрах моделей (Выборка 2)

Примеры синтеза, полученные с помощью нейросетей с различными параметрами, приведены в (Таб. 5).

Вход 1	Вход 2	Тренд	Сеть	nf64e1	nf64e5	nf64e10
			nf32e5			

Таблица 5: Примеры синтеза (Выборка 2)

График приближения тренда различными сетями показан на (Рис. 6).

Рис. 6: Аппроксимация тренда различными сетями (Выборка 2)

Значения введенной метрики для различных сетей указаны в (Таб. 6).

Сеть	Метрика
nf64e10	0.11168
nf64e5	0.06501
nf32e5	0.04827
nf64e1	0.01393

Таблица 6: Значения метрики для разных сетей (меньше - лучше)

Лучшей по введенной метрике оказалась сеть nf64e1.

4.1.3 Выборка 3

Выборка состояла из 6000 обучающих троек, 316 валидационных и 50 тестовых. Частицами среды были единичные пиксели. Все изображения содержали в себе различные случайные реализации одного тренда интенсивности

$$\lambda(x) = 1 + 0.19140625x : \lambda_i = 1, \quad \lambda_f = 50$$

Отличия в параметрах обученных нейросетей указаны в (Таб. 7). Во всех обученных моделях в качестве сети-генератора использовалась сеть “U-Net”. В качестве метода оптимизации использовался Adam.

Сеть	Число фильтров на первом слое	η
nf64e1	64	1
nf64e5	64	5
nf100e0	100	0
nf100e5	100	5

Таблица 7: Отличия в параметрах моделей (Выборка 3)

Примеры синтеза, полученные с помощью нейросетей с различными параметрами, приведены в (Таб. 8).

Вход 1	Вход 2	Тренд	Сеть nf64e1	nf64e5	nf100e0	nf100e5

Таблица 8: Примеры синтеза (Выборка 3)

График приближения тренда различными сетями показан на (Рис. 7).

Рис. 7: Аппроксимация тренда различными сетями (Выборка 3)

Значения введенной метрики для различных сетей указаны в (Таб. 9).

Сеть	Метрика
nf64e5	3.13633
nf100e0	0.1178
nf100e5	0.08266
nf64e1	0.08139

Таблица 9: Значения метрики для разных сетей (меньше - лучше)

Лучшей по введенной метрике оказалась сеть nf64e1. Однако видно, что на этой выборке сети не смогли распознать тренд, и сошлись примерно к средней по выборке интенсивности (не учитывая сети nf64e5, явно являющейся выбросом).

4.2 Выборки с реализацией различных трендов

4.2.1 Выборка 4

Выборка состояла из 6000 обучающих троек, 316 валидационных и 50 тестовых. Частицами среды были единичные пиксели. Все изображения содержали в себе различные случайные реализации разных трендов интенсивности $\lambda(x)$:

$$\lambda_{init}, \lambda_{final} \sim \mathcal{U}\{0, 50\},$$

то есть, значения λ_{init} и λ_{final} для каждого отдельного экземпляра выбирались случайно.

Отличия в параметрах обученных нейросетей указаны в (Таб. 10). Во всех обученных моделях в качестве сети-генератора использовалась сеть “U-Net”. Каждая сеть была обучена как с помощью Adam, так и с помощью RMSprop.

Сеть	Число фильтров на первом слое	η
nf32e5	32	5
nf32e10	32	10
nf32e20	32	20
nf32e50	32	50

Таблица 10: Отличия в параметрах моделей (Выборка 3)

Примеры синтеза, полученные с помощью нейросетей с различными параметрами, приведены в (Таб. 11).

Вход 1	Вход 2	Тренд	Сеть	nf32e10	nf32e20	nf32e50
			nf32e5			

Таблица 11: Примеры синтеза (Выборка 4)

График приближения тренда различными сетями показан на (Рис. 8).

Рис. 8: Аппроксимация тренда различными сетями (Выборка 4)

Значения введенной метрики для различных сетей указаны в (Таб. 12).

Лучшей по введенной метрике оказалась сеть nf32e10. Однако на этой выборке тоже видно, что сети не смогли распознать тренд и сошлись в локальный минимум, примерно соответствующий интенсивности на правом крае.

Поскольку все сети были обучены дважды двумя разными оптимизаторами с одинаковым коэффициентом обучения в течение 10 эпох каждая, то можно провести побочное сравнение между Adam и RMSprop (Таб. 13).

Сеть	Метрика
nf32e5	1.27751
nf32e50	0.84558
nf32e20	0.64971
nf32e10	0.40388

Таблица 12: Значения метрики для разных сетей (меньше - лучше)

Сеть	G-loss Adam	G-loss RMSprop
nf32e5	5.30946	7.01412
nf32e10	3.60198	5.75309
nf32e20	7.82371	10.14108
nf32e50	14.55342	15.99096

Таблица 13: Значение функции потерь сети-генератора на 10-й эпохе

Видно, что во всех случаях Adam достиг меньших значений функции потерь.

ВЫВОДЫ

ВЫВОДЫ

ЗАКЛЮЧЕНИЕ

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge: “Texture Synthesis Using Convolutional Neural Networks” // arXiv: 1505.07376 [cs.CV], 2015.
- [2] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, Victor Lempitsky: “Texture Networks: Feed-forward Synthesis of Textures and Stylized Images” // arXiv: 1603.03417 [cs.CV], 2016.
- [3] Воронцов К. В.: “Математические методы обучения по прецедентам (теория обучения машин)”.
- [4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio: “Generative Adversarial Nets” // arXiv: 1406.2661 [stat.ML], 2014.
- [5] Mehdi Mirza, Simon Osindero: “Conditional Generative Adversarial Nets” // arXiv: 1411.1784 [cs.LG], 2014.
- [6] Jon Gauthier: “Conditional generative adversarial nets for convolutional face generation”, Tech. rep., 2015.
- [7] Junbo Zhao, Michael Mathien, Yann LeCun: “Energy-based Generative Adversarial Networks” // arXiv: 1609.03126 [cs.LG], 2016.
- [8] David Berthelot, Thomas Schumm, Luke Metz: “BEGAN: Boundary Equilibrium Generative Adversarial Networks” // arXiv: 1703.10717 [cs.LG], 2017.
- [9] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: “Backpropagation applied to handwritten zip code recognition” // Neural Comput. 1(4), 541–551, 1989.
- [10] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros: “Image-to-Image Translation with Conditional Adversarial Networks” // arXiv: 1611.07004 [cs.CV], 2016.
- [11] Pedro Costa, Adrian Galdran, Maria Inês Meyer, Michael David Abràmoff, Meindert Niemeijer, Ana Maria Mendonça, Aurélio Campilho: “Towards Adversarial Retinal Image Synthesis” // arXiv: 1701.08974 [cs.CV], 2017.

- [12] Olaf Ronneberger, Philipp Fischer, Thomas Brox: “U-Net: Convolutional Networks for Biomedical Image Segmentation” // arXiv: 1505.04597 [cs.CV], 2015.
- [13] Amari, Shunichi: “A theory of adaptive pattern classifiers” // Electronic Computers, IEEE Transactions on 3, стр. 299-307, 1967.
- [14] Tieleman, Tijmen, Geoffrey Hinton: “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude” // Coursera: Neural Networks for Machine Learning 4: 2, 2012.
- [15] Diederik P. Kingma, Jimmy Lei Ba: “Adam: A method for stochastic optimization” // arXiv:1412.6980 [cs.LG], 2014.
- [16] François Chollet: Keras, 2015. Software available from <http://keras.io/>.
- [17] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng: “TensorFlow: Large-scale machine learning on heterogeneous systems”, 2015. Software available from <http://tensorflow.org/>.
- [18] Martin Arjovsky, Soumith Chintala, Léon Bottou : “Wasserstein GAN” // arXiv: 1701.07875 [stat.ML], 2017

ПРИЛОЖЕНИЕ

Приложение 1. Использованные версии пакетов

- Python 3.7.2
- PyTorch 1.0.1

Приложение 2. Дополнительные примеры генерации КАРТИНКИ