Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

ОТЧЕТ
О КУРСОВОЙ РАБОТЕ

по теме: Реализация алгоритма анализа межслойной трансформации данных внутри нейросетей посредством персистентных гомологий внутрислойных многообразий для полносвязных нейронных сетей

по дисциплине: Глубокое обучение

Специальность: 09.03.03 Мобильные и сетевые технологии

Проверил: Томилов Иван

Дата: «23» декабря 2022г.

Оценка _____

Выполнили:
студенты группы K34401
Кулёмин Семён

Санкт-Петербург

2022 г

Генерация искусственных данных.

Для работы с нейросетью сгенерируем датасет с двумя вложенными окружностями с радиусами (r1, r2, r1 < r2). Количество элементов - 2N, размерность - d.

Код:

```
r1 = 3
r2 = 6
n = 10000
d = 10

# 1) Генерируем с помощью numpy.random.randn два массива размерности (N,
# d) -первый будет отвечать внутренней сфере радиуса r1, а второй - внешней
# с радиусом r2
sphere1 = np.random.randn(n, d)
sphere2 = np.random.randn(n, d)

# 2) Для каждого из массивов вычисляем поэлементную сумму квадратов
# (вдоль axis = 1 с помощью np.square и np.sum), после чего вычисляем
# поэлементный квадратный корень (через np.sqrt), после чего делим
# исходный массив на получившуюся вещь. Данное заклинание - ничто иное,
# как поэлементная нормировка массивов.
#sphere1 = np.divide(sphere1.T,np.sqrt(np.sum(np.square(sphere1), axis =
1))).T
#sphere2 = np.divide(sphere2.T,np.sqrt(np.sum(np.square(sphere2), axis =
1))).T
sphere1 = sphere1 / np.sqrt(np.sum(np.square(sphere1), axis = 1,
keepdims=True))
sphere2 = sphere2 / np.sqrt(np.sum(np.square(sphere2), axis = 1,
keepdims=True))

# 3) Первый массив умножаем на r1, а второй - на r2.
sphere1 = sphere1 * r1
sphere2 = sphere2 * r2

# 4) К обоим массивам добавляем гауссов шум (np.random.randn с
соответствующими размерностями)
# со средним (mean) 0 и дисперсией (std) np.abs((r2-r1)/4.) - таким образом,
# получившиеся данные с нужной нам вероятностью не будут пересекаться.
sphere1 = sphere1 + np.random.normal(0, np.abs((r2-r1)/4.), size=(n, d))
sphere2 = sphere2 + np.random.normal(0, np.abs((r2-r1)/4.), size=(n, d))
```

```
# 5) Теперь первому массиву присваиваем класс 0, а второму – класс 1 в любом удобном формате.
df = pd.DataFrame(np.concatenate((sphere1, sphere2), axis = 0))
df.loc[:n-1, d] = 0
df.loc[n-1:, d] = 1
```

Для более точной работы НС, перемешаем строки полученного датасета:

```
from sklearn.utils import shuffle
df = shuffle(df)
```

Полученный датасет:

```
df
```

	0	1	2	3	4	5	6	7	8	9	10
1660	-0.809352	-0.069266	-0.772267	0.988481	-0.086579	-0.051168	0.111427	2.093245	0.765947	-1.857988	0.0
11479	2.307585	-1.070957	-0.933675	1.502535	2.365834	-3.482796	3.480935	0.994890	2.134678	3.114922	1.0
8159	1.856172	0.141667	-0.125183	-0.215562	-1.156439	-2.226766	-0.602074	1.161856	0.023069	0.126729	0.0
10859	-0.598674	-2.436558	0.259183	-1.048992	-2.434763	-1.056303	-2.095217	1.654031	1.814695	-1.811447	1.0
15111	4.720262	-1.112246	1.035963	-0.784196	1.019091	-2.848365	-0.425561	2.443662	-0.228733	0.470662	1.0
...	...	...	...	...	...	...	...	...	...	...	...
2317	1.437151	1.042878	-0.715548	0.981581	-1.788381	1.216319	-1.382056	-1.804190	-0.683321	1.092493	0.0
11820	2.887970	-1.172404	1.996983	-0.890380	2.311310	-3.216570	2.239314	0.163564	0.778829	-0.219766	1.0
5809	0.983806	-0.061248	-0.637898	0.605854	-1.693639	1.306929	-1.219570	0.137053	-1.589694	-2.113697	0.0
18376	-1.283216	0.489067	2.118021	0.930456	2.247446	-0.514577	3.539090	0.086320	-4.096264	0.801887	1.0
10462	-1.932810	-0.250599	2.256764	0.120808	3.694423	-3.041607	-1.500389	-2.762503	1.045732	2.376497	1.0

20000 rows × 11 columns

Разделим данные на обучающую и тестовую выборку:

```
from sklearn.model_selection import train_test_split
X = df.loc[:, :d-1]
y = df.loc[:, d:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Построение архитектуры нейронной сети

Для изучения трасформации данных построим две модели нейронной сети:

- Модель, способная качественно решить задачу классификации данных
- Модель, которая не может классифицировать данные

Для работы были построены две модели полносвязных нейронных сетей.

Код для создания первой модели:

```
def get_basic_model():
  model = tf.keras.Sequential([
    normalizer,
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(5, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam',
                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=['accuracy'])
  return model

model = get_basic_model()
```

Архитектура первой модели:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 10)               21
 n)
 dense (Dense)               (None, 10)                110
 dense_1 (Dense)             (None, 8)                 88
 dense_2 (Dense)             (None, 5)                 45
 dense_3 (Dense)             (None, 1)                 6
=================================================================
Total params: 270
Trainable params: 249
Non-trainable params: 21
```

_____

Модель решает задачу классификации, после обучения точность результата составляет 93%.

Код для создания второй модели:

```
def get_another_model():
  model = tf.keras.Sequential([
    normalizer,
    tf.keras.layers.Dense(2, activation='relu'),
    tf.keras.layers.Dense(3, activation='relu'),
    tf.keras.layers.Dense(2, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam',
                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=['accuracy'])
  return model

model = get_another_model()
```

Архитектура второй модели:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 10)               21
 n)
 dense (Dense)               (None, 2)                 22
 dense_1 (Dense)             (None, 3)                 9
 dense_2 (Dense)             (None, 2)                 8
 dense_3 (Dense)             (None, 1)                 3
=================================================================
Total params: 63
Trainable params: 42
Non-trainable params: 21
_________________________________________________________________
```

Модель не способна решить задачу классификации, после обучения точность модели - не более 50%.

Нормализация входных даннных:

```
normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(X_train)
```

Обучение модели:

```
model.fit(X_train, y_train, epochs=EPOCHS_NUM, batch_size=BATCH_SIZE)
```

Активации слоёв сохранялись следующим образом:

```
output_names = [l.name for l in model.layers]
model.outputs = [l.output for l in model.layers]
model.build(input_shape=X_train.shape)
output_values = model(X_train)
layer_name_to_output_value = dict(zip(output_names, output_values))
```

Оценка работы модели:

```
model.evaluate(X_test, y_test)
```

Персистентые гомологии

Для рассчёта персистентых гомологий использовалась библиотека giotto-tda.

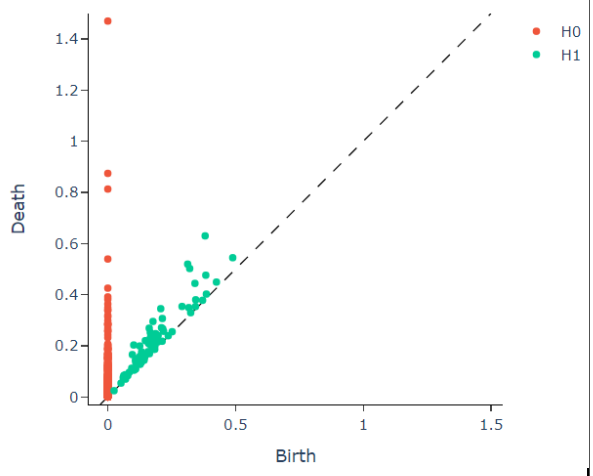
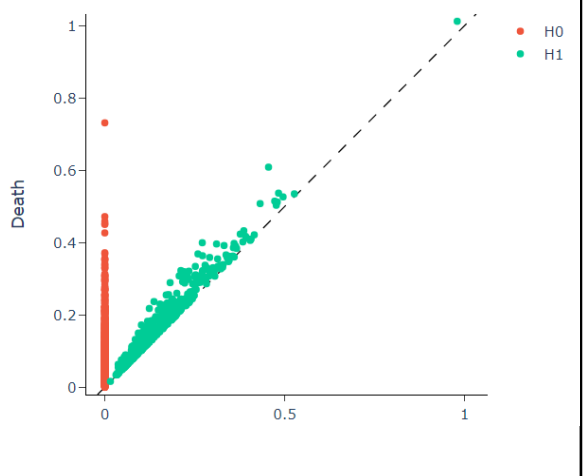
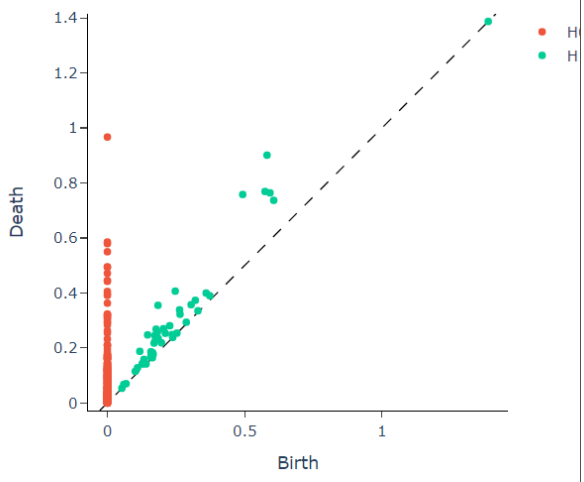
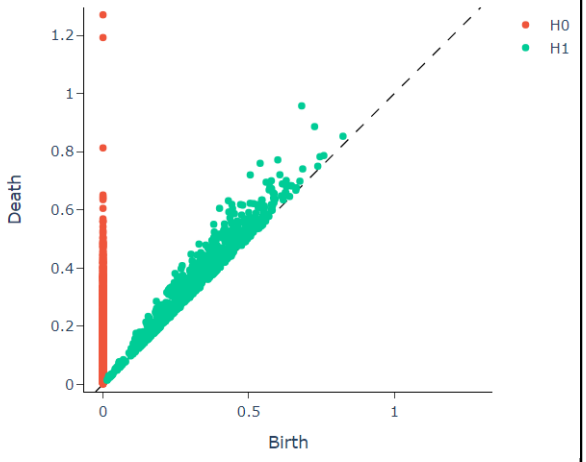
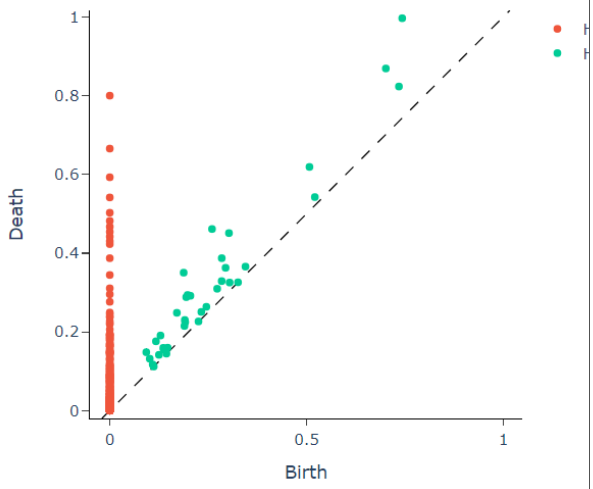
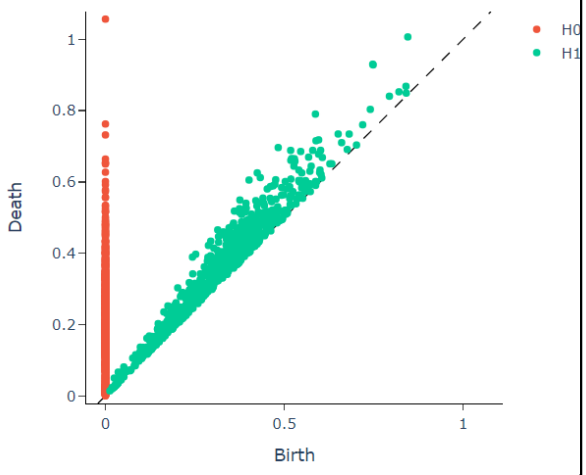
Были сохранены активации слоёв двух нейросетей и при помощи VietorisRipsPersistence были рассчитаны гомологии и постоены диаграммы персистентности.

Каждая модель обучалась в течении 25 эпох, активации сохранялись в начале обучения, после 10 и 25 эпох.

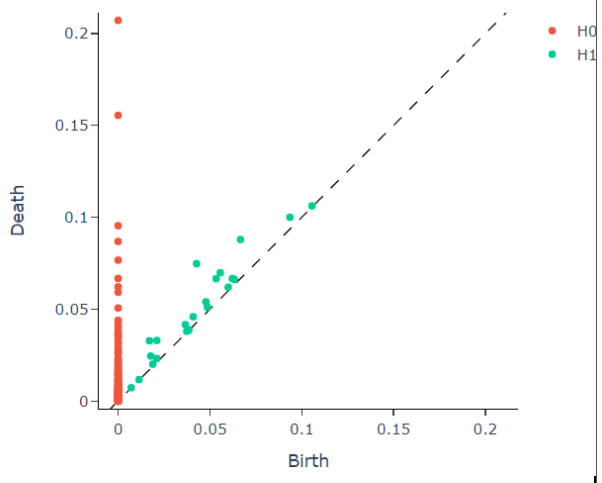
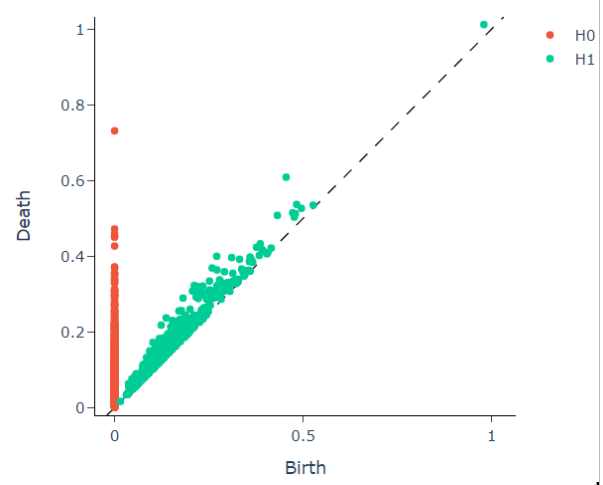
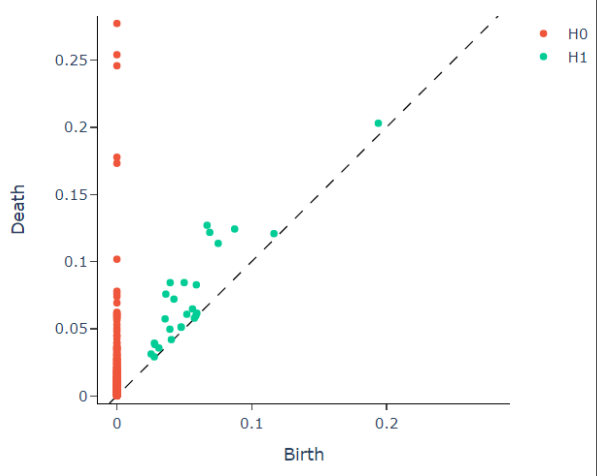
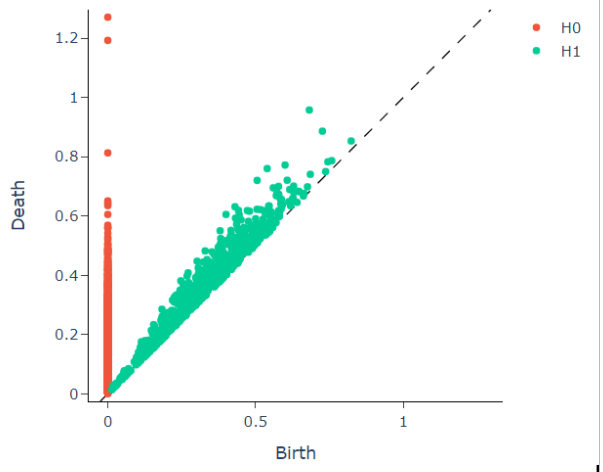
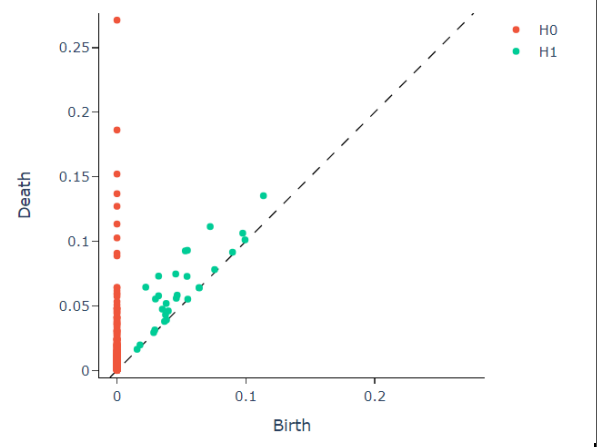
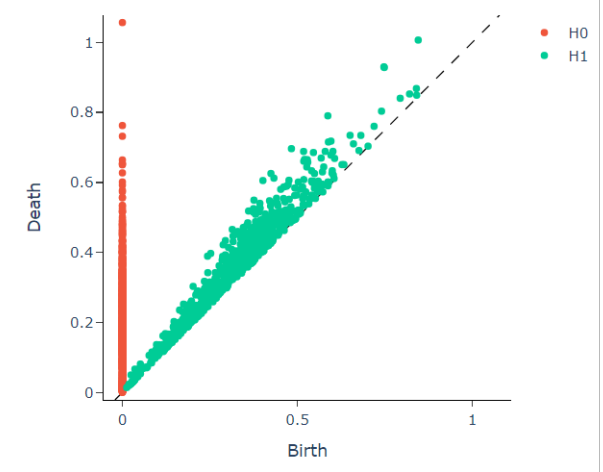
```
layer_name = ''
activation_1 = layer_name_to_output_value[layer_name]
activation_2 = layer_name_to_output_value_1[layer_name]
activation_3 = layer_name_to_output_value_2[layer_name]

VR = VietorisRipsPersistence()
diagrams = VR.fit_transform([activation_1, activation_2, activation_3])
```

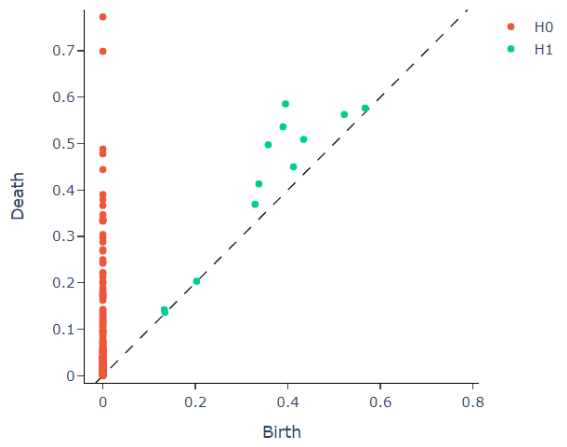
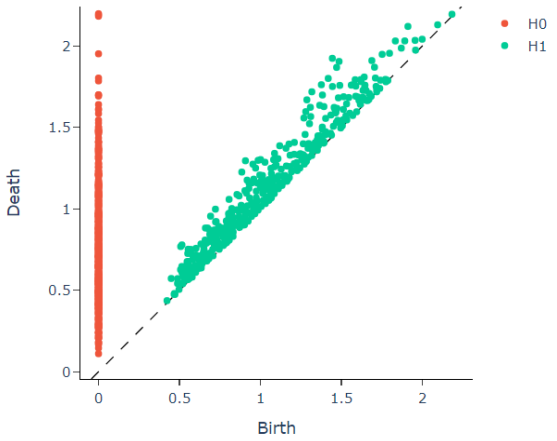
Диаграммы персистентности для 1 слоя (Train dataset)

	Плохая НС	Хорошая НС
начало		
10 эпоха		
25 эпоха		

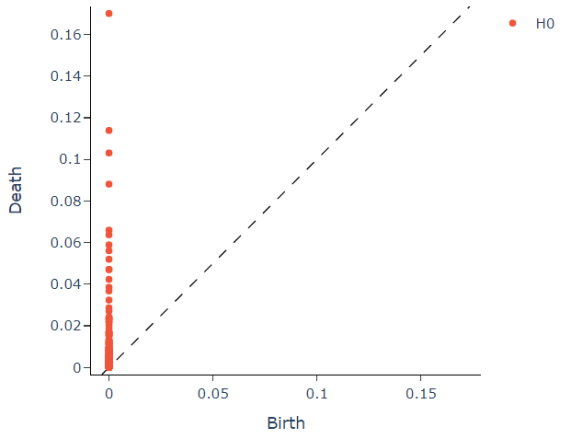
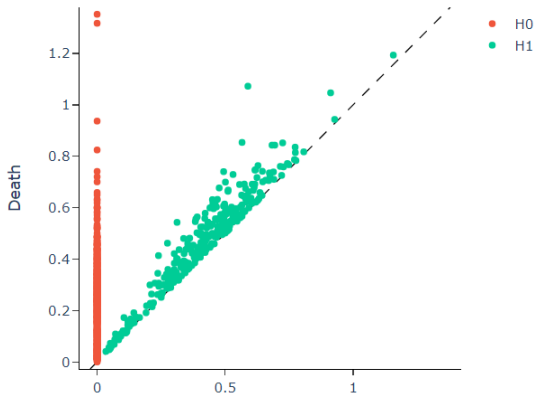
Диаграммы персистентности для 3 слоя (Train dataset)

	Плохая НС	Хорошая НС
начало		
10 эпоха		
25 эпоха		

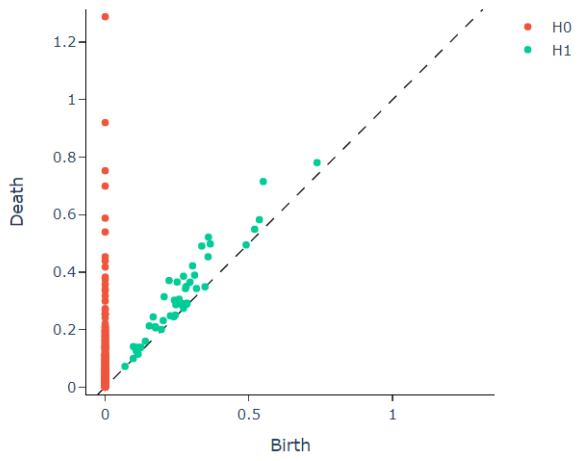
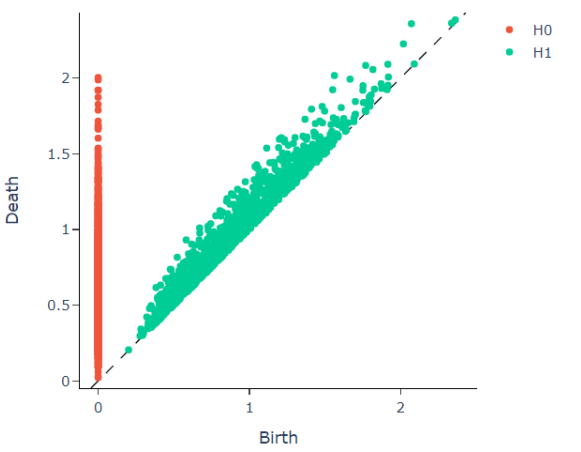
Диаграммы персистентности для 1 слоя (Test dataset)

Плохая НС	Хорошая НС
	

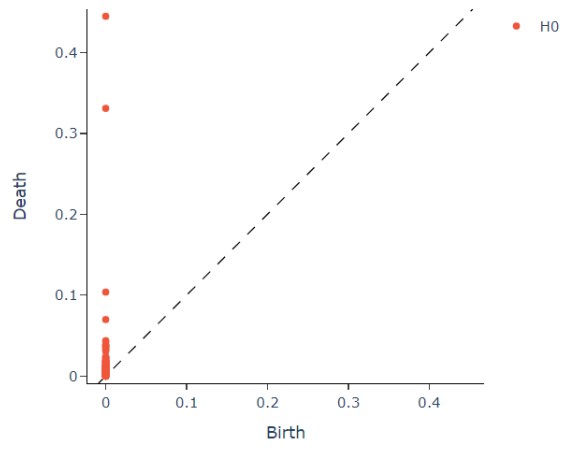
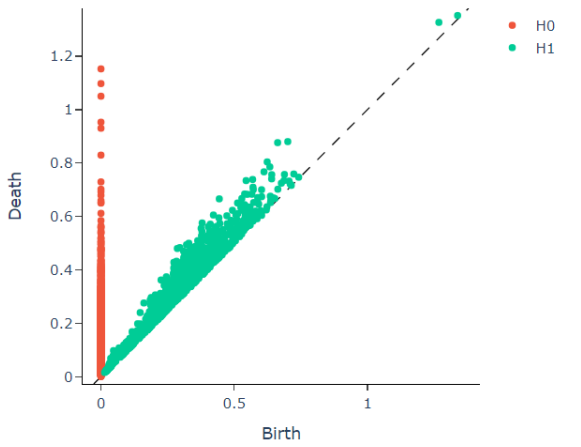
Диаграммы персистентности для 3 слоя (Test dataset)

Плохая НС	Хорошая НС
	

Диаграммы персистентности для 1 слоя (Train+Test dataset)

Плохая НС	Хорошая НС
	

Диаграммы персистентности для 3 слоя (Train+Test dataset)

Плохая НС	Хорошая НС
	

Анализ диаграмм персистнентности

Описание диаграмм:

- красные точки - нульмерные гомологии (компоненты связности)
- зеленые точки - одномерные гомологии (петли)

Нульмерные гомологии рождаются при e = 0 и постепенно умирают с увеличением e, в то время как одномерные гомологии появляются при e > 0 и живут гораздо меньше (зеленые точки расположены близко к прямой Birth = Death).

Диаграммы персистентности двух моделей сильно отличаются. При обучении в “хорошей” модели выделяется гораздо больше одномерных гомологий. Также важно отметить, что гомологии в “плохой” жизнь гомологий короче, как и их масштаб. На третьем слое в “плохой” нейросети одномерные гомологии живут до 0.1-0.2, в то время как в “хорошей” нейросети до 0.7-1.

При построении диаграмм для тестовой выборки в “хорошей” нейросети гомологии живут в среднем до 1.4, а в “плохой” НС на первом слое одномерные гомологии перестают формировать после 0.6, а на третьем слое не формируются вообще.

При построении диаграмм для полной выборке результаты аналогичные - масштаб в“хорошей” НС гораздо выше, чем в “плохой”.

“Плохая” нейросеть неспособна трансофрмировать пространство из-за недостаточного количества нейронов, тем самым не справляется с поставленной задачей. Размерность сферы может быть уменьшена и преобразована в окружность, однако для выполнения этого при помощи нейросети необходимо достаточное количество нейронов, которое было у “хорошей” нейросети.

Выводы

Персистентные голологии помогли проанализировать трансформацию данных внутри полносвязных нейронных сетей, продемонстрировать разницу работы двух схожих по архитектуре НС.