Лабораторная работа №3 по ИИС

Составил: доц. М.А. Бакаев, каф. АСУ НГТУ.

Введение

Цель работы: получить навыки практической реализации и обучения нейронных сетей для решения задач машинного обучения, а также представление о влиянии гиперпараметров и архитектуры.

1. Нейронные сети (краткая теоретическая справка)

Основные типы задач, которые решаются с использованием **искусственных нейронных сетей** (НС) – это классификация и регрессия. По сравнению с некоторыми другими методами машинного обучения, НС обладают рядом преимуществ: возможностью самостоятельно определять значимые для решения задачи факторы, способностью к обучению и дообучению, устойчивостью к «шуму» в данных, универсальность. К недостаткам НС относят высокую требовательность к вычислительным ресурсам, к объемам и разнообразию обучающих данных, но вместе с тем опасность «переобучения». Кроме того, результаты, получаемые НС, с трудом поддаются объяснению с точки зрения человека.

Исторически, первой широко известной моделью НС стал перцептрон, в котором поступающие от датчиков сигналы передаются ассоциативным элементам, а затем реагирующим элементам. Выход перцептрона – это линейная комбинация значений на его входах (x_i) и соответствующих весов в «связях» (w_i) – см. Рис. 1. Используемые в настоящее время НС, как правило, имеют **входной слой**, **выходной слой** и произвольное количество промежуточных (**скрытых**) слоёв.

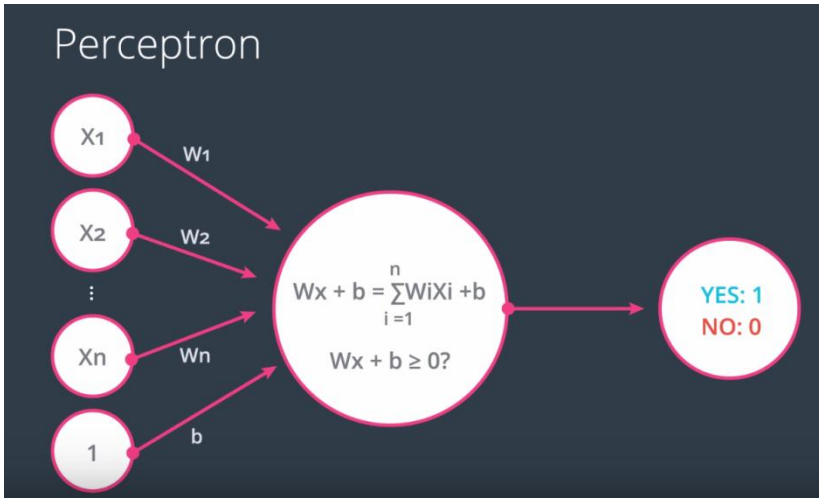


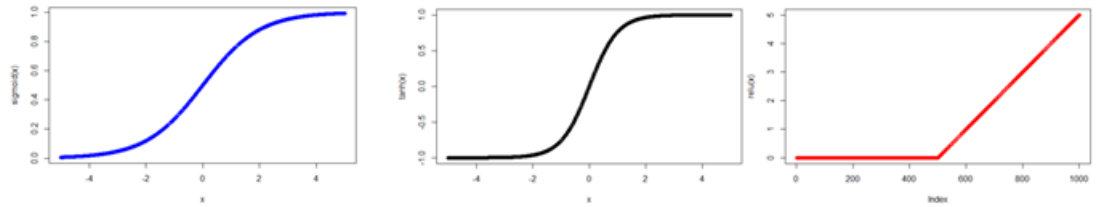
Рис. 1. Принципиальная схема классического перцептрона.

В современных НС узлами являются искусственные **нейроны**, распределённые по слоям НС. Нейроны фактически реализуют некоторую математическую функцию (**функцию активации**) от линейной комбинации всех входных сигналов. Полученный результат пересылается на единственный имеющийся у нейрона выход. Нейроны классифицируют на основе их положения в топологии сети: входные, выходные (таковой часто один) и нейроны в промежуточных слоях. В большинстве классических архитектур НС связи имеются только между нейронами предыдущего и последующего слоя, но не внутри слоя.

1.1. Настройки нейросетевой модели

В отличие от других алгоритмов машинного обучения, где главной составляющей успеха является выбор входных параметров, при создании нейросетевой модели основной задачей ИИ-инженера является подбор характеристик НС. Хотя в последнее время появились алгоритмы, способные к автоматизированному подбору таких характеристик, они обладают высокой вычислительной сложностью, поэтому опыт инженера по-прежнему востребован. К характеристикам НС относят:

- Собственно коэффициенты модели (**параметры**). Их итоговые значения устанавливаются алгоритмом обучения НС, который оптимизирует их так, чтобы итоговая ошибка модели была минимальной. Тем не менее, важной моментом может быть инициализация – определение начальных значений параметров, которая позволили бы более быстрое выполнение алгоритма (сокращая время обучения). Одной из **стратегий инициализации** является «перенос знаний» (transfer learning) из существующих моделей. Например, имеющаяся НС, обученная предсказывать возраст людей по фотографиям лиц, может быть доработана для распознавания выражаемых эмоций. Ещё одна из возможных стратегий при построении НС – **нормализация данных**, т.е. приведение входных значений к одинаковой шкале и диапазону, например [0;1].
- **Гиперпараметры** – в отличие от параметров модели, они не изменяются в ходе работы алгоритма машинного обучения и, следовательно, должны быть обоснованно выбраны инженером. К основным гиперпараметрам относят:
    - **Количество скрытых слоёв**. Чем оно меньше, тем быстрее работает НС, однако чем оно больше, тем качественнее НС может решать сложные задачи. При его выборе инженер может либо руководствоваться теоретическими соображениями (например, о структурах головного мозга человека, отвечающих за распознавание визуальной информации), либо начать с небольшого количества слоёв, постепенно увеличивая их, пока качество модели не перестанет увеличиваться.
    - **Функция активации.** Это функция, которая вычисляет выходное значение нейрона на основе линейной комбинации входных параметров. Часто используемые функции это Sigmoid, Tanh и RELU (см. на Рис. 2). Например, из них RELU обладает наименьшей вычислительной сложностью. Поэтому часто она используется во всех внутренних слоях, в то время как выходной слой использует более сложную функцию Sigmoid.



$$S(x) = \frac{1}{1+e^{-x}} \qquad T(x) = \frac{e^{2x}+1}{e^{2x}-1} \qquad R(x) = \begin{cases} 0 & if\ x < 0 \\ x & if\ x \geq 0 \end{cases}$$

Рис. 2. Некоторые функции активации, используемые в НС.

- **Алгоритм оптимизации**, задающий «движение» к оптимуму. Классическим вариантом является Stochastic Gradient Descent (SGD), который минимизирует потери на основе градиента, выбирая на каждой итерации обучающие данные случайным образом. В настоящее время более прогрессивными считаются

алгоритмы RMSProp (отличается от SGD тем, что у каждого параметра адаптируемая скорость обучения) и Adam Optimizer (позволяет устанавливать гиперпараметры НС).

- **Скорость обучения**. Размер «шага» на каждой итерации алгоритма, стремящегося к минимизации функции потерь. Слишком большой шаг ведёт к повышению риска, что алгоритм «промахнётся» мимо оптимума, а слишком маленький – к увеличению времени обучения НС. В настоящее время существуют алгоритмы, способные с определённым успехом варьировать этот шаг в зависимости от «кривизны» функции потерь в текущей точке.
- **Эпохи обучения**. Отражает заданную продолжительность обучения НС. Может задаваться инженером исходя из вычислительной сложности модели, но может и определяться алгоритмом. Например, обучение может автоматически останавливаться в случае, когда ошибка (значение функции потерь) перестаёт уменьшаться в течение какого-либо количества эпох.

- **Архитектуры** – это то, как организованы слои в НС и установлены связи между нейронами. Схема, представленная в Приложении А, даёт некоторое представление о сложности и разнообразии современных архитектур НС. В настоящее время для решения реальных задач наиболее популярны две архитектуры:

Свёрточные НС (CNN) – приобрели популярность для многих задач, связанных с распознаванием изображений (с тех пор, как в 2012 г. на конкурсе ImageNet свёрточная НС значительно превзошла конкурентов). CNN представляет собой чередование свёрточных слоёв (обрабатывает предыдущий слой по фрагментам – например, «окнами» 8*8 клеток) и субдискретезирующих слоёв («уплотняет» фрагменты предыдущего слоя, уменьшая их размер). Такой подход является чрезвычайно прогрессивной альтернативой попиксельному распознаванию изображений. К недостаткам CNN относят значительное количество гиперпараметров сети (количество слоёв, шаг сдвига ядра при обработке слоя, необходимость слоёв субдискретизации, степень уменьшения ими размерности, функция по уменьшению размерности и др.), значения которых в основном приходится подбирать «творчески».

Рекуррентные НС (RNN) – наиболее широко подходят для решения задач, связанных с распознаванием каких-либо последовательностей: голосовых, текстовых, музыкальных и т.п. RNN отличаются наличием памяти: можно сказать, что нейроны запоминают свои предыдущие ответы и при следующем запуске используют их как дополнительные входные данные. Для ограничения роста количества связей применяются алгоритмы, осуществляющие сброс запомненных значений («забывание»). Тем не менее, для обучения RNN требуются очень значительные вычислительные ресурсы (помимо значительного количества обучающих данных).

Также, новинкой в области архитектур НС (предложенной в 2014 г.) являются **генеративно-состязательные сети** (GAN), способные к обучению без учителя. Основным принципом в таком алгоритме является наличие двух НС, одна из которых генерирует образцы, а другая (дискриминативная) старается отличить правильные образцы от неправильных. Генеративная сеть работает на основе смешивания нескольких исходных образцов (имеющихся данных) и стремится повысить процент ошибок второй сети. Дискриминативная сеть обучается различать подлинные и поддельные образцы, а результаты различения подаются на вход генеративной сети, чтобы она могла подправить входной набор латентных параметров. С использованием данной технологии были в частности сгенерированы фотографии людей, которые неотличимы от настоящих.

1.2. Обучение и качество модели

В машинном обучении выделяют **обучение с учителем** (когда известны пары «стимул-реакция»), **обучение без учителя** (нахождение ранее неизвестных закономерностей в данных) и **обучение с подкреплением** (в ходе взаимодействия с некоторой средой). Как было отмечено выше, основное достоинство НС – способность обучаться на доступных примерах данных, т.е. адаптировать свою внутреннюю структуру (прежде всего, значения параметров) для достижения наибольшей точности решения задачи (минимизации наблюдаемой ошибки). Наиболее эффективным алгоритмом обучения для современных НС считается **обратное распространение ошибки**. Фактически, полученный на выходе НС результат сравнивается с верным, а затем алгоритм, двигаясь от выходного слоя по внутренним слоям, изменяет значения весов в связях нейронов, чтобы минимизировать ошибку.

Для проведения обучения, как правило весь доступный набор данных разделяется в некоторых пропорциях (например, 0.7:0.1:0.2) на три поднабора:

- **Обучающие данные** (training set) – используются собственно для настройки параметров модели посредством минимизации ошибки на них.
- **Данные для проверки** (validation set) – периодически используются для контроля процесса обучения, чтобы предотвратить переобучение НС.
- **Тестовые данные** (testing set) – используются для проверки способности обученной модели к генерализации (способности работать с новыми, ранее не «виденными» моделью данными). **Качество модели**, как правило, понимается именно как точность её работы на тестовых данных

Одним из наиболее распространенных показателей итогового качества нейросетевой модели является **среднеквадратичная ошибка (MSE)**, которая отражает усреднённое расстояние между предсказанными моделью значениями и реальными значениями. Значение MSE положительное, и чем ближе к 0 – тем лучше качество модели.

Даже в идеальном случае значение ошибки у НС не должно в точности равняться 0; в противном случае говорят о явлении «**переобучения**», т.е. установке функциональной зависимости от обучающих данных. Такая модель, скорее всего покажет низкую точность на тестовых данных, т.к. они будут отличаться от данных, к которым «приспособилась» модель. Обычно переобучение возникает, когда объем обучающих данных мал, а количество входных параметров и нейронов в НС относительно велико. Кроме увеличения объема обучающих данных и **снижения количества эпох обучения**, для того, чтобы избежать переобучения, используются также следующие стратегии:

- **Регуляризация** – добавляет штраф в модель, если веса слишком велики или их слишком много.
- **Отсев** (dropout) – удаление (обычно с некоторой заданной вероятностью) некоторых узлов из НС, чтобы её структура не переусложнялась.
- **Разделение данных на пакеты** (batch size) – предоставление алгоритму не всех обучающих данных сразу, а разделение их на подмножества.

2. Инструменты и библиотеки

2.1. Colab

Google Colaboratory (Colab) – облачный сервис, в настоящий момент бесплатно предоставляемый компанией Google. Он позволяет запускать Jupiter Notebooks на языке Python 2 или Python 3 и может использоваться для решения задач машинного обучения. Особо интересно то, в рамках Colab пользователям бесплатно предоставляется возможность использования графического процессора (GPU), а также тензорного процессора (TPU), что существенно повышает производительность вычислений. Документы сохраняются на Google Drive и могут использоваться совместно с коллегами.

Jupiter Notebook (расширение .ipynb) – документ в формате JSON, который представляет собой упорядоченный набор ячеек, каждая из которых может содержать программный код, текст (с разметкой), различные элементы мультимедиа и т.п. По умолчанию Jupiter Notebook работает с IPython, однако доступна возможность использования таких языков программирования как R, Julia, Haskell. При работе с ними через Colab, программный код исполняется на сервер в docker контейнере, который выдаётся пользователю на 12 часов (после этого значения переменных и пр. сбрасываются, однако данные могут быть сохранены[1]). Для запуска кода используется иконка «play» слева от ячейки; результат выполнения или системные сообщения выдаются ниже.

Для работы рекомендуется использование браузера Google Chrome, также необходимо наличие аккаунта Google.

2.2. Tensorflow

TensorFlow — открытая (с 2015 г.) программная библиотека для машинного обучения, разработанная компанией Google для построения и тренировки нейронных сетей. Название происходит от слова «тензор», означающего массив произвольной размерности (например, нулевая размерность соответствует скалярному значению, размерность 2 – матрице).

Основной API для работы с библиотекой реализован для Python (есть также реализации для C Sharp, C++, Haskell, Java, Go и Swift). Программы в TensorFlow, по сути, состоят из двух этапов: 1) сбора констант, переменных и операций в граф и 2) их вычисления в рамках одной сессии. Подключение tensorflow в Colab может быть осуществлено при помощи следующего программного кода:

```
!pip install tensorflow==1.14 # для выполнения данной работы рекомендуется использование именно версии 1.14
import tensorflow as tf
```

Ниже мы приводим краткое описание ещё ряда библиотек, которые будут вам нужны для выполнения задания.

2.3. Pandas

Pandas – программная библиотека на языке Python, ориентированная на анализ данных, представленных в структурированном виде (в колонках). Название происходит от словосочетания «панельные данные» (**pan**el **da**ta), которое часто используется в эконометрике. Pandas поддерживает различные операции с данными (groupby, join и др.), а также подготовку и очистку данных (заполнение, замену, обработку пустых значений).

[1] См. https://tzeny.ddns.net/index.php/Google_Colaboratory и https://habr.com/ru/post/348058/

Многие фреймворки машинного обучения, включая TensorFlow[2], работают со структурами данных, поддерживаемых Pandas. В первую очередь это два класса:

- DataFrame, примерно соответствующий таблице в реляционной базе данных – со строками и поименованными колонками,
- Series, соответствующий отдельной колонке – DataFrame содержит одну или несколько Series, а также название для каждой из них.

Используя Pandas, вы можете также импортировать данные в различных форматах – csv, excel и др. Большинство приводимых ниже примеров программного кода подразумевают, что в вашей среде установлена Pandas. В Colab / TensorFlow для этого может использоваться следующий программный код:

```
import pandas as pd
```

2.4. Keras

Keras – открытая библиотека, предназначенная для работы с нейронными сетями, написанная на языке Python. Она может работать с такими средами как TensorFlow, Microsoft Cognitive Toolkit, R, Theano и PlaidML. Основная идея при разработке Keras заключалась в том, чтобы создать независимый от вычислительной платформы дружественный интерфейс, а не ещё один самостоятельный фреймворк машинного обучения. Соответственно, Keras содержит ряд блоков, часто используемых в НС: слои, цели, функции активации, оптимизаторы, инструменты для работы с изображениями и текстом и т.д.

С 2017 г. Keras поддерживается как одна из библиотек в ядре TensorFlow. Для её подключения может использоваться следующий код:

```
import keras as ks
```

2.5. Seaborn

Библиотека Seaborn предназначена для визуализации данных и является надстройкой над популярной библиотекой matplotlib. Seaborn особенно широко используется в Notebook'ах и в сочетании с Pandas, давая возможности для построения эстетично выглядящих и информативных статистических диаграмм.

Для подключения Seaborn в Colab / TensorFlow может быть использован следующий программный код:

```
!pip install -q seaborn # при необходимости
import matplotlib.pyplot as plt
import seaborn as sns
```

Есть возможность выбора в качестве основной одной из шести палитр (цветовых комбинаций), предоставляемых Seaborn[3]:

```
def plot_color_palette(palette: str):
    figure = sns.palplot(sns.color_palette())
    plt.xlabel("Color palette: " + palette)
    plt.show(figure)
```

[2] См. документацию https://colab.research.google.com/notebooks/mlcc/intro_to_pandas.ipynb
[3] См. https://towardsdatascience.com/matplotlib-seaborn-basics-2bd7b66dbee2

```
palettes = ["deep", "muted", "pastel", "bright", "dark", "colorblind"]
for palette in palettes:
    sns.set(palette=palette)
    plot_color_palette(palette)
```

3. Наборы данных (датасеты)

В настоящее время в открытом доступе находится значительное количество данных, которые могут быть использованы для машинного обучения (наборы данных = «датасеты»)[4]. Это связано, в частности, с тем, что ведущие научные журналы при публикации статей требуют и обнародования соответствующих датасетов. Ещё одна важная цель – дать возможность апробации новых алгоритмов в сфере машинного обучения на одинаковых данных, для сравнения достигаемой точности моделей. Крупные компании, впрочем, редко делают общедоступными данные, на которых они строят свои коммерческие продукты, использующие технологии искусственного интеллекта.

В любом случае, формат данных в общедоступных датасетах редко в точности соответствует тому, который требуется в конкретном проекте. Таким образом, первым этапом (после, конечно, нахождения подходящего датасета) является **подготовка данных**. Это может включать в себя:

- очистку данных: например, удаление некорректных значений, таких как отрицательный или нереально большой возраст клиента,
- обработку пропущенных значений: например, их удаление из датасета в Colab может быть сделано следующим образом:

```
my_dataset = my_dataset.dropna() # удалить из dataset строки с пустыми значениями
```

- преобразование данных, представленных в шкале категорий (в большинстве случаев нейросетевые модели работают с числовыми данными): например, следующий программный код в Colab создаёт 3 разные колонки со значениями 0/1 вместо одной `MadeIn`, содержащей страну-производитель:

```
madein = dataset.pop('MadeIn') # удаляем колонку с категориальными данными
dataset['Russia'] = (madein == 1)*1.0
dataset['China'] = (madein == 2)*1.0
dataset['USA'] = (madein == 3)*1.0
dataset.tail()# показать 5 последних строк датасета
```

- разделение имеющихся данных на обучающую и тестовую выборку: например, в пропорции 0,8:0,2:

```
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

- нормализацию данных: например, для реализации этого в Colab простейшим способом – вычитанием мат. ожидания и делением на среднее отклонение – может быть использован нижеследующий код, который реализует и вызывает функцию `norm()`:

```
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

[4] Например, http://deeplearning.net/datasets/ или

Для подготовки данных могут использоваться и другие инструменты, позволяющие наглядное представление и редактирование структурированных данных. Например, MS Excel способен импортировать данные различных форматов и сохранение их в виде файлов формата csv (comma-separated values), которые затем могут быть легко загружены в Colab. Например, следующий программный код позволяет осуществить **загрузку данных** из файла *train.csv*, хранящегося на компьютере пользователя:

```
import pandas as pd
from google.colab import files
file = files.upload()
training_set = pd.read_csv("train.csv", header=None) # предполагается, что файл не содержит заголовков столбцов
```

Часто на этапе подготовки данных производится также их **первичное изучение.** Например, средства классической описательной статистики позволяют вычислить математическое ожидание, дисперсию и прочие характеристики:

```
desc_stats = dataset.describe() # подробнее см. документацию функции[5]
desc_stats = desc_stats.transpose() # форматирование выводимых чисел для более удобного чтения
desc_stats
```

Удобным способом первичного изучения данных является также их **графическое представление** в диаграммах соответствующего формата. Ниже приведён пример построения диаграмм типа KDE в библиотеке seaborn:

```
sns.pairplot(dataset[["Price", "Area", "Age"]], diag_kind="kde")
```

[5] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html

4. Ход работы

Прежде, чем выполнять нижеследующие задания, ознакомьтесь с теоретической справкой, описанием инструментов, библиотек и датасетов.

По результатам выполнения работы **преподавателю должны быть предоставлены** от каждой из бригад:

- программные коды в виде файлов Colab / Jupiter Notebooks,
- отчёт в формате MS Word, оформленный в соответствии с требованиям ГОСТ к научным (учебным) документам.

Все указанные выше файлы должны быть размещены на Google Drive вашего аккаунта в **отдельной папке**. Доступ к ней должен был **выдан преподавателю** (maxis81@gmail.com), с рассылкой уведомления (когда ваша работа будет **полностью завершена**).

4.1. Реализация и обучение НС для задачи классификации[6]

Создайте новый Colab notebook, перейдя по адресу https://colab.research.google.com/ или сделав это из Google Drive.

При помощи меню выберите работу на GPU: Runtime -> Change runtime type -> GPU. Для проверки успешности подключения и работы графического процессора (иногда он оказывается недоступен, в этом случае нужно повторить попытку его включения позже) может быть использован следующий программный код:

```
import tensorflow as tf
tf.test.gpu_device_name() # в случае успешности будет выдан примерно такой результат '/device:GPU:0'
```

Произведите подключение всех необходимых библиотек (Pandas, Keras, Seaborn) при помощи вызова конструкций `import`, как было описано в разделе «Инструменты и библиотеки».

Произведите загрузку обучающего и тестового набора данных из четырёх файлов в формате csv (прилагаются к лабораторной работе, являются обработкой датасета, описывающего диагностику рака груди[7]):

```
from google.colab import files
file = files.upload()
X_train = pd.read_csv("xtrain.csv", header=None)
Y_train = pd.read_csv("ytrain.csv", header=None)
X_test = pd.read_csv("xtest.csv", header=None)
Y_test = pd.read_csv("ytest.csv", header=None)
```

Создайте нейронную сеть из четырёх слоёв, задав для всех, кроме последнего слоя, быстро вычисляемую функцию активации RELU. Для этого целесообразно использовать модель Keras Sequential[8]. Параметр входного слоя `input_dim` определяется тем, что в используемом датасете количество факторов равно 30 (ещё 2 колонки – описательные). При первом выполнении

[6] Данный пример описан в серии видео https://www.youtube.com/watch?v=inN8seMm7UI&list=PLQY2H8rRoyvyK5aEDAI3wUUqC_F0oEroL

[7] Первоисточник и описание: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

[8] См. содержательное описание https://keras.io/getting-started/sequential-model-guide/ и документацию по основным методам (compile, fit, predict) https://keras.io/models/sequential/

работы количество нейронов в каждом из слоёв рекомендуется использовать как в примере ниже:

```
from keras.models import Sequential
from keras.layers import Dense
classifier = Sequential() # Инициализация НС
classifier.add(Dense(units = 16, activation = 'relu', input_dim = 30))
classifier.add(Dense(units = 8, activation = 'relu'))
classifier.add(Dense(units = 6, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))
```

Укажите метод оптимизации[9] и функцию потери[10]:

```
classifier.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy')
```

Теперь мы можем провести обучение нейронной сети. Вместо количества эпох, равного 100, как в нижеследующем примере, **задайте количество, равное (20 + номер вашей бригады*5)**. Прежде, чем запускать нижеследующий код, засеките время, которое потребуется на обучение НС (это значение должно быть отражено в отчёте, как и количество указанных вами эпох). После обучения вы получите значение функции ошибки для последней эпохи (например, loss: 0.0530) – также зафиксируйте его для отчёта.

```
classifier.fit(X_train, Y_train, batch_size = 1, epochs = 100)
```

Получите набор значений, предсказанных вашей нейросетевой моделью (наличие или отсутствие рака груди). В примере ниже для интерпретации прогноза, выдаваемого НС, используется функция, схожая с RELU (нет, если вероятность ниже 0.5, да, если больше или равна):

```
Y_pred = classifier.predict(X_test) # подаём на вход обученной НС тестовый
набор данных
Y_pred = [ 1 if y>=0.5 else 0 for y in Y_pred ]
```

Оцените точность прогноза, даваемого вашей моделью, сравнив предсказанные значения с известными:

```
total = 0
correct = 0
wrong = 0
for i in range(len(Y_pred)):
  total=total+1
  if(Y_test.at[i,0] == Y_pred[i]):
   correct=correct+1
  else:
   wrong=wrong+1

print("Total " + str(total))
print("Correct " + str(correct))
print("Wrong " + str(wrong))
```

Сравните точность прогноза со значением функции ошибки, зафиксированным ранее, и сделайте выводы.

[9] См. описание и возможные параметры https://keras.io/optimizers/

[10] См. описание и возможные параметры https://keras.io/losses/

Создайте и обучите (фиксируя затрачиваемое время) ещё не менее 3-х нейросетевых моделей:

- с иным количеством слоёв, нейронов в них и функциями активации,
- с различным количеством эпох обучения,
- с иным методом оптимизации и/или функцией потерь.

Оцените точность прогнозов, выдаваемых нейросетевыми моделями. Зафиксируйте в отчёте:

- гиперпараметры всех моделей,
- время, затраченное на обучение,
- значение функции ошибки и точность прогнозов.

4.2. Реализация и обучение НС для задачи регрессии

Самостоятельно реализуйте в новом документе НС для решения задачи регрессии, описанную в следующем примере: https://www.tensorflow.org/tutorials/keras/regression (работает версия на русском языке). Вы также можете руководствоваться видео (на английском языке), которое доступно по следующему адресу: https://www.youtube.com/watch?v=-vHQub0NXI4&list=PLQY2H8rRoyvyK5aEDAI3wUUqC_F0oEroL&index=6

Обучите три версии одной и той же НС с использованием 3-х различных типов процессоров (CPU, GPU и TPU), замеряя время, затрачиваемое на обучение. Отметьте время и достигнутую точность модели в отчёте. Сделайте выводы.

Определите и укажите в отчёте оптимальное количество эпох для обучения, позволяющее избежать переобучения (overfitting) нейронной сети.

Ваш отчёт по выполнению данного задания должен содержать, помимо указанного выше, также описательную статистику для используемых данных и диаграммы (в различных цветовых палитрах для различных бригад).

4.3. Реализация и обучение НС для распознавания изображений

Самостоятельно реализуйте и обучите свёрточную нейронную сеть для распознавания изображений из популярного учебного датасета CIFAR-10[11]. Руководствуйтесь следующим примером: https://www.tensorflow.org/tutorials/images/cnn . Отметьте в отчёте затраты времени на обучение и достигнутую точность.

Используйте ту же самую архитектуру НС для распознавания изображений из другого популярного учебного датасета, CIFAR-100[12]. Отметьте в отчёте затраты времени на обучение и достигнутую точность. Сравните с предыдущими результатами и сделайте выводы.

Предложите и реализуйте в архитектуре НС решения, позволяющие повысить точность модели для датасета CIFAR-100. Приведите в отчёте и обоснуйте сделанные изменения.

[11] См. описание по адресу https://www.cs.toronto.edu/~kriz/cifar.html
[12] См. документацию по скачиванию датасетов по адресу https://keras.io/datasets/

5. Рекомендуемые источники

Машинное обучение и архитектура нейронных сетей

https://vas3k.ru/blog/machine_learning/

Tensorflow

https://colab.research.google.com/notebooks/mlcc/tensorflow_programming_concepts.ipynb

https://www.tensorflow.org/tutorials/quickstart/beginner

Colab

https://habr.com/ru/post/428117/

https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d

https://habr.com/ru/post/413229/

Data

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

https://towardsdatascience.com/top-sources-for-machine-learning-datasets-bb6d0dc3378b

https://docs.microsoft.com/ru-ru/azure/machine-learning/team-data-science-process/prepare-data

http://blog.datalytica.ru/2018/04/blog-post.html

Приложение А

Основные архитектуры НС (https://www.asimovinstitute.org/neural-network-zoo/)