|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Робототехника и комплексная автоматизация (РК)**

КАФЕДРА **Системы автоматизированного проектирования (РК6)**

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***«Применение методов машинного обучения (ML) для решения задач технического анализа при управлении активами на фондовом рынке»***

Студент \_\_\_\_**РК6-86Б**\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_А.А. Онюшев\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_Ф.А. Витюков\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_С.В. Грошев\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2024г.*

АННОТАЦИЯ

**Выпускная квалификационная работа на тему:** «Применение методов машинного обучения (ML) для решения задач технического анализа при управлении активами на фондовом рынке».

**Цель работы:** создать и провести исследование различных архитектур нейронных сетей для решения задач технического анализа при управлении активами на фондовом рынке.

**В данной выпускной квалификационной работе представлены:** обзор теории устройства нейронных сетей, основанных на архитектурах MLP, CNN, ViT; анализ входных данных, поступаемых на вход НС; описание различных функций, необходимых для данной корректного обучения НС; Анализ эффективности использования метода ежедневного дообучения; проведение тестов различных НС; описание метрик, используемых для наглядного сравнения различных НС; анализ результатов, полученных по окончанию исследования; подбор оптимальных параметров с пояснением.

**СОДЕРЖАНИЕ**

[ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ 9](#_Toc169707165)

[ВВЕДЕНИЕ 10](#_Toc169707166)

[1. Разработка архитектуры MLP 14](#_Toc169707167)

[1.1. Разработка архитектуры 14](#_Toc169707168)

[1.2. Работа с yahoo finance 16](#_Toc169707169)

[1.3. Кастомный датасет 17](#_Toc169707170)

[1.4. Настройка и параметризация НС 19](#_Toc169707171)

[1.5. Обучение НС 20](#_Toc169707172)

[1.6. Проведение исследования 24](#_Toc169707173)

[1.7. Заключение по MLP 25](#_Toc169707174)

[2. Разработка архитектуры CNN 26](#_Toc169707175)

[2.1. Устройство CNN 26](#_Toc169707176)

[2.2. Изображение «в глазах» компьютера 27](#_Toc169707177)

[2.3. Свертка 29](#_Toc169707178)

[2.4. Слой подвыборки (пулинга) 30](#_Toc169707179)

[2.5. Собственная CNN 31](#_Toc169707180)

[2.6. Проведение исследования 33](#_Toc169707181)

[2.7. Заключение по CNN 41](#_Toc169707182)

[3. Разработка архитектуры ViT 43](#_Toc169707183)

[3.1 Устройство визуальной трансформерной НС 43](#_Toc169707184)

[3.2. Что поступает в трансформер 45](#_Toc169707185)

[3.3. Механизм «внимания» 46](#_Toc169707186)

[3.4. Полносвязная нейронная сеть 49](#_Toc169707187)

[3.5. Блок энкодер 50](#_Toc169707188)

[3.6. Проведение исследования 51](#_Toc169707189)

[3.7. Заключение по ViT 53](#_Toc169707190)

[4. Метод дообучению 55](#_Toc169707191)

[4.1. Разработка модуля дообучения 55](#_Toc169707192)

[4.2. Реализация класса создания датасетов 56](#_Toc169707193)

[4.3. Реализация метода дообучения 58](#_Toc169707194)

[4.4. Проведение исследования 58](#_Toc169707195)

[4.5. Исследования 58](#_Toc169707196)

[4.6. Эффективность метода дообучения 62](#_Toc169707197)

[4.7. Закономерности 63](#_Toc169707198)

[4.8. Оптимальные настройки 64](#_Toc169707199)

[ЗАКЛЮЧЕНИЕ 65](#_Toc169707200)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 66](#_Toc169707201)

[ПРИЛОЖЕНИЕ А. Графическая часть выпускной квалификационной работы 68](#_Toc169707202)

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

**НС** – нейронная сеть.

**ДС** – датасет.

**CNN** – (Convolutional Neural Networks) сверточная НС.

**MLP** – (Multilayer Perceptron) линейный перцептрон.

**ViT** – (Vision Transformer) зрительный трансформер.

**Функция-потерь** – функция, показывающая расстояние от предугаданного НС ответа до истинного ответа.

**Функция-активации** – функция, отвечающая за изменение весов синапсисов.

**Тикер** – краткое название в биржевой информации котируемых инструментов (акций, облигаций, индексов).

**EMA** – (Exponential Moving Average) экспоненциальная скользящая средняя. Один из показателей, помогающих при техническом анализе.

**Stop-Loss** – биржевая заявка трейдера, в которой условием исполнения указано достижение цены, которая хуже, чем текущая рыночная.

**Take-Profit** – биржевая заявка трейдера, которая выставляется заранее, чтобы в случае роста рынка зафиксировать прибыль по бумаге.

**MSE** – (Mean Squared Error) метрика, используемая для оценки эффективности работы регрессионной модели.

**MAE** – (Mean Absolute Error) метрика, используемая для оценки эффективности работы регрессионной модели.

ВВЕДЕНИЕ

В последние десятилетия развитие технологий дало мощный импульс для применения методов машинного обучения в различных сферах экономики и финансов. Одной из ключевых областей, где эти методы находят своё применение, является технический анализ на фондовом рынке. Компании и индивидуальные инвесторы активно ищут способы повысить эффективность управления активами, используя передовые алгоритмы и аналитические инструменты.

Машинное обучение откроет новые горизонты для предсказания рыночных трендов, автоматизации торговых стратегий и минимизации рисков. Введение данных технологий в процесс управления активами может позволить принимать более обоснованные решения, улучшая как краткосрочные, так и долгосрочные инвестиционные результаты.

Целью данной выпускной квалификационной работы является создание и проведение исследования на нейронных сетях с различными архитектурами. Само исследование заключается в получении наглядных графиков и различных метрик, которые будут указывать на эффективность той или иной настройки нейронной сети. Результаты исследований, в свою очередь, будут помогать принимать решения по оптимизации гиперпараметров НС (нейронная сеть). Работа делиться на четыре основные части:

1. Решение задачи технического анализа на архитектуре MLP (линейный перцептрон);
2. Решение задачи технического анализа на архитектуре CNN (сверточная НС);
3. Решение задачи технического анализа на архитектуре ViT (зрительный трансформер);
4. Применение метода дообучения.

*Часть первая. Разработка архитектуры MLP*

Цель данного исследования состоит в изучении различных аспектов линейных нейронных сетей, их структуры, алгоритмов обучения и применения. Мы будем рассматривать перцептрон с различным количеством скрытых слоев, а также с различным количеством нейронов в скрытом слое. Также будут исследованы различные методы оптимизации и регуляризации для улучшения производительности нейронных сетей.

В рамках выполнения этого этапа изучим создание перцептрона, пригодного для анализа графика тикеров на бирже и повторения стратегии торговли брокера. Проведем исследование с различными параметрами. Также данное исследование поможет лучше ознакомиться с математической составляющей НС.

*Часть вторая. Разработка архитектуры CNN*

В настоящее время область сверточных нейронных сетей (CNN) активно исследуется и применяется в различных областях, включая финансовый рынок и биржевую торговлю. CNN позволяют анализировать и обрабатывать большие объемы данных, что идеально подходит для работы с финансовой информацией, которая обычно является многомерной и динамичной.

Использование CNN для анализа биржи и стратегий торговли брокера может быть эффективным способом прогнозирования рыночных движений и принятия обоснованных решений о покупке или продаже активов. Эти модели имеют преимущество в распознавании сложных закономерностей и паттернов на графиках цен, что может помочь в принятии более точных и выгодных решений.

На данном этапе работе рассмотрено, что такое сверточная нейронная сеть и чем она отличается от простого линейного перцептрона. Описана работа по настройке и подбору нужных нам функций-оптимизации, функций-потерь и функций активации, а также размера и количества нейронных слоев и сверточных слоев. Описана работа по составлению датасета на основе данных о тикере MSFT. Также в данной работе были проведены тесты с разными настройками и проанализированы результаты, полученные на выходе. Было проведено сравнение с результатами работы линейного перцептрона при схожих настройках.

*Часть третья. Разработка архитектуры ViT*

Одним из новейших и наиболее перспективных инструментов в области машинного обучения являются зрительные трансформеры (ViT). Изначально разработанные для обработки визуальных данных, ViT демонстрируют высокий потенциал и в финансовом секторе. Их способность эффективно работать с временными рядами и большим объемом рыночных данных открывает новые возможности для предсказания ценовых движений и оптимизации стратегий управления активами. Введение зрительных трансформеров в процесс технического анализа может позволить инвесторам и трейдерам более точно оценивать рыночные тренды, минимизировать риски и принимать обоснованные решения, что существенно повышает общий успех стратегий.

На данном этапе работе рассмотрено, что такое трансформерная нейронная сеть и, в частности, визуальный трансформер. Описана работа по настройке и подбору нужных нам функций-оптимизации, функций-потерь и функций активации, а также различных гиперпараметров. Описана работа по корректировке датасета на основе данных о тикере MSFT. Также в данной работе были проведены тесты с разными настройками и проанализированы результаты, полученные на выходе. Было проведено сравнение с результатами работы сверточной нейронной сети.

*Часть четвертая. Метод дообучения*

В современном мире доступ к знаниям обширен и большинство людей обучаются чему-то новому каждый день. Новые знания помогают находить новые пути решения проблемы, помогают принимать решения основываясь на похожих ситуациях в прошлом и просто делают человека многогранным. Нейронные сети берут свое начало в биологии и анатомии человека и имеют внутри себя многие схожие системы и архитектуры. Работы, написанные мною до этого, использовали только простой (конечный) метод прямого обучения нейронной сети, в данной же работе, основываясь на примере реального человека, который получает новые знания каждый день, будет проведено исследование с использованием так называемого цикла дообучений НС. Каждый новый день у брокера появляется новая сущность ДС (датасет) с информацией о выставленных им take-profit и stop-loss. Этой новой информации и будем дообучать НС.

Целью данного этапа работы является разработка и проведение исследования модуля дообучения для CNN. Также будут найдены оптимальные настройки для метода.

1. Разработка архитектуры MLP

Нейронные сети являются мощным инструментом в области искусственного интеллекта и машинного обучения. Они моделируют работу человеческого мозга, позволяя компьютерам обрабатывать и анализировать сложные данные. Исследования по нейронным сетям имеют огромный потенциал для решения задач по техническому анализу биржи.

Архитектура перцептронов схематично изображена на рисунке 1.

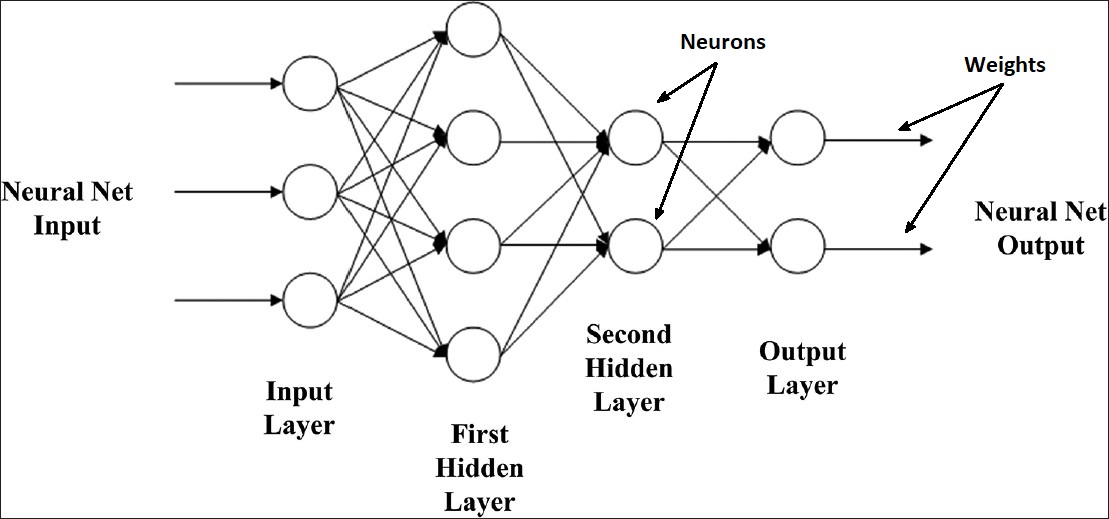


Рисунок 1 – Нейронно-синапсисная архитектура НС

* 1. Разработка архитектуры

При разработке архитектуры нейронной сети требуется учитывать, что на вход должны передаваться данные некоторого размера, которые и будут проанализированы НС. В данной работе на вход НС передается информация о тикерах в формате: Массив[N] со значениями High, массив[N] со значениями Low, массив[N] со значениями EMA 200 (Exponential Moving Average), где N – количество исследуемых дней от 40 до 2000, а также передается asset – число отвечающее за наличие данного тикера в портфеле.

Проанализировав данные, поданные на вход НС, она возвращает некий результат своего анализа – выходные данные. При разработке архитектуры важно учесть правильность выходных данных. В данной работе выходными данными должны быть два числа в диапазоне от -0.5 до 1.5, означающие биржевые заявки трейдеру – take-profit и stop-loss.

Основное обучение НС происходит в скрытых слоях. Таких слоев может быть огромное количество и каждый из них может содержать абсолютно разное количество нейронов. В данной работе было выбрано использовать 2 скрытых слоя по 100 нейронов в каждом.

Полученная архитектура НС показана на рисунке 2.

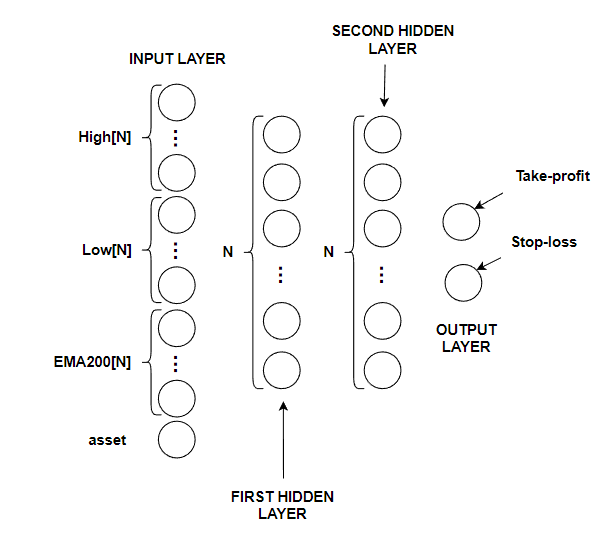


Рисунок 2 – Архитектура исследуемой НС

Реализация данной архитектуры в коде представлена в листинг 1, где в классе NN\_nasdaq в переменных linear1, linear2, linear3 реализованы слои синапсисов между входным и первым скрытым, первым скрытым и вторым скрытым и вторым скрытым и выходным слоями, соответственно.

Листинг 1 – Класс нейронной сети.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | *class* NN\_Nasdaq(nn.Module):  *def \_\_init\_\_*(*self*, input\_size, hidden\_size, output\_size):  *super*().\_\_init\_\_()  *self*.linear1 = nn.Linear(input\_size, hidden\_size) *# входной слой  self*.linear2 = nn.Linear(hidden\_size, hidden\_size) *self*.linear3 = nn.Linear(hidden\_size, output\_size) *# скрытый слой  self*.act\_func = nn.ReLU() *# Задали функцию активации*  *self*.flat = nn.Flatten()  '''self.model = nn.Sequential(  linear1,  act\_func,  linear2  )'''   *def* forward(*self*, x):  *#print(x)* out = *self*.flat(x) out = *self*.linear1(out)  out = *self*.act\_func(out)  out = *self*.linear2(out)  out = *self*.act\_func(out)  out = *self*.linear3(out)  out = out.reshape((out.shape[0]))  *return* out |

* 1. Работа с yahoo finance

Для получения необходимой информации о дневных свечах тикера, необходимо обратится к API yahoo finance, но так как они прекратили его поддержку в 2016 году, воспользуемся модулем yfinance.

Чтобы выбрать конкретный тикер, создадим переменную ticker (смотреть листинг 2). Чтобы выбрать количество дней информацию по которым мы хотим получить, создадим переменную days (смотреть листинг 2). Чтобы выбрать количество дней для ЕМА введем переменную EMA\_N (смотреть листинг 2).

Листинг 2 – Функция загрузки информации с yahoo finance

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Days = 200 # Кол-во дней в датасете  EMA\_N = 200 # Кол-во дней в ЕМА  num\_candle = Days+EMA\_N  ticker = "MSFT" # Названия тикера который отслеживаем  hist = fin.download(ticker, period=f'{num\_candle}d', interval='1d') |

Функция fin.download() (смотреть листинг 2) загрузить всю необходимую нам информацию о свечах, а код указанный в листинг 3 создаст столбец с информацией о EMA [1].

Листинг 3 – Создание столбца с ЕМА

|  |  |
| --- | --- |
| 1 | hist[f'EMA{EMA\_N}'] = hist['Close'].ewm(span=EMA\_N, ajust=False).mean() # Считаем нужный ЕМА |

Итоговая таблица будет иметь вид, указанный на рисунке 3.

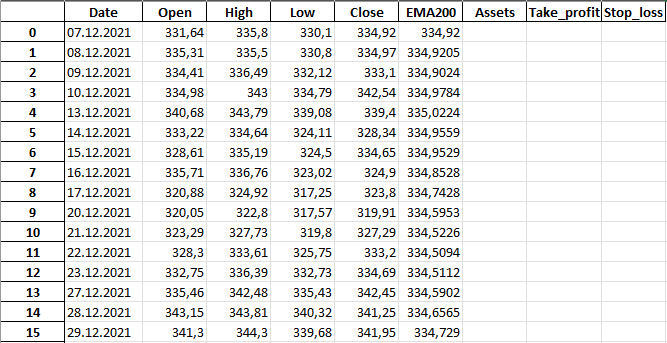


Рисунок 3 – Итоговый вид таблицы датасета

* 1. Кастомный датасет

Требуемый для исследования датасет, находится вне модуля PyTorch, следовательно, для его корректной загрузки в НС, а также преобразования в тензер, необходимо написать собственный модуль подготавливающий наш датасет под класс Dataset описанный в модуле.

В классе PreDataLoader (смотреть листинг 4) как раз происходит составление предварительного датасета, который отнормирован и сформирован под требуемые параметры. Основная работа происходит в методе create\_exit\_data() данного класса (смотреть листинг 4). Метод get\_data() (смотреть листинг 5) просто возвращает все значения полученные в методе create\_exit\_data() в формате списка списков списков(необходимая информация о тикере) и значения (требуемый take-profit/stop-loss).

Листинг 4 – Класс предварительного форматирования входных данных

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | class PreDataLoader(Dataset):  """  Class for correct load all data  """  def \_\_init\_\_(self, data, pred\_size=4, label\_size=2, candle\_count=50, start=200, stop=350, normalization\_pred=True, normalization\_label=False, label\_offset=0):  """Create DataLoader for your DataSet  :Parameters:  tickers : str, list  List of tickers to download  data: DataFrame  DataFrame of candle parameters  pred\_size: int  Prediction size(Count of columns with prediction parameters)  label\_size: int  Label size(Count of columns with label parameters)  butch\_size: int  Butch size for NN  start: int  Number of row that you want to start  stop: int  Number of row that you want to finish  """  self.data = data.copy()  self.data2 = data.copy()  self.label\_offset = label\_offset  self.pred\_size = pred\_size  self.label\_size = label\_size  self.candle\_count = candle\_count  self.start = start  self.stop = stop  self.predictions = []  self.labels = []  self.butches = []  self.normalization\_pred = normalization\_pred  self.create\_exit\_data()  def create\_exit\_data(self):  prediction = []  label = [] if self.label\_size>1 else 0  for i in range(self.start, self.stop-self.candle\_count):  for j in range(self.pred\_size-1):  for k in range(self.candle\_count):  prediction.append(self.data.iloc[:, j][i+k])  prediction.append(self.data.iloc[:, self.pred\_size-1][i+self.candle\_count-1])  for j in range(self.label\_size):  if (self.label\_size == 1):  label = self.data.iloc[:, j+self.pred\_size+self.label\_offset][i+self.candle\_count-1]  else:  label.append(self.data.iloc[:, j+self.pred\_size+self.label\_offset][i+self.candle\_count-1])  if(self.normalization\_pred == True):  max0 = max(prediction[0:2\*self.candle\_count])  min0 = min(prediction[0:2\*self.candle\_count])  for i in range(len(prediction)-1):  prediction[i] = (prediction[i]-min0)/(max0-min0)  self.predictions.append(prediction)  self.labels.append(label)  label = []  prediction = [] |

Листинг 5 – Метод возвращающий датасет в требуемом формате

|  |  |
| --- | --- |
| 1  2  3 | def get\_data(self):  exit\_data = [[self.predictions[i], self.labels[i]] for i in range(len(self.labels))]  return exit\_data |

Класс NASDAQDataSet (смотреть листинг 6) требуется для использования исследуемого датасета с корректными для НС типами данных. Поскольку НС использует тензоры и модуль PyTorch [2][3] предоставляет возможность переноса данных на VDDR память GPU, требуется использовать правильные типы данных. Этого можно достичь унаследовав класс Dataset, который расположен в модуле PyTorch.

Листинг 6 – Класс подготавливающий датасет в требуемом типе данных

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class NASDAQDataSet(Dataset):  def \_\_init\_\_(self, data=None):  if (data==None):  raise Exception("Plz, entry all data")  else:  self.data=data  def \_\_len\_\_(self):  return len(self.data)  def \_\_getitem\_\_(self, index):  info = torch.tensor(self.data[index][0])  label = torch.tensor(self.data[index][1])  info = info.to(torch.float32)  label = label.to(torch.float32)  return (info, label) |

* 1. Настройка и параметризация НС

Помимо настройки количества скрытых слоев и количества нейронов в этих слоях, НС требует настройку функции-оптимизации и функции-потерь.

Исследуемая задача более всего похожа на задачу регрессии, основываясь на этом, максимально логичным решением будет использовать функцию MSELoss(Mean Squared Error Loss), как функцию-потерь (смотреть листинг 7).

Листинг 7 – Добавление функции-потерь MSELoss()

|  |  |
| --- | --- |
| 1 | loss\_func = nn.MSELoss() |

Формула MSE:

,

Где MSE – mean squared error (среднеквадратичная ошибка);

n – количество входных значений;

– точная величина;

– предугаданная величина.

После выбора подходящей функции-потерь, необходимо подобрать функцию-оптимизатор. На данный момент одной из самых популярных и эффективных функций-оптимизаторов является Adam(adaptive moment), он отлично подойдет и для этой задачи (смотреть листинг 8). Lr (шаг поиска) достаточно будет указать равным . Отличный пример сравнения разных функций-оптимизаций приведен на данном сайте: <https://emiliendupont.github.io/2018/01/24/optimization-visualization/>

Листинг 8 – Добавление функции-оптимизатора Adam()

|  |  |
| --- | --- |
| 1 | optimizer = torch.optim.Adam(model.parameters(), lr=1e-3) |

* 1. Обучение НС

Чтобы НС хорошо работала требуется обучить её на данных из всего датасета. Прогоняя эти данные снова и снова мы повышаем вероятность более правильного ответа НС. Прогон всего датасета через НС называется – эпоха. Совсем не обязательно в одной эпохе обучать НС всем датасетом за раз, можно разбить его на меньшие части. Такие части называют – бач данных.

Для исследования был составлен пробный датасет, который содержал 91 сущностей для обучения и 22 сущности для тестов. Размер бача равен 50 сущностей. Количество эпох равно 5000. Используемый код для обучения НС (смотреть листинг 9).

Листинг 9 – Обучение НС

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | for epoch in range(epochs):  for info, label in dataloader:  optimizer.zero\_grad() # Обнуляем градиенты, чтобы они не помешали нам на прогоне новой картинки  info = info.to(device)  label = label.to(device)  pred = model(info)  loss = loss\_func(pred, label) # посчитали ошибку (значение в label - значение полученое нашим model(img))  loss.backward() # Прошелись по всему графу вычислений и посчитали все градиенты для нейронов  loss\_item = loss.item()  optimizer.step() # Сделали шаг градиентным спуском с учетом градиента посчитанного loss.backward()  print(f'Full time learning : {time.time() - start\_time}')  path\_name = 'model\_take\_profit\_'+device+'.pth' if label\_offset == 0 else 'model\_stop\_loss\_'+device+'.pth'  torch.save(model.state\_dict(), path\_name) |

Время обучения указано на рисунке 4.





Рисунок 4 – Время обучения НС на CPU и CUDA соответственно

Ошибка при этом имеет вид, указанный на рисунке 5.

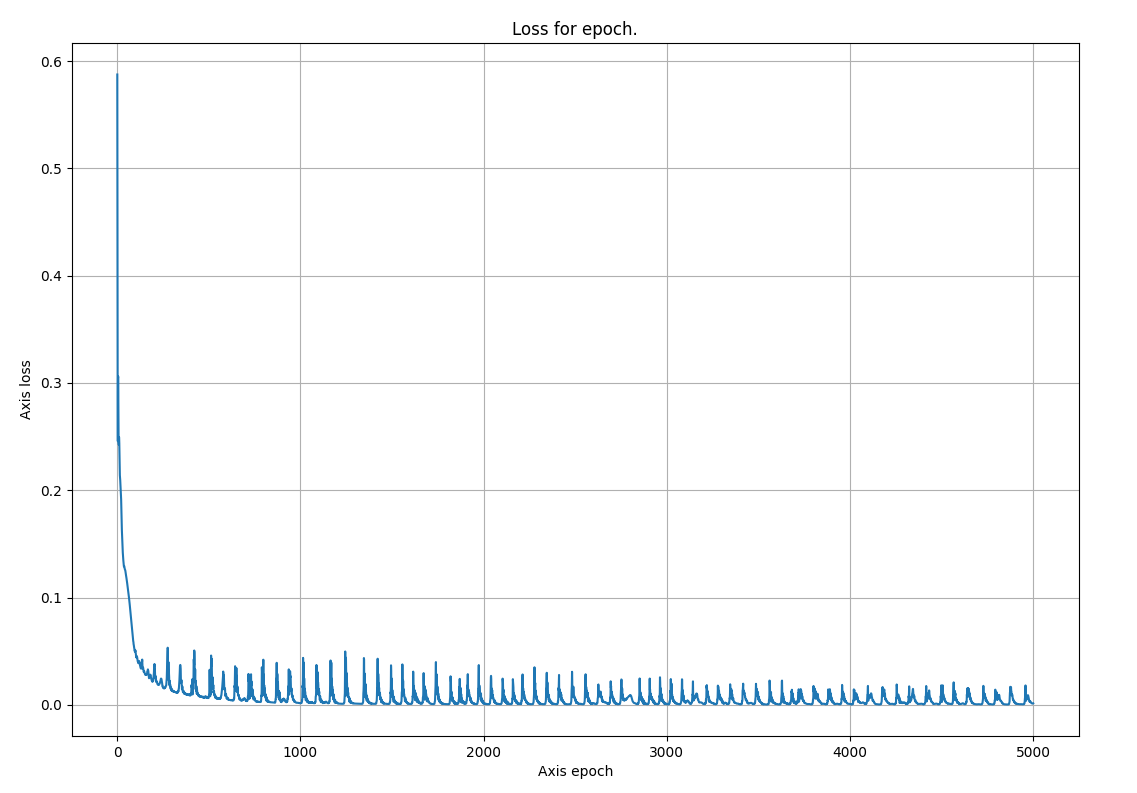


Рисунок 5 – Ошибка при указанных настройках НС

Точность данной НС выглядит, как показано на рисунке 6.

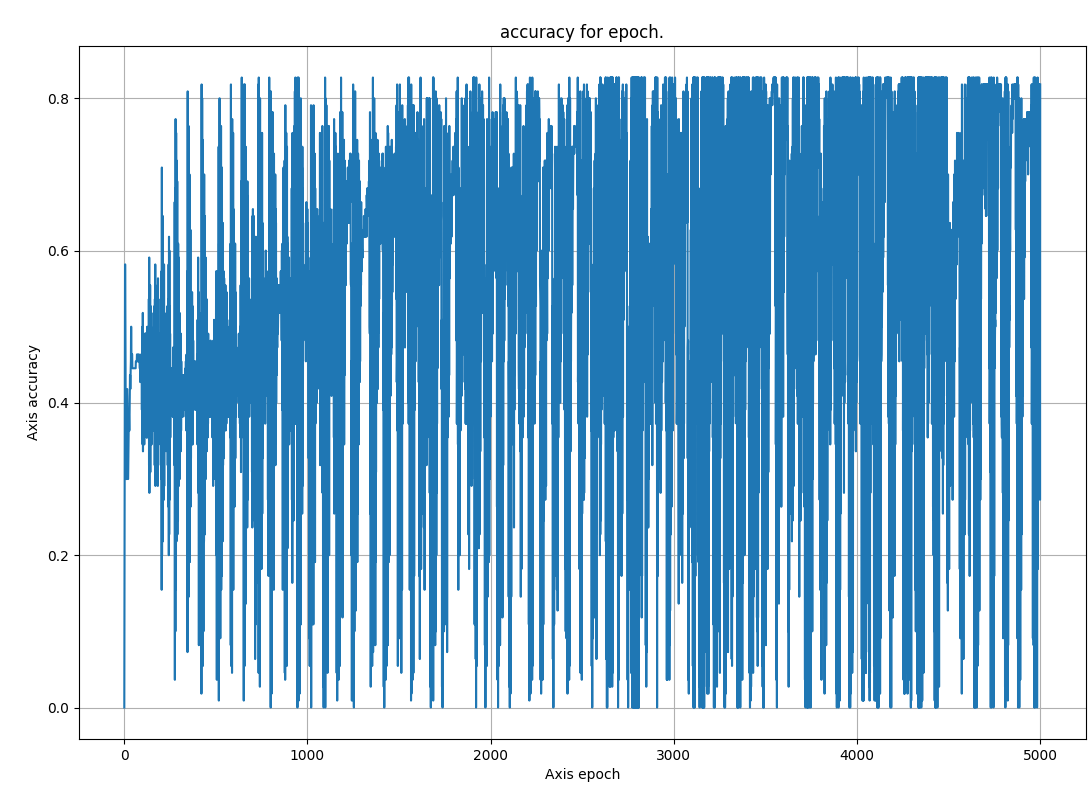


Рисунок 6 – Точность при указанных настройках НС

Ошибка в данной задаче считается, как:

где – значение, полученное на выходе НС;

– требуемое значение.

Реализация данной функции в коде представлена на листинге 10.

Листинг 10 – Функция подсчета ошибки

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | def lossing(pred,label):  count = 0  predict = pred.detach().numpy()  lbl = label.detach().numpy()  for i in range(len(label)):  loss\_now = np.abs(lbl[i]-predict[i])  count += loss\_now  return count |

Точность для задач регрессии сложно посчитать просто сложив все ответы НС, которые идентичны требуемым ответам, поэтому в данной работе точность считается как:

где – значение, полученное на выходе НС;

– требуемое значение;

– какое-то число, показывающее насколько мала ошибка.

Реализация данной функции в коде представлена на листинге 11.

Листинг 11 – Функция подсчета точности

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | def accuracy(pred, label, epsilon=1):  count = 0  predict = pred.detach().numpy()  lbl = label.detach().numpy()  for i in range(len(label)):  accuracy\_now = (lbl[i]-predict[i])\*100  if (accuracy\_now<epsilon):  count+=1  return count |

Также были учтены следующие метрики:

1. Среднее значение величин, которые мы требуем от НС, и которые получаются на выходе из неё
2. Среднеквадратичное отклонение полученного результата от требуемого
3. Среднее значение ошибки
4. Максимальная ошибка

Данные метрики проиллюстрированы на рисунке 7 и 8.

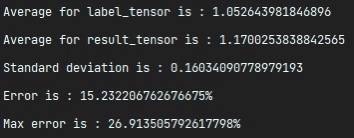


Рисунок 7 – Полученные метрики на этапе тестирования

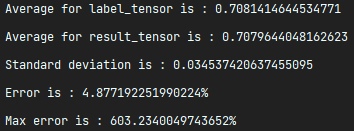


Рисунок 8 – Полученные метрики на этапе обучения НС

* 1. Проведение исследования

В качестве функций-потерь будем тестировать MSELoss и MAELoss. Также рассмотрим разное количество эпох (50, 100, 200, 400, 800, 1600, 3200, 6400, 10000). Все предложенные параметры нужны нам, чтобы четко понять как архитектура НС, функция-потерь и количество эпох влияют на итоговый результат.

График красного цвета – требуемый (истинный) результат. График синего цвета – результат, предугаданный НС. По оси Х – номера сущностей из тестового датасета. По оси У – «stop-loss».

На рисунке 9 и 10 представлены лучшие графики предсказания.



Рисунок 9 – График результата исследования на MLP



Рисунок 10 – График результата исследования на MLP

* 1. Заключение по MLP

В ходе выполнения данного этапа прошло ознакомление с написанием искусственного интеллекта на основе НС. Изучены возможности модуля PyTorch языка программирования Python. Разобраны главные математические и биологические основы НС.

Создан линейный перцептрон с возможностью обучения на различных датасетах. Изучены различия в поведении разных функций-оптимизаторов и функций-потерь. Изучены перемены, которые появляются при изменении параметров нейронной сети. Разобрана теоретическая составляющая для сверточных нейронных сетей.

При обучении НС на датасете, составленном по показателям дневных свечей на бирже, можно понять, что линейный персептрон плохо находит закономерность. Нужен либо датасет большего размера, либо другая структура и архитектура НС, но для данной архитектуры подобранные параметры являются оптимальными.

1. Разработка архитектуры CNN

Сверточные нейронные сети (Convolutional Neural Networks, CNN) стали одним из самых мощных и широко используемых инструментов в области машинного обучения и компьютерного зрения. Изначально разработанные для анализа двумерных данных, таких как изображения, они оказали значительное влияние на технологии распознавания образов, классификации изображений и обработки естественного языка. Принцип работы CNN основан на применении сверток и пуллинга для выделения иерархических признаков из входных данных, что позволяет сети эффективно распознавать сложные паттерны и структуры.

Благодаря своей архитектуре и способности к автоматическому извлечению признаков CNN находят применение в медицине, автономных транспортных средствах, системах безопасности и многих других областях. Их способность обучаться на больших объемах данных и адаптироваться к различным задачам делает CNN неотъемлемой частью современного искусственного интеллекта.

* 1. Устройство CNN

Сверточная нейронная сеть (ConvNet/CNN) [4] — это специальная архитектура искусственных нейронных сетей, которая может принимать входное изображение (матрицу данных), присваивать важность (изучаемые веса и смещения) аспектам или объектам изображения (матрицы) и отличать одно от другого. При этом изображения (матрицы), в сравнении с другими архитектурами НС, требуют гораздо меньше предварительной обработки. В примитивных методах фильтры разрабатываются вручную, но достаточно обученные сети CNN учатся применять эти фильтры/характеристики самостоятельно.

Архитектура CNN аналогична структуре связей нейронов в мозгу человека, учёные черпали вдохновение в организации зрительной коры головного мозга. Отдельные нейроны реагируют на стимулы только в некоторой области поля зрения, также известного как перцептивное поле. Множество перцептивных полей перекрывается, полностью покрывая поле зрения CNN.

Архитектура сверточной нейронной сети схематично изображена на рисунке 11 [5][6]:

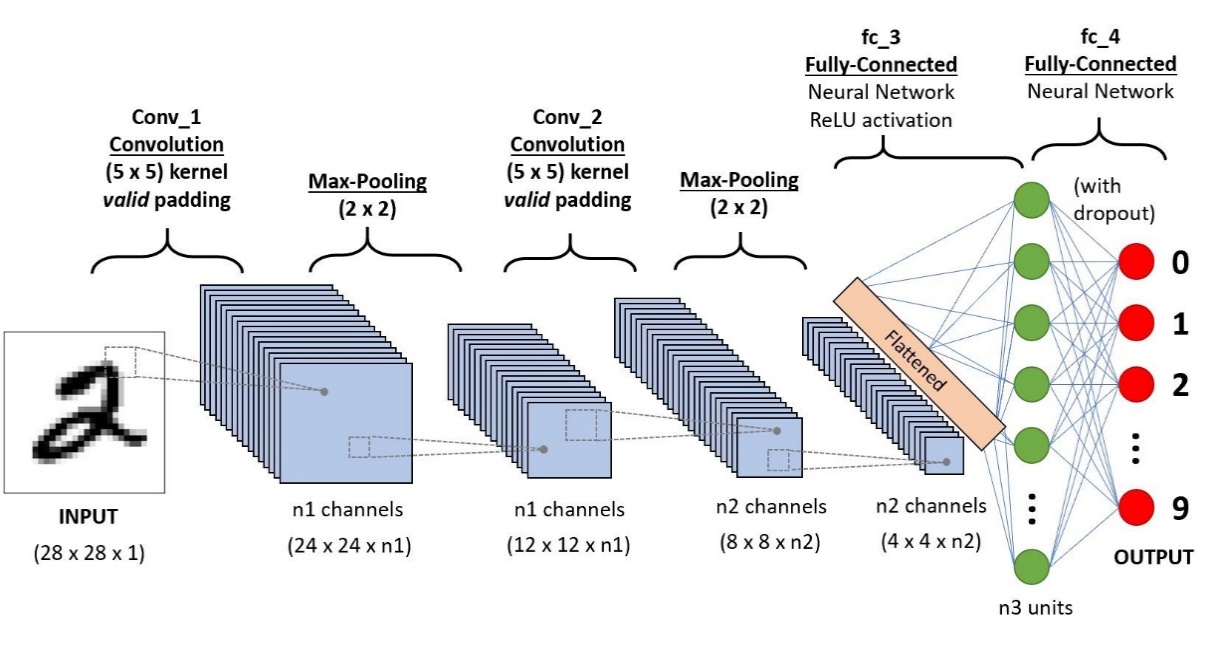


Рисунок 11 – Архитектура сверточной нейронной сети

Главной особенностью свёрточных сетей является то, что они обычно работают именно с матрицами данных, а потому можно выделить особенности, свойственные именно им. Многослойные персептроны работают с векторами, а потому для них нет никакой разницы, находятся ли какие-то точки рядом или на противоположных концах, так как все точки равнозначны и считаются совершенно одинаковым образом. Изображения или же матрицы могут обладают локальной связностью. Например, если речь идёт об изображениях человеческих лиц, то вполне логично ожидать, что точки основных частей лица будут рядом, а не разрозненно располагаться на изображении. Поэтому требовалось найти более эффективные алгоритмы для работы с матрицами данных и ими оказались свёрточные сети.

* 1. Изображение «в глазах» компьютера

Изучая CNN, невольно появляется вопрос «Что такое свертка?». Перед тем, как ответить на него, нужно разобраться, как компьютер «видит» какое-то изображение.

На рисунке 12 можно увидеть, как компьютер представляет изображение.



Рисунок 12 – Как компьютер видит изображение

Изображения в компьютере представляются в виде пикселей, а каждый пиксель – это значения интенсивности соответствующих цветовых каналов. При этом интенсивность каждого из каналов описывается целым числом от 0 до 255. Чаще всего используются цветные изображения, которые состоят из RGB пикселей – пикселей, содержащих яркости по трём каналам: красному, зелёному и синему. Различные комбинации этих цветов позволяют создать любой из цветов всего спектра. Чтобы хранить информацию о всех пикселях удобнее всего использовать тензор ‒ 3D массив чисел, или, проще говоря, массив матриц чисел.

На рисунке 13 представлено преобразование из изображения в тензор.



Рисунок 13 – Преобразование из изображения в тензор

На рисунке 14 представлен пример хранения какого-то изображения компьютером.



Рисунок 14 – Изображение цифры и его представление в компьютере.

* 1. Свертка

Слой свёртки, как можно догадаться по названию типа нейронной сети, является самым главным слоем сети. Его основное назначение – выделить признаки на входном изображении и сформировать карту признаков. Карта признаков – это всего лишь очередной тензор (массив матриц), в котором каждая матрица отвечает за какой-нибудь выделенный признак.

Для того, чтобы слой мог выделять признаки, в нём имеются так называемые фильтры (или ядра). Фильтр – квадратная матрица небольшого размера (обычно 3х3 или 5х5), заполненная определенным образом. Чем больше будет таких фильтров – тем больше признаков удастся выделить и тем больше будет глубина каналов у выходного тензора слоя

Данный фильтр поочередно накладывается на изображение начиная с левого верхнего угла и пока не дойдет до правого нижнего угла. На каждом шаге числа в матрице фильтра перемножаются с соответствующими числами на изображении, полученные результаты складываются и записываются в матрицу признака. Лучше понять это можно, посмотрев на рисунок 15.

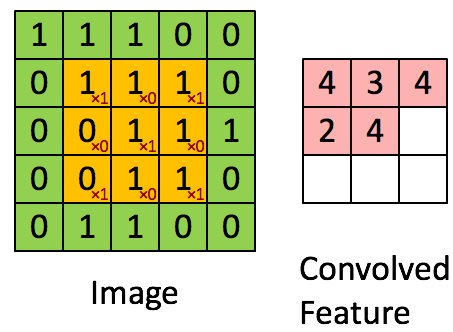


Рисунок 15 – Свертка изображения 5х5х1 с фильтром 3х3х1 для получения признака 3х3х1

На нем желтым цветом изображен фильтр размером 3х3х1, а красные цифры в правом углу –числа матрицы фильтра. Зеленым цветом изображена матрица изображения 3х3х1, а красным полученный признак. Таким образом на 5-ом шаге итерации мы получим:

Полученные в итоге матрицы признаков могут иметь такой же размер, что и исходное изображение, либо размер меньше (как в нашем случае). Это зависит от заданных размера шага и начального заполнения. В [данном](https://github.com/vdumoulin/conv_arithmetic/tree/master/gif) репозитории (См. https://github.com/vdumoulin/conv\_arithmetic/tree/master/gif) можно найти множество разных GIF-файлов, которые помогут лучше понять, как заполнение и длина шага работают [7].

* 1. Слой подвыборки (пулинга)

Так как сверточные НС используют так же и простую сеть прямого распространения, нам необходимо уменьшить кол-во выходных параметров из слоя свертки и при этом не потерять важную информацию. Для этого используют слои подвыборки.

Данный слой позволяет уменьшить пространство признаков, сохраняя наиболее важную информацию. Существует несколько разных версий слоя пулинга, среди которых максимальный пулинг, средний пулинг и пулинг суммы. Наиболее часто используется именно слой макспулинга.

Примеры различных версий пулинга показаны на рисунке 16:

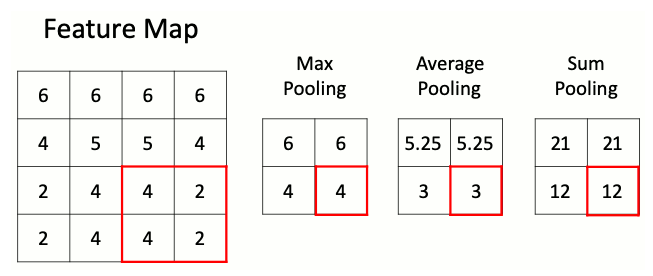


Рисунок 16 – Преобразования слоя подвыборки

Действия этого слоя идентичны действиям слоя свертки, только используются другие операции (для макспулинга – берется максимальное число, для среднего пулинга – берется среднее арифметическое от чисел).

* 1. Собственная CNN

Изучив, как устроена сверточная нейронная сеть, можно приступать к написанию собственной.

При разработке архитектуры нейронной сети требуется учитывать, что на вход должны передаваться данные некоторого размера, которые и будут проанализированы НС. В данной работе на вход НС передается информация о тикерах в формате: Массив[N] со значениями High, массив[N] со значениями Low, массив[N] со значениями EMA 200, где N – количество исследуемых дней от 40 до 2000, а также передается массив[N] со значениями asset – число отвечающее за наличие данного тикера в портфеле. Все эти 4 массива по N значений образуют матрицу 4×N. Её мы и будем использовать, как входные данные.

Проанализировав данные, поданные на вход НС, она возвращает некий результат своего анализа – выходные данные. При разработке архитектуры важно учесть правильность выходных данных. В данной работе выходными данными должно быть число в диапазоне от -0.5 до 1.5, означающие биржевые заявки трейдеру – take-profit или stop-loss.

Преобразуем нашу матрицу 4×N в удобный для обычных линейных нейронов формат, используя три слоя свертки с ядрами 2×2 и итоговым выходным количеством сверток 1×64. Далее с помощью скрытых слоев обычных линейных нейронов проанализируем получившиеся свертки. Было использовано 4 скрытых линейных слоя для лучшего анализа.

Полученную архитектуру сети можно увидеть на рисунке 17:

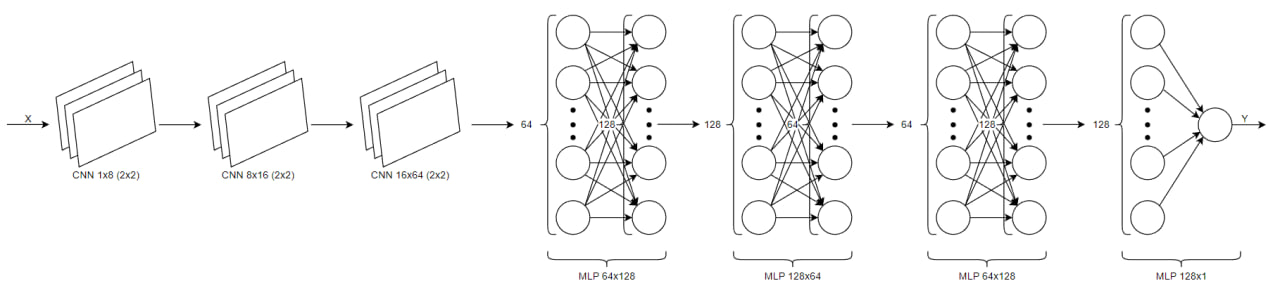


Рисунок 17 – Архитектура НС

Реализация данной архитектуры в коде показана на 12 листинге:

Листинг 12 – Класс нейронной сети.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | *class ConvNet(nn.Module):*  *def \_\_init\_\_(self):*  *super().\_\_init\_\_()*  *self.act = nn.LeakyReLU(0.18) #функция активации*  *self.conv0 = nn.Conv2d(1, 8, 2, stride=1, padding=0)*  *# сверточный слой со сверточным ядром 2 на 2*  *self.conv1 = nn.Conv2d(8, 16, 2, stride=1, padding=0)*  *self.conv2 = nn.Conv2d(16, 64, 2, stride=1, padding=0)*  *self.adaptivepool = nn.AdaptiveAvgPool2d((1, 1))*  *self.flatten = nn.Flatten()*  *self.linear1 = nn.Linear(64, 128)*  *# линейный слой 64 на 128*  *self.extra\_linear\_1 = nn.Linear(128, 64)*  *# линейный слой 128 на 64*  *self.extra\_linear\_2 = nn.Linear(64, 128)*  *# линейный слой 64 на 128*  *self.linear2 = nn.Linear(128, 1)*  *# линейный слой 128 на 1*  *def forward(self, x):*  *# функция описывающая одну итерацию обучения НС*  *out = self.conv0(x)*  *out = self.act(out)*  *out = self.conv1(out)*  *out = self.act(out)*  *out = self.conv2(out)*  *out = self.act(out)*  *out = self.adaptivepool(out)*  *out = self.flatten(out)*  *out = self.linear1(out)*  *out = self.act(out)*  *out = self.extra\_linear\_1(out)*  *out = self.act(out)*  *out = self.extra\_linear\_2(out)*  *out = self.act(out)*  *out = self.linear2(out)*  *return out* |

* 1. Проведение исследования

Чтобы НС хорошо работала требуется обучить её на данных из всего датасета. Прогоняя эти данные повторно множество раз, мы повышаем вероятность более правильного ответа НС. Прогон всего датасета через НС называется – эпоха. Совсем не обязательно в одной эпохе обучать НС всем датасетом за раз, можно разбить его на меньшие части. Такие части называют – бач данных.

Для исследования был составлен пробный датасет, который содержал 200 сущностей для обучения и 50 сущности для тестов. Размер бача равен 25 сущностей. Используемый код для обучения НС на Python:

Сама реализация полного цикла обучения представлена на листинге 13.

Листинг 13 – Код обучающего цикла НС.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | *for epoch in range(epochs):*  *loss\_val = 0*  *acc\_val = 0*  *for sample in train\_loader:*  *info, lbl = sample['info'], sample['label']*  *# переносим требуемые данные на то устройство, где будем*  *обучать НС (CPU или GPU)*  *info = info.to(device)*  *lbl = lbl.to(device)*  *optimizer.zero\_grad()*  *with autocast(use\_amp):*  *pred = CNNet(info)*  *loss = loss\_fn(pred, lbl)*  *# Считаем функцию-потерь*  *scaler.scale(loss).backward()*  *loss\_item = loss.item()*  *loss\_val += loss\_item*    *# Обновляем веса*  *scaler.step(optimizer)*  *scaler.update()*  *acc\_current = accuracy\_v3(pred.cpu().float(), lbl.cpu().float(), epsilon=epsilon)*  *acc\_val += acc\_current*  *print(f"Epoch : {epoch+1}")*  *print(f"Loss : {loss\_val / len(train\_loader)}")*  *print(f"Acc : {acc\_val / (len(train\_loader)\*batch\_size)}")*  *print(f'Full time learning : {time.time() - start\_time}')* |

Исследуемая задача более всего похожа на задачу регрессии, основываясь на этом, максимально логичным решением будет использовать функцию MSELoss (Mean Squared Error Loss) или MAELoss (Mean Absolute Error Loss), как функцию-потерь.

Формула MSE:

,

где MSE – mean squared error (среднеквадратичная ошибка);

n – количество входных значений;

– точная величина;

– предугаданная величина.

Формула MAE выглядит так:

,

где MAE – mean absolute error (средняя абсолютная ошибка);

n – количество входных значений;

– точная величина;

– предугаданная величина.

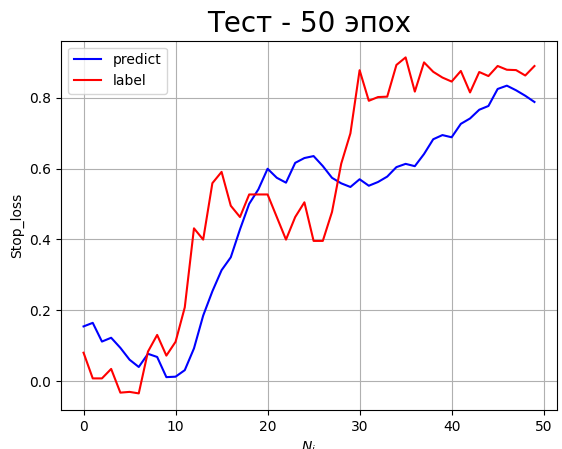
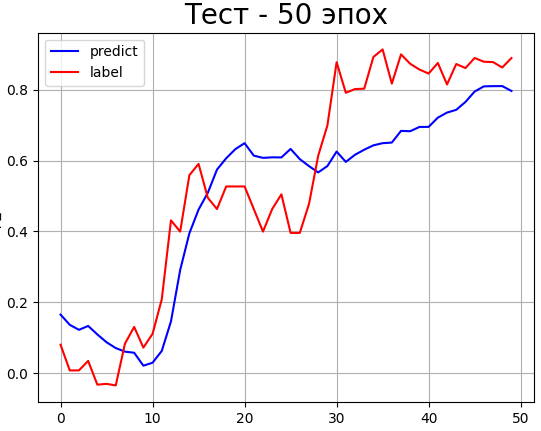
После выбора подходящей функции-потерь, необходимо подобрать функцию-оптимизатор. На данный момент одной из самых популярных и эффективных функций-оптимизаторов является Adam (adaptive moment), он отлично подойдет и для этой задачи. Lr (шаг поиска) достаточно будет указать равным . Отличный пример сравнения разных функций-оптимизаций приведен на данном сайте: <https://emiliendupont.github.io/2018/01/24/optimization-visualization/>

Подобрав все возможные необходимые настройки, можно приступать к тестированию.

Тестирование будем проводить и на CNN архитектуре, и на MLP. В качестве функций-потерь будем тестировать MSELoss и MAELoss. Также рассмотрим разное количество эпох (50, 100, 200, 400, 800, 1600, 3200, 6400, 10000). Все предложенные параметры нужны нам, чтобы четко понять как архитектура НС, функция-потерь и количество эпох влияют на итоговый результат.

Полученные результаты представлены ниже. График красного цвета – требуемый (истинный) результат. График синего цвета – результат, предугаданный НС. По оси Х – номера сущностей из тестового датасета. По оси У – “stop loss”.

На рисунке 18 представлены результаты 50 эпох.

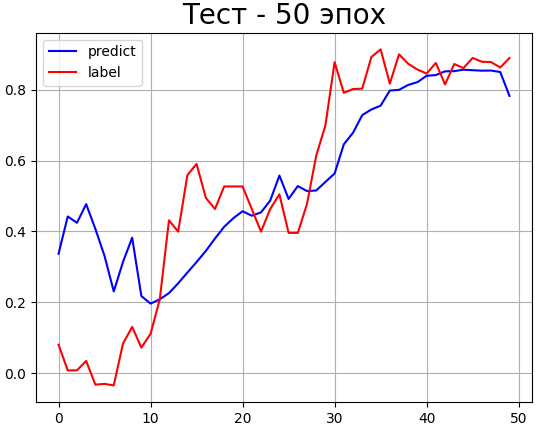
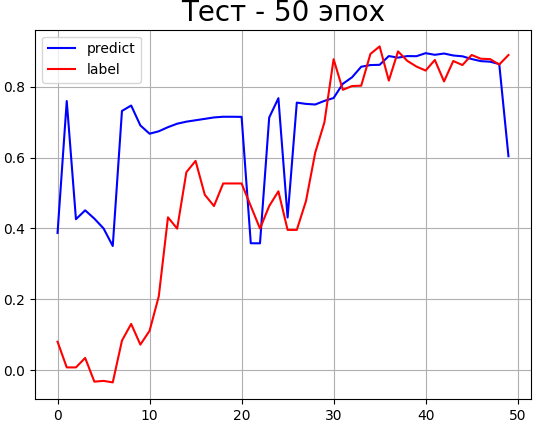
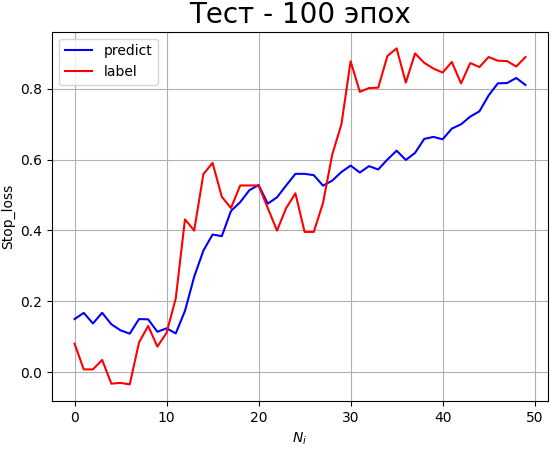
 

Рисунок 18 – Тест с 50 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 19 представлены результаты 100 эпох.

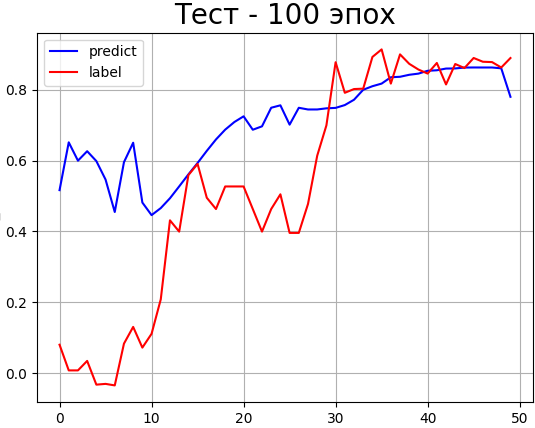
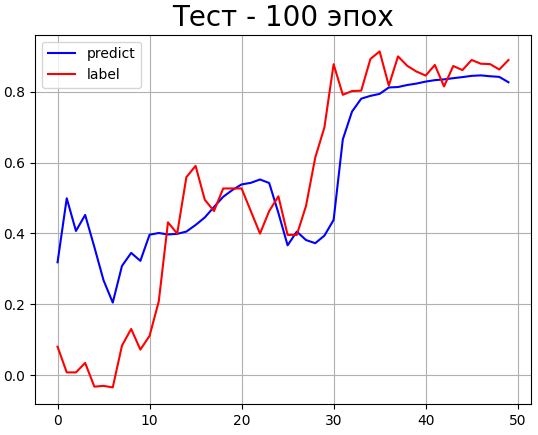
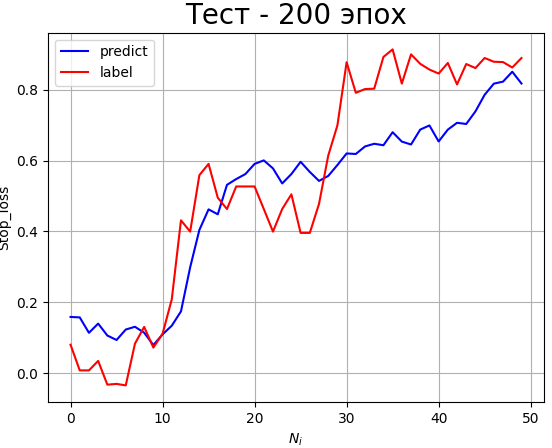
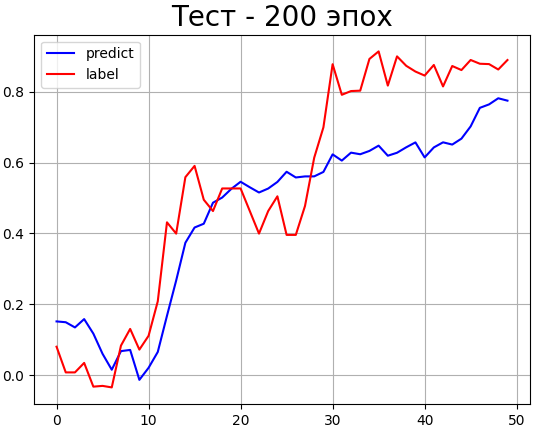
 

Рисунок 19 – Тест с 100 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 20 представлены результаты 200 эпох.

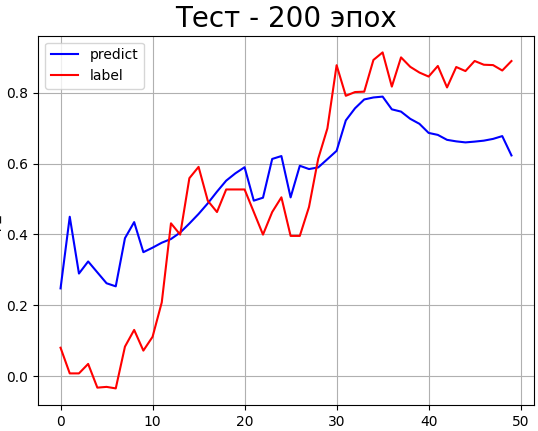
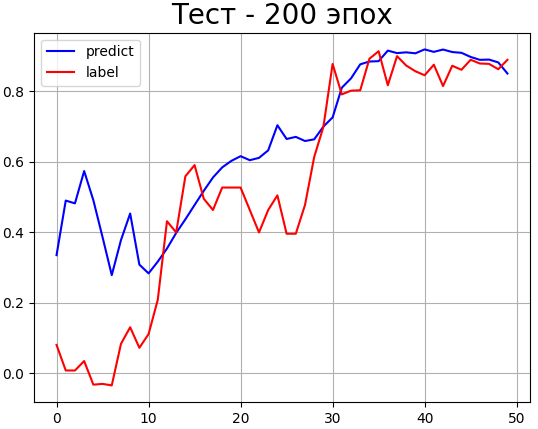
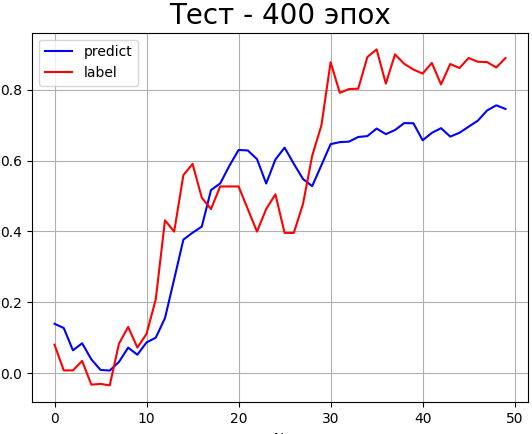
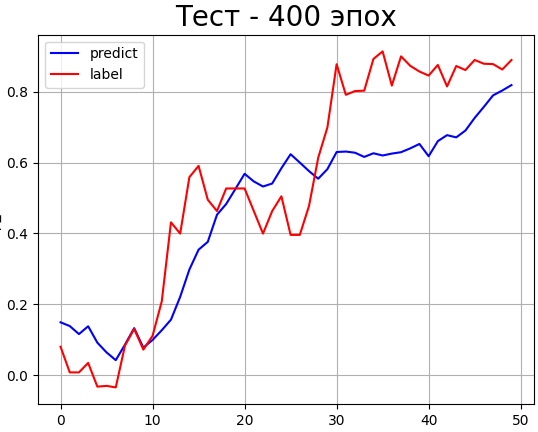
 

Рисунок 20 – Тест с 200 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 21 представлены результаты 400 эпох.

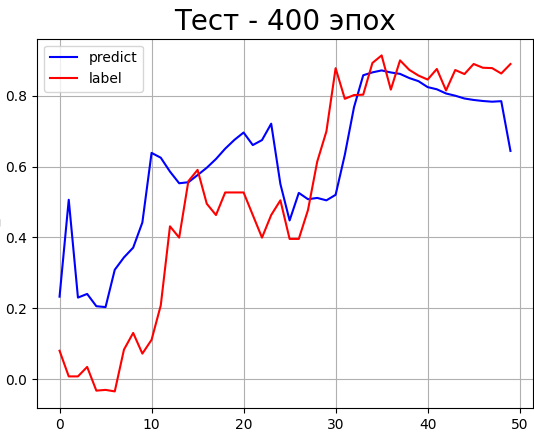
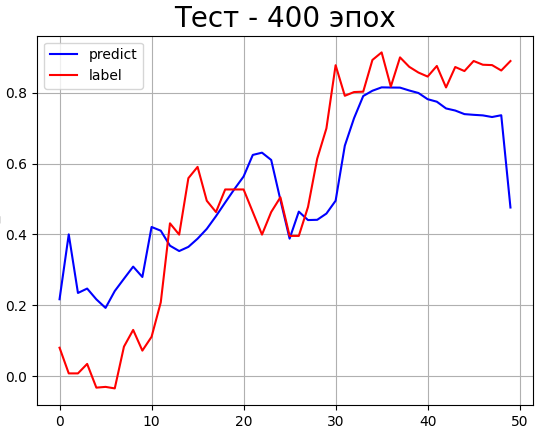
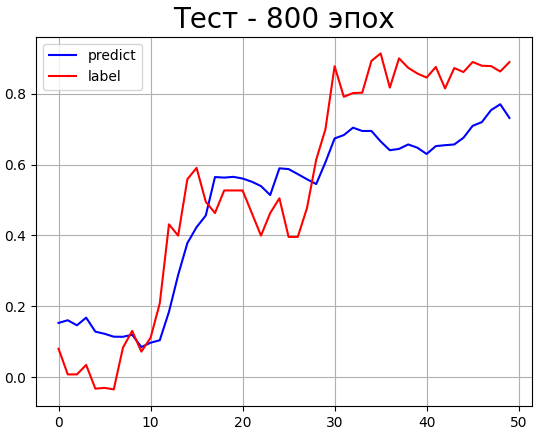
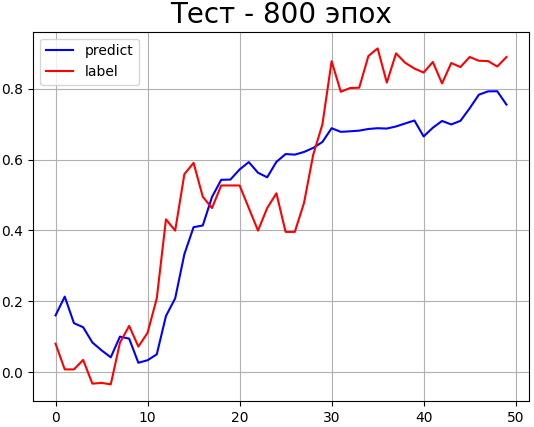
 

Рисунок 21 – Тест с 400 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 22 представлены результаты 800 эпох.

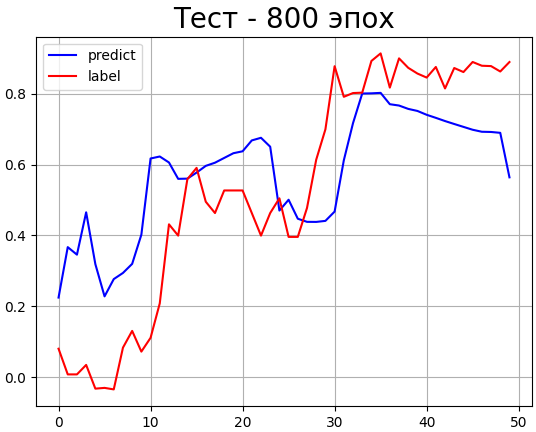
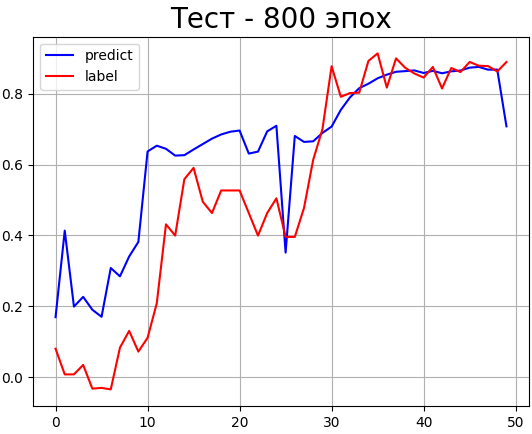
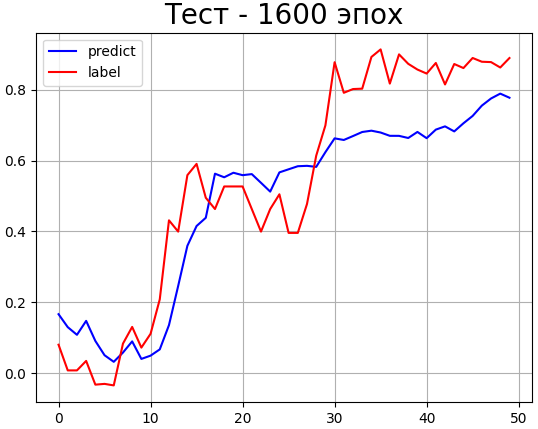
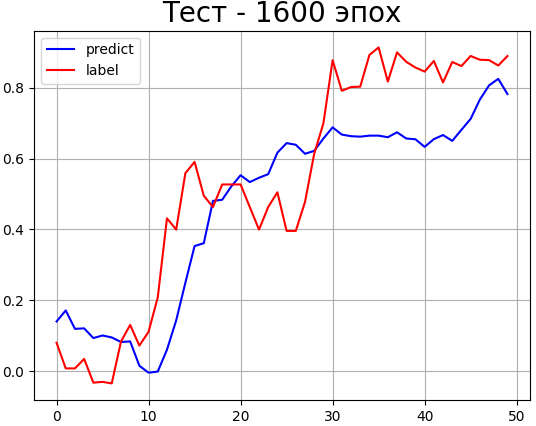
 

Рисунок 22 – Тест с 800 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 23 представлены результаты 1600 эпох.

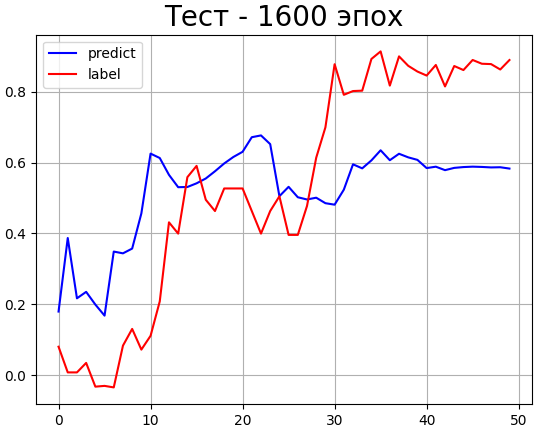
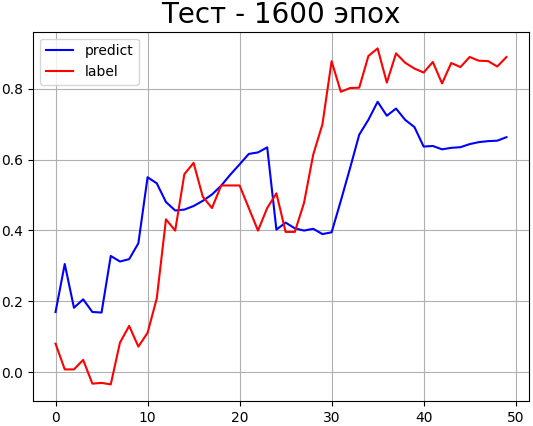
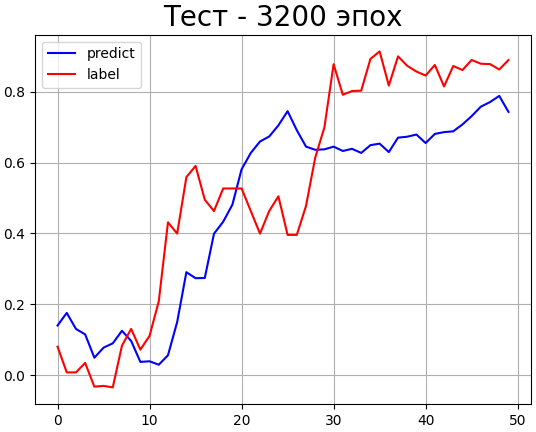
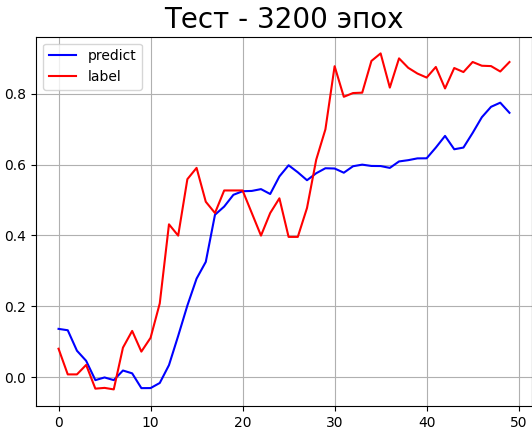
 

Рисунок 23 – Тест с 1600 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 24 представлены результаты 3200 эпох.

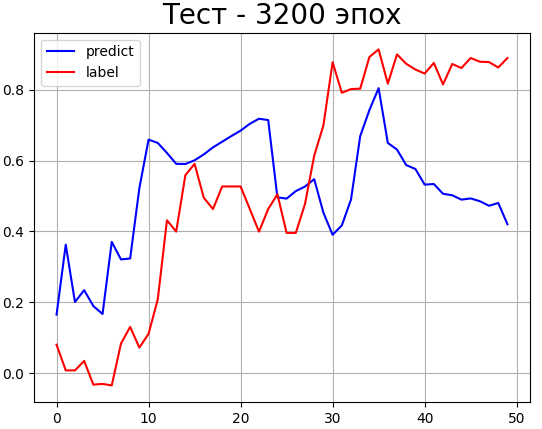
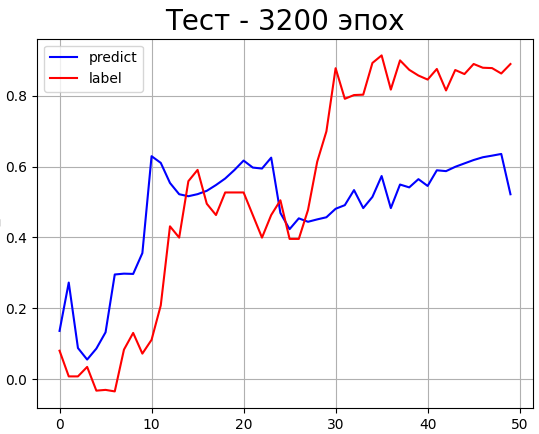
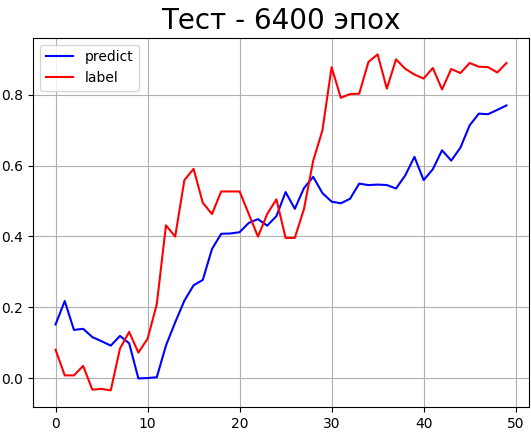
 

Рисунок 24 – Тест с 3200 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 25 представлены результаты 6400 эпох.

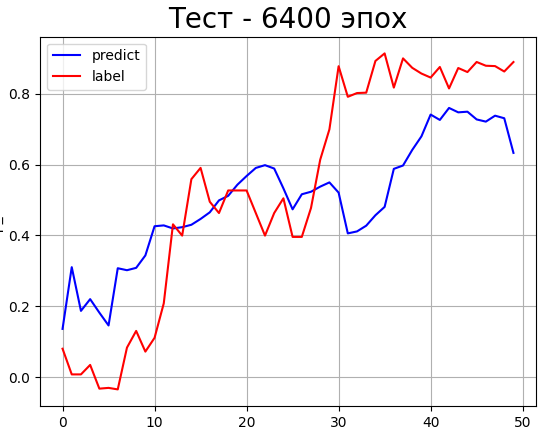
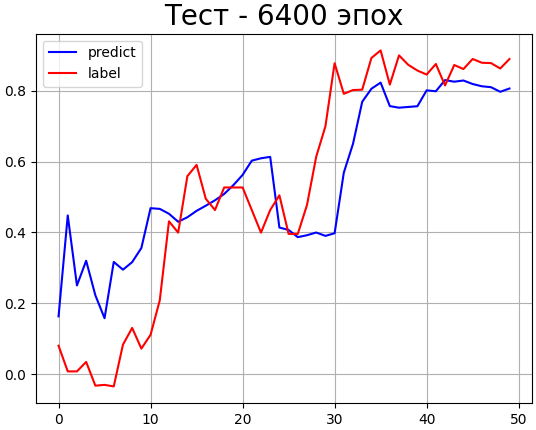
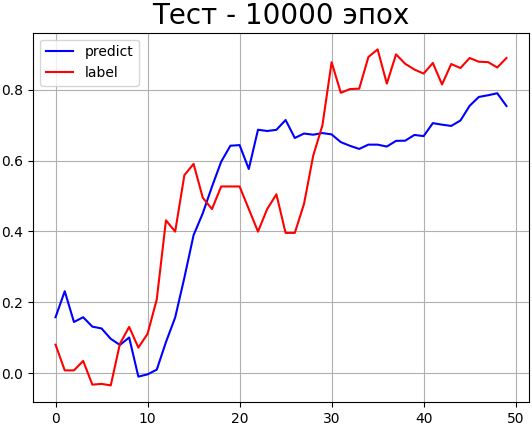
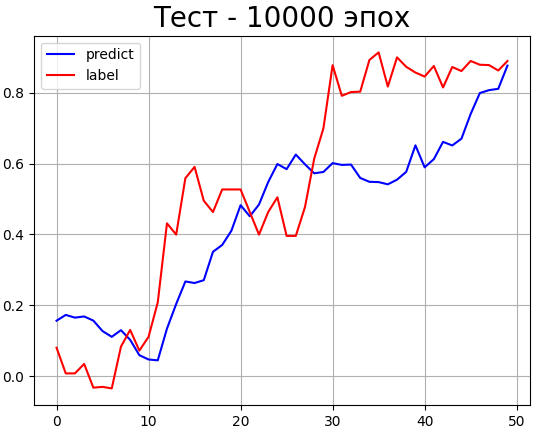
 

Рисунок 25 – Тест с 6400 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 26 представлены результаты 10000 эпох.

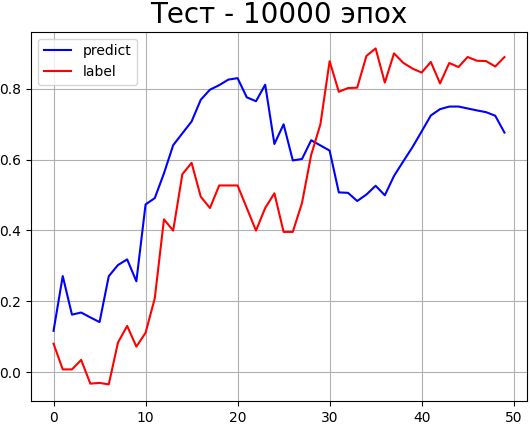
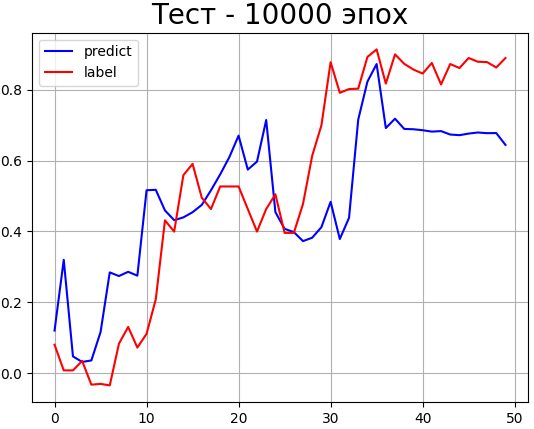
 

Рисунок 26 – Тест с 10000 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

Данные графики описывают результаты «предугадывания» НС в результате работы с тестовым датасетом (этот набор информации НС еще не видела и обучалась она не на нем).

По рисунку 27 и 28 можно увидеть, что время обучения НС от количества эпох зависит линейно.



Рисунок 27 – Зависимость времени обучения НС от количества эпох при использовании MSELoss



Рисунок 28 – Зависимость времени обучения НС от количества эпох при использовании MAELoss

По остальным метрикам, которые представлены с точными значениями в файле «KP.xlsx», можно понять, что MLP обучается в 1.5 раза быстрее, чем CNN и имеет более точный результат и на этапе обучения, и на этапе тестирования. Также можно увидеть, что отсутствует сильная разница между использованием MSELoss или MAELoss. Обе функции справляются со своей задачей.

* 1. Заключение по CNN

В ходе выполнения данной работы были достигнуты следующие цели:

1. Рассмотрено, что такое сверточная нейронная сеть;
2. Чем сверточная сеть отличается от простого линейного перцептрона;
3. Описана работа по составлению датасета на основе данных о тикере MSFT;
4. Описана работа по настройке и подбору нужных функции-оптимизации, функции-потерь, функции-активации и других настроек архитектуры НС;
5. Были проведены анализы работы сверточной НС при разных параметрах;
6. Было проведено сравнение с работой линейного перцептрона при разных параметрах.

Результаты экспериментов оказались достаточно «живые» и говорят о том, что сверточные нейронные сети способны анализировать не только изображения, но и большие матрицы данных. Однако биржа является предметом с очень сложными зависимостями и закономерностями и CNN не имеет возможностей точно предугадать следующее поведение биржи.

Оптимизация параметров модели и правильный выбор архитектуры CNN существенно влияют на достижение высокой точности классификации. Также важным аспектом является качество предварительной обработки данных, включая масштабирование, аугментацию и нормализацию, что дополнительно улучшает способность модели к обобщению.

Полученные результаты предоставляют основу для дальнейших исследований и применения сверточных нейронных сетей в различных областях. Возможным продолжением для данной работы, является проведение тестовых торгов на основе предсказанных нейронной сетью результатов. Таким способом можно будет корректно проанализировать прибыльность использования такой сети и ее актуальность.

1. Разработка архитектуры ViT

Нейронные сети являются мощным инструментом в области искусственного интеллекта и машинного обучения. Они моделируют работу человеческого мозга, позволяя компьютерам обрабатывать и анализировать сложные данные. Исследования по нейронным сетям имеют огромный потенциал для решения различного рода задач.

Один из важнейших инструментов машинного обучения — трансформеры. Популярность трансформеров взлетела до небес в связи с появлением больших языковых моделей вроде ChatGPT, GPT-4 и LLama. Эти модели созданы на основе трансформерной архитектуры и демонстрируют отличную производительность в понимании и синтезе естественных языков.

Помимо понимания и синтеза естественных языков, НС с трансформерной архитектурой имеют большой успех и спрос во многих других доменах машинного обучения, таких как распознавание человеческой речи, анализ изображений и видео.

Зрительные трансформеры (ViT) представляют собой одну из последних инноваций среди трансформерных архитектур НС. В отличие от сверточных нейронных сетей (CNN), которые используют свертки для извлечения признаков, ViT разбивают изображение или матрицу на патчи (небольшие куски) и обрабатывают их как последовательности, что позволяет модели эффективно захватывать глобальные взаимосвязи и контекст.

Подход ViT открывает новые возможности в задачах, требующих распознавания сложных взаимосвязей между элементами. Благодаря высокому уровню параллелизма и способности масштабироваться на большие объемы данных, ViT могут значительно улучшить точность и производительность моделей компьютерного зрения.

* 1. Устройство визуальной трансформерной НС

На рисунке 29 схематично показано внутреннее устройство простейшего визуального трансформера [8]:

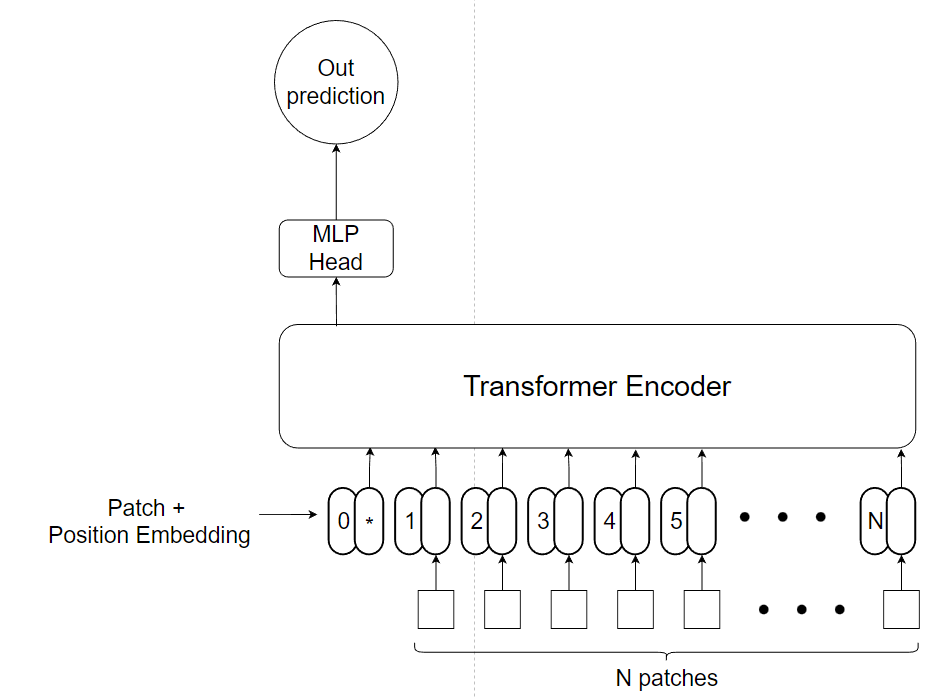


Рисунок 29 – Архитектура ViT НС

Перед реализацией архитектуры трансформера на практике, стоит узнать, что такое трансформер. Трансформер — это такой вид нейросетевой архитектуры, который хорошо подходит для обработки последовательностей данных. Пожалуй, самый популярный пример таких данных это предложение, которое можно считать упорядоченным набором слов. Однако, также часто используются трансформеры для обработки изображений и матриц данных.

Трансформеры создают цифровое представление каждого элемента последовательности, инкапсулируют важную информацию о нём и окружающем его контексте. Получившиеся представления затем можно передать в другие нейросети, которые воспользуются этой информацией для решения разных задач, в том числе для синтеза и классификации. Создавая такие информативные представления, трансформеры помогают последующим нейросетям лучше понять скрытые паттерны и взаимосвязи во входных данных. И поэтому они лучше синтезируют последовательные и взаимосвязанные результаты.

Главное преимущество трансформеров заключается в их способности обрабатывать длительные зависимости в последовательностях. Кроме того, они очень производительны, могут обрабатывать последовательности параллельно. Это особенно полезно в задачах вроде машинного перевода, анализа настроений и синтеза текста.

Для лучшего понимания, будем рассматривать ViT последовательно по рисунку 29.

* 1. Что поступает в трансформер

Прежде чем подать данные в трансформер, нужно сначала преобразовать их в последовательность векторов (токенов) — набор целых чисел, представляющих входные данные. Будем рассматривать сценарий использования трансформера для конкретно нашей задачи.

Чтобы получить вектор данных, возьмем нашу матрицу значений (окно дневных свечей размером N) и разобьем его на более мелкие кусочки – патчи. В одном патче будет содержаться информация об одной дневной свече. И наш патч будем записывать как вектор.

Получив последовательность целых чисел (вектор), представляющих входные данные, мы можем превратить их в эмбеддинги — это способ представления информации, облегчающий её обработку алгоритмами машинного обучения. Эмбеддинги передают смысл токенов в сжатом формате, представляя информацию в виде последовательности чисел. Сначала они синтезируются как случайная последовательность, а значимое представление формируется во время обучения. Однако у эмбеддингов есть наследственное ограничение: они не учитывают контекст, в котором синтезировались токены.

В зависимости от задачи, при превращении токенов в эмбеддинги нам может потребоваться сохранить порядок токенов. Это особенно важно для нашей задачи, иначе НС может потерять связь между днями. Чтобы этого не допустить, мы применяем к эмбеддингам позиционное кодирование (на рисунке 29 Positional Embedding).

Есть разные способы это сделать, но основная идея в том, что у нас есть ещё один набор эмбеддингов, представляющих положение каждого токена во входной последовательности. Этот второй набор комбинируется с эмбеддингами токенов [9]. Такой процесс схематично изображен на рисунке 30.

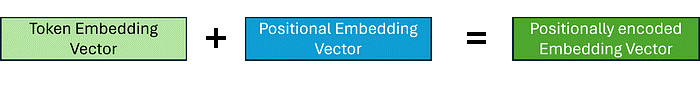


Рисунок 30 – Получаем итоговый вектор эмбеддинга.

Другая сложность в том, что у токенов могут быть разные значения в зависимости от соседних токенов. Чтобы легче было понять, рассматрим сценарий использования трансформера для обработки естественного языка:

* Андрей не любит арбуз, **он** слишком сладкий.
* Андрей не любит арбуз, **он** любит дыню.

Здесь слово «он» используется в двух абсолютно разных контекстах, поэтому имеют разные значения. В первом предложении слово «он» подразумевает арбуз. Во втором же – Андрей. Трансформер решает эту проблему с помощью механизма «Attention» или же «Внимание».

* 1. Механизм «внимания»

На рисунке 31 схематично изображено подробное устройство простейшего трансформера [10]:

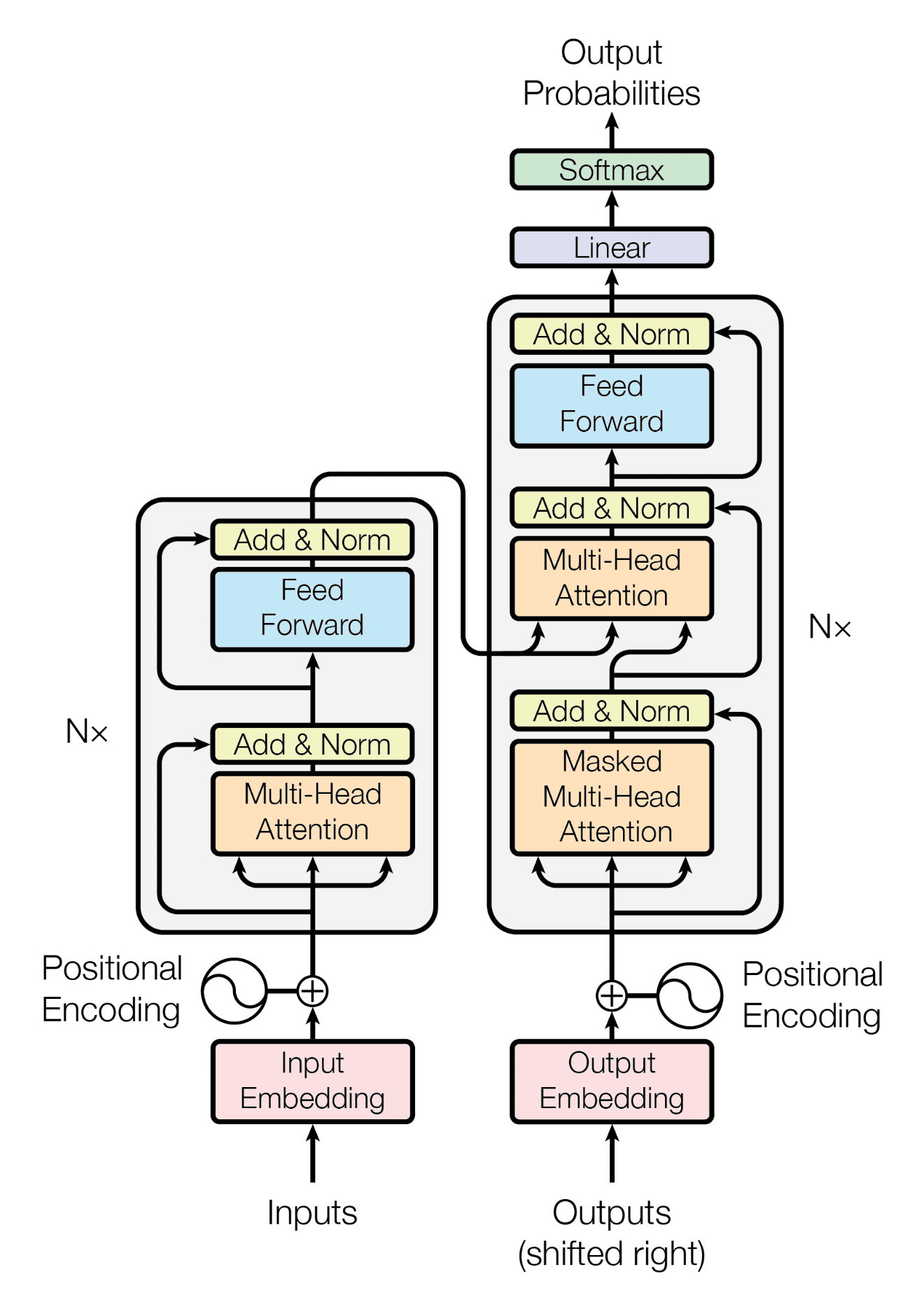


Рисунок 31 – Архитектура трансформерной НС.

Рассмотрим внимательно эту архитектуру и заметим, что большое количество блоков в ней отведены под «Attention» – внимание.

Пожалуй, самый важный механизм в трансформенной архитектуре — это внимание (на рисунке 31 – светло-оранжевые блоки). Он позволяет нейросети понять, какая часть входной последовательности наиболее релевантна задаче. Механизм внимания определяет для каждого токена последовательности, какие другие токены необходимы для его понимания в данном контексте. Прежде чем мы перейдем к тому, как это реализовано в трансформере, давайте сначала разберемся, чего пытается добиться механизм внимания.

Этот механизм можно представить как метод, который заменяет каждый эмбеддинг токена на эмбеддинг, содержащий информацию о соседних токенах, вместо использования одинакового эмбеддинга для каждого токена вне зависимости от контекста. Если бы мы знали, какие токены релевантны текущему, то узнать его контекст можно с помощью средневзвешенного — или, в общем случае, линейной комбинации — этих эмбеддингов.

На практике мы часто параллельно запускаем несколько таких блоков «внимания» (self-attention), чтобы трансформер одновременно обрабатывал разные части входной последовательности — это называют multi-head attention. Идея проста: выходы нескольких независимых блоков self-attention конкатенируются и передаются через линейный слой. Он позволяет модели комбинировать контекстуальную информацию из каждого блока внимания.

Мы поверхностно рассмотрели, что пытается добиться механизм внимания. Давайте теперь рассмотрим на листинге 14, как именно это реализовано в коде.

Листинг 14 – Класс слоя «Внимания»

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | class MultiHeadAttentionLayer(nn.Module):      def \_\_init\_\_(self, hid\_dim, n\_heads, dropout, device):          super().\_\_init\_\_()          assert hid\_dim % n\_heads == 0          self.hid\_dim = hid\_dim          self.n\_heads = n\_heads          self.head\_dim = hid\_dim // n\_heads          self.fc\_q = nn.Linear(hid\_dim, hid\_dim)          self.fc\_k = nn.Linear(hid\_dim, hid\_dim)          self.fc\_v = nn.Linear(hid\_dim, hid\_dim)          self.fc\_o = nn.Linear(hid\_dim, hid\_dim)          self.dropout = nn.Dropout(dropout)          self.scale =  torch.sqrt(torch.FloatTensor([self.head\_dim])).to(device)      def forward(self, query, key, value, mask = None):          batch\_size = query.shape[0]          #query = [batch size, query len, hid dim]          #key = [batch size, key len, hid dim]          #value = [batch size, value len, hid dim]          Q = self.fc\_q(query)          K = self.fc\_k(key)          V = self.fc\_v(value)          #Q = [batch size, query len, hid dim]          #K = [batch size, key len, hid dim]          #V = [batch size, value len, hid dim]          Q = Q.view(batch\_size, -1, self.n\_heads, self.head\_dim)  .permute(0, 2, 1, 3)          K = K.view(batch\_size, -1, self.n\_heads, self.head\_dim)  .permute(0, 2, 1, 3)          V = V.view(batch\_size, -1, self.n\_heads, self.head\_dim)  .permute(0, 2, 1, 3)          #Q = [batch size, n heads, query len, head dim]          #K = [batch size, n heads, key len, head dim]          #V = [batch size, n heads, value len, head dim]          energy = torch.matmul(Q, K.permute(0, 1, 3, 2))  / self.scale          #energy = [batch size, n heads, query len, key len]          if mask is not None:              energy = energy.masked\_fill(mask == 0, -1e10)          # [batch size, 1, trg len, trg len]          attention = torch.softmax(energy, dim = -1)          #attention = [batch size, n heads, query len, key len]          x = torch.matmul(self.dropout(attention), V)          #x = [batch size, n heads, query len, head dim]          x = x.permute(0, 2, 1, 3).contiguous()          #x = [batch size, query len, n heads, head dim]          x = x.view(batch\_size, -1, self.hid\_dim)          #x = [batch size, query len, hid dim]          x = self.fc\_o(x)          #x = [batch size, query len, hid dim]          return x, attention |

* 1. Полносвязная нейронная сеть

После блока «внимания», если посмотреть на рисунок 31, идет блок с полносвязной нейронной сетью (блок голубого цвета с надписью «Feed Forward»). Этот блок включает в себя самую простую нейронную архитектуру MLP (похожую мы рассматривали в первой части данной работы), обрабатывающую представления, полученные на выходе из слоя «внимания».

Этот слой нужен, чтобы среди связей слов в предложении найти самые нужные, найти какие-то закономерности и просто добавляет «свободные» нейроны для «мыслительных» процессов.

Реализация этого блока в коде представлена на листинге 15.

Листинг 15 – Класс полносвязной НС

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | class PositionwiseFeedforwardLayer(nn.Module):  def \_\_init\_\_(self, hid\_dim, pf\_dim, dropout):  super().\_\_init\_\_()  self.fc\_1 = nn.Linear(hid\_dim, pf\_dim)  self.fc\_2 = nn.Linear(pf\_dim, hid\_dim)  self.dropout = nn.Dropout(dropout)  def forward(self, x):  #x = [batch size, seq len, hid dim]  x = self.dropout(torch.relu(self.fc\_1(x)))  #x = [batch size, seq len, pf dim]  x = self.fc\_2(x)  #x = [batch size, seq len, hid dim]  return x |

* 1. Блок энкодер

После того, как мы поняли устройство блоков «внимания» и полносвязной НС, можно заметить по рисунку 31, что вмести эти блоки образуют один большой блок под названием «Transformer Encoder».

Именно такой блок нам нужен для нашего ViT и его реализация в коде представлена на листинге 16.

Листинг 16 – Класс энкодера

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | class TransformerEncoderBlock(nn.Module):  """Creates a Transformer Encoder block."""  # 2. Initialize the class with hyperparameters from Table 1 and Table 3  def \_\_init\_\_(self,  embedding\_dim: int = 4, # Hidden size D from Table 1 for ViT-Base  num\_heads: int = 10, # Heads from Table 1 for ViT-Base  mlp\_size: int = 128, # MLP size from Table 1 for ViT-Base  mlp\_dropout: float = 0.1, # Amount of dropout for dense layers from Table 3 for ViT-Base  attn\_dropout: float = 0.1): # Amount of dropout for attention layers  super().\_\_init\_\_()  # 3. Create MSA block (equation 2)  self.msa\_block= MultiheadSelfAttentionBlock(embedding\_dim=embedding\_dim, num\_heads=num\_heads, attn\_dropout=attn\_dropout)  # 4. Create MLP block (equation 3)  self.mlp\_block = MLPBlock(embedding\_dim=embedding\_dim,  mlp\_size=mlp\_size,  dropout=mlp\_dropout)  # 5. Create a forward() method  def forward(self, x):  # 6. Create residual connection for MSA block (add the input to the output)  x = self.msa\_block(x) + x  # 7. Create residual connection for MLP block (add the input to the output)  x = self.mlp\_block(x) + x  return x |

Таким образом мы получили готовую НС, которая показана на рисунке 32. Можно приступать к тестированию.

* 1. Проведение исследования

Чтобы НС хорошо работала требуется обучить её на данных из всего датасета. Прогоняя эти данные повторно множество раз, мы повышаем вероятность более правильного ответа НС.

Для исследования использован датасет, который содержит 200 сущностей для обучения и 50 сущности для тестов. Размер бача равен 25 сущностей.

В результате проведения прошлых тестов было выявлено, что функции-оптимизаторы MSE и MAE показывают схожую эффективность, поэтому было принято решение использовать для исследования только одну функцию, а именно MAE.

В процессе тестирования изменялись различные гиперпараметры НС, такие как: количество эпох, количество блоков энкодера, количество нейронов в полносвязном блоке, параметры дропаута различных блоков и количество голов.

Полные результаты исследования приведены в файле по ссылке: (<https://github.com/Relax-FM/Diploma/blob/main/Diploma/excel/DIPLOMA.xlsx>)

На рисунке 32 представлена часть результатов, полученных в результате изменения параметра дропаута после блока внимания:

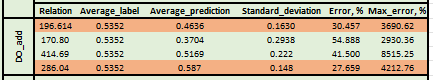


Рисунок 32 – Результаты исследования на этапе тестирования модели

На рисунках 33-37 изображены графики наилучших результатов, полученных в ходе исследования:



Рисунок 33 – График полученного результата

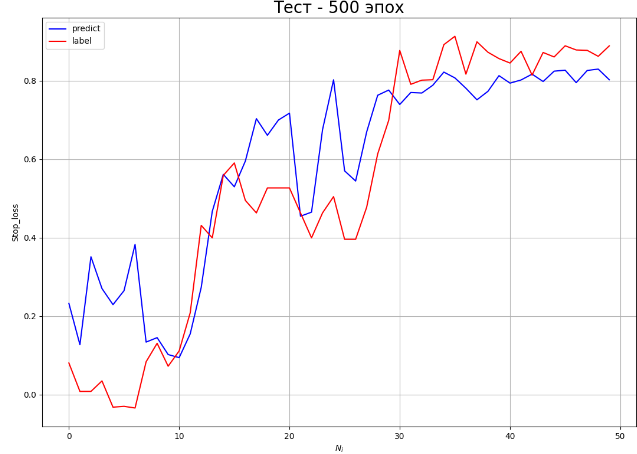


Рисунок 34 – График полученного результата

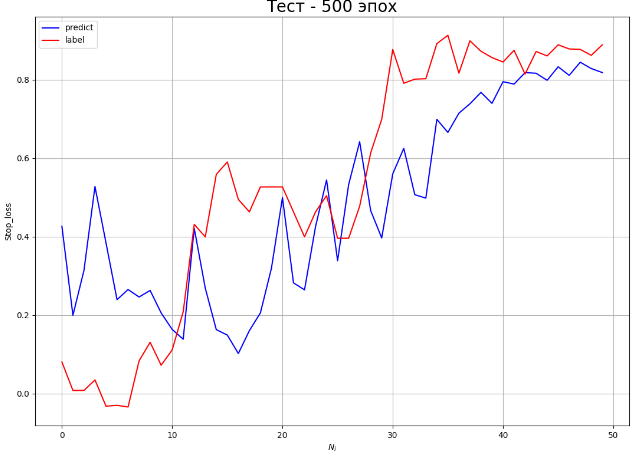


Рисунок 35 – График полученного результата

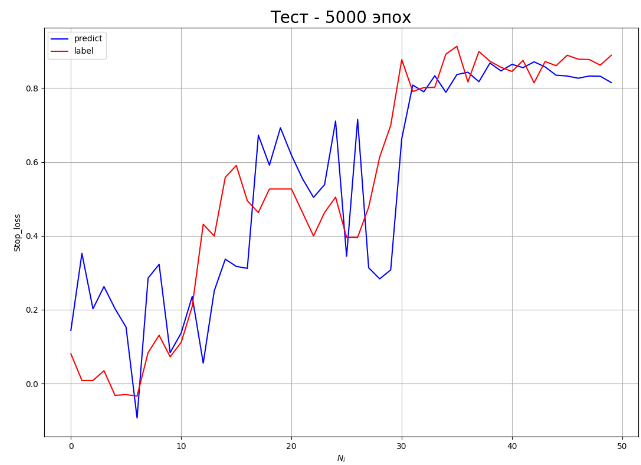


Рисунок 36 – График полученного результата



Рисунок 37 – График полученного результата

График красного цвета – требуемый (истинный) результат. График синего цвета – результат, предугаданный НС. По оси Х – номера сущностей из тестового датасета. По оси У – “stop loss”.

* 1. Заключение по ViT

В ходе выполнения данной работы были достигнуты следующие цели:

1. Рассмотрено, что такое трансформерная нейронная сеть;
2. Чем трансформер отличается от сверточной НС;
3. Были проведены анализы работы трансформерной НС при разных параметрах;
4. Было проведено сравнение с работой сверточной НС при разных параметрах.

Результаты экспериментов оказались достаточно говорят о том, что трансформерные сети могут обрабатывать не только текст, но и большие матрицы данных. Однако, технический анализ биржевых фондовых котировок является действительно сложно предсказуемым процессом.

Оптимизация параметров модели и правильный настройка гиперпараметров существенно влияют на достижение высокой точности классификации. Также важным аспектом является качество предварительной обработки данных, включая масштабирование, аугментацию и нормализацию, что дополнительно улучшает способность модели к обобщению.

По полученным данным исследования, в сравнение с результатами сверточной НС, можно сказать, что конкретно для данной задачи визуальный трансформер дает больше неточностей (похожих на осцилляции) на итоговом графике. Также само обучение ViT занимает значительно большего времени для достижения схожей с CNN точностью. Возможно, если увеличить размер датасета, получиться достигнуть большей точности и эффективности от ViT, но на данный момент результаты CNN кажутся более значительными.

1. Метод дообучению

В современном мире доступ к знаниям обширен и большинство людей обучаются чему-то новому каждый день. Новые знания помогают находить новые пути решения проблемы, помогают принимать решения основываясь на похожих ситуациях в прошлом и просто делают человека многогранным. Нейронные сети берут свое начало в биологии и анатомии человека и имеют внутри себя многие схожие системы и архитектуры. Работы, написанные мною до этого, использовали только простой (конечный) метод прямого обучения нейронной сети, в данной же работе, основываясь на примере реального человека, который получает новые знания каждый день, будет проведено исследование с использованием так называемого цикла дообучений НС. Каждый новый день у брокера появляется новая сущность ДС с информацией о выставленных им take-profit и stop-loss. Этой новой информации и будем дообучать НС.

Целью данного этапа является разработка и проведение исследования модуля дообучения для CNN и ViT. Также в данной работе будут найдены оптимальные настройки для такой НС.

* 1. Разработка модуля дообучения

Добавление в проект дообучения, предположительно, поможет увеличить точность предугаданных НС значений. Чтобы реализовать такой модуль, нужно разобраться, как будет работать наше дообучение. В данной работе я буду использовать такой вариант, который показан схематично на рисунке 38 [11]:

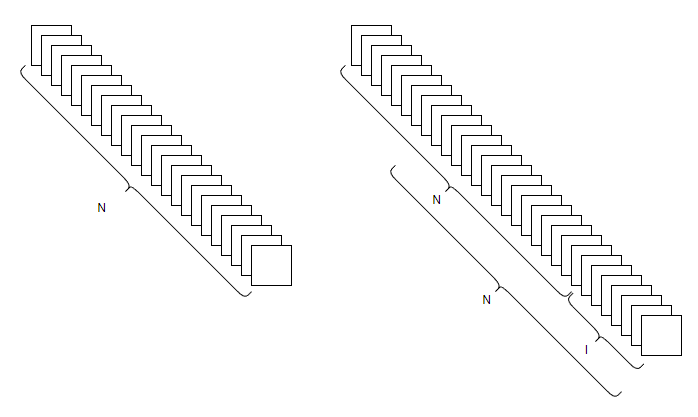


Рисунок 38 – Вариант дообучения.

Пусть у нас имеется ДС состоящий из N сущностей и начинающийся с 0-ой позиции. Обучим нашу НС на этом ДС используя обычное (конечное) обучение. После первого дня использования НС мы получим от брокера ещё одну сущность в наш ДС (того N+1). Не будем терять возможность получения нейронной сетью новой информации и дообучим её используя ДС, состоящий также из N сущностей, но уже начинающийся с 1 позиции (напомню, что у нас, на данный момент, всего N+1 сущность всего). На i-ый день у нас уже будет N+I сущностей и мы опять дообучим НС ДС из N сущностей но с i-ой позиции и так далее.

Такой метод дообучения позволит нам обучить сперва НС всем данным, что у нас были до начала работы брокера, а далее дообучать актуальными данными на основе уже поведения брокера.

* 1. Реализация класса создания датасетов

Как можно заметить, для такого обучения, нам часто придется создавать ДС, начинающиеся с разных позиций, поэтому удобно будет вынести эту функциональность в отдельный класс, как это реализовано в листинге 17 [12].

Листинг 17 – Класс создания датасетов.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | class Dataloader():  def \_\_init\_\_(self):  options\_path = 'config.yml'  with open(options\_path, 'r') as options\_stream:  options = yaml.safe\_load(options\_stream)  self.dataset\_options = options.get('dataset')  data\_path = self.dataset\_options.get('data\_file\_name')  EMA\_N = self.dataset\_options.get('EMA')  self.data = load\_data(data\_path, EMA\_N).data  def get\_dataloader(self, start: int | None = None, stop: int | None = None, additional: bool = False, test: bool = False) -> torch.utils.data.DataLoader:  """  Метод создает даталоадер с нужными настройками в зависимости от этапа(обучение/дообучение).\n  С нужной начальной позиции и до нужной конечной позиции.  :param start: С какой позиции  :param stop: По какую позицию  :param additional: Дообучение (Да/Нет)  :param test: Валидация (Да/Нет)  :return: loader: Возвращает даталоадер  """  loader\_options = self.dataset\_options.get('train\_loader')  if additional is True:  loader\_options = self.dataset\_options.get('additional\_loader')  if test is True:  loader\_options = self.dataset\_options.get('test\_loader')  if start is None:  start = loader\_options.get('start')  if stop is None:  stop = loader\_options.get('stop')  pdl = PreDataLoader(self.data, candle\_count=self.dataset\_options.get('candle\_count'),  start=start, stop=stop, normalization\_pred=self.dataset\_options.get('normalization'),  vers=self.dataset\_options.get('vers'), lbl=get\_label(self.dataset\_options.get('label'), False if additional or test else True))  ds = DataSet(pdl.batches)  loader = torch.utils.data.DataLoader(  ds, shuffle=loader\_options.get('shuffle'),  batch\_size=loader\_options.get('batch\_size'), num\_workers=loader\_options.get('num\_workers'),  drop\_last=loader\_options.get('drop\_last')  )  return loader |

В ходе выполнения нашей программы, будет создаваться много ДС, поэтому в данной реализации при инициализации класса (она происходит единоразово при вызове конструктора класса) в параметр self.data класса будут записаны все имеющиеся сущности, а уже позже, при вызове функции get\_dataloader(), из них будут выбраны сущности с нужной позиции и в нужном количестве.

* 1. Реализация метода дообучения

Так как теперь в работе участвует две разных версии обучения – простая (конечная) и дообучения, удобно будет сразу же вынести эти методы в функции.

Функции большого размера, их можно посмотреть в файле по ссылке (<https://github.com/RelaxFM/CNN_VKR/blob/master/Func/learning_version.py>). Функция простого (конечного) метода обучения названа в файле, как “feedforward”. Функция дообучения же названа, как “additional\_learning”.

Теперь вся необходимая функциональность у нас готова и можно переходить к экспериментальной части.

* 1. Проведение исследования

Возложим на исследование несколько целей:

1. Провести большое количество исследований;
2. Доказать эффективность использования метода дообучения;
3. Выявить какие-то закономерности;
4. Выявить оптимальные настройки.

Так как, при проведении предыдущего исследования на CNN с простым (конечным) методом обучения, функции-ошибки MAE и MSE показали схожие результаты по эффективности, то в данном исследовании будут приведены примеры только с одной из метрик. Будем использовать MAE – она немного лучше подходит для данного вида задач.

* 1. Исследования

Мы уже выбрали нужную нам реализацию метода дообучения и теперь нам остается только решить с какими настройками количества эпох проводить тесты. Так как в предыдущих работах для простого (конечного) метода обучения CNN были выбраны 10000, 6400, 3200, 1600, 800, 400, 200, 100, 50, то для простоты сравнения будем использовать эти же количества для тестов при использовании простого метода на CNN. Для дообучения сверточной НС же будем использовать другие величины эпох, а именно: 1000, 512, 256, 128, 64, 32, 16, так как дообучение это, возможно, бесконечное количество этапов простого обучения.

Для тестирования метода дообучения на ViT будем использовать: для этапа простого (конечного) обучения – настройки, хорошо показавшие себя на исследовании в третьей части данной работы, а для этапа дообучения будем использовать 25, 50, 75, 100 эпох.

Все полученные исследования можно найти по ссылке (<https://github.com/Relax-FM/Diploma/blob/main/Diploma/excel/DIPLOMA.xlsx>).

Часть из них представлена на рисунке 39 и 40 для CNN и на рисунке 41 и 42 для ViT:



Рисунок 39 – Часть эксперимента с простым (конечным) обучением CNN.



Рисунок 40 – Часть эксперимента с дообучением CNN после простого обучения с рисунка 39.

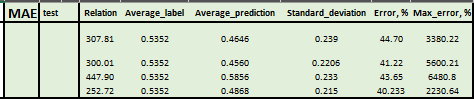


Рисунок 41 – Часть эксперимента с простым (конечным) обучениемViT.

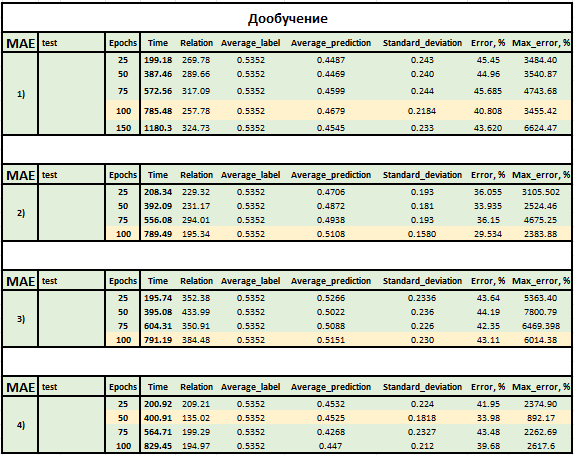


Рисунок 42 – Часть эксперимента с дообучением ViT после простого обучения с рисунка 41.

* 1. Эффективность метода дообучения

Как можно увидеть по метрикам или графикам, результаты, полученные после дообучения выглядят более точными, нежели результаты, полученные после простого обучения. Это говорит о том, что использование метода дообучения приводит к увеличению аналитических способностей НС и повышению точности.

На рисунках 43 и 44 представлены графики для сравнения до дообучения и после:

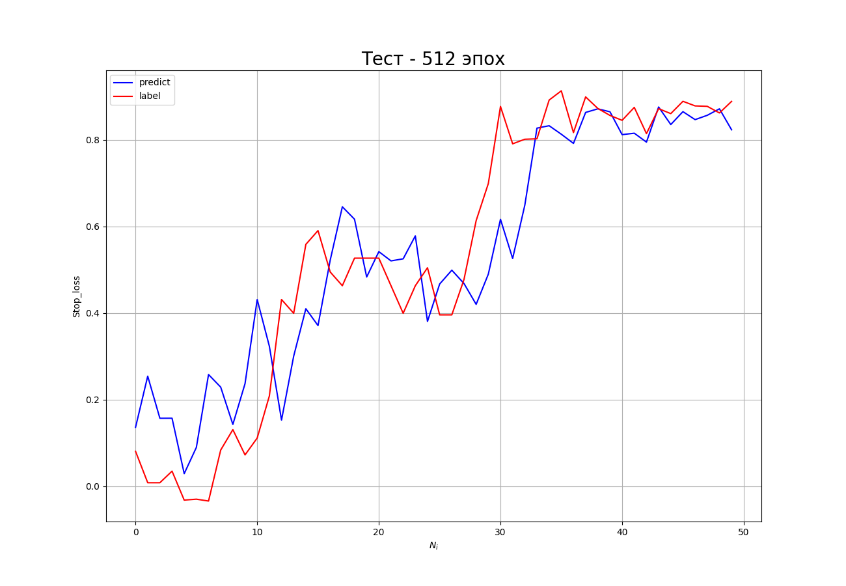
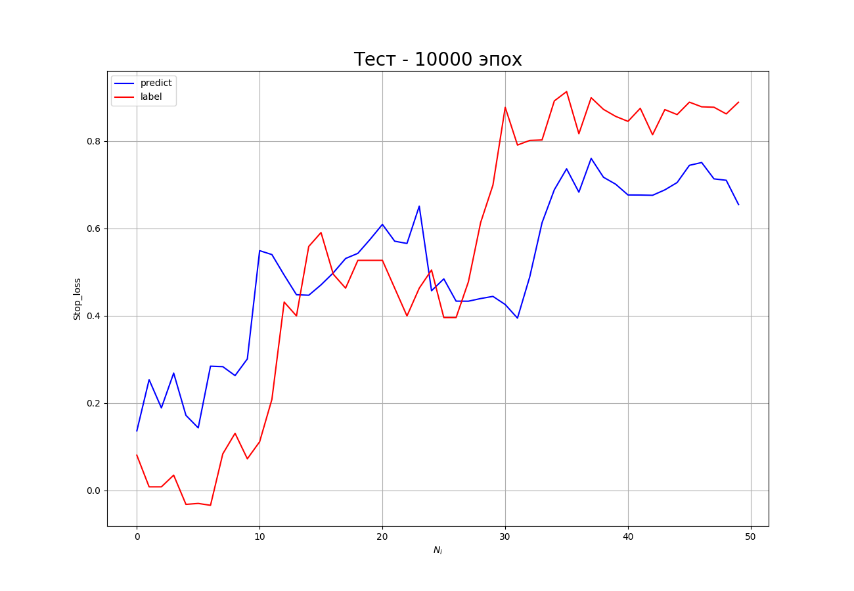


Рисунок 43 – Слева изображен график до дообучения, справа – после.

(Сверху подписано количество эпох, использованных на этапе простого обучения и дообучения, соответственно)

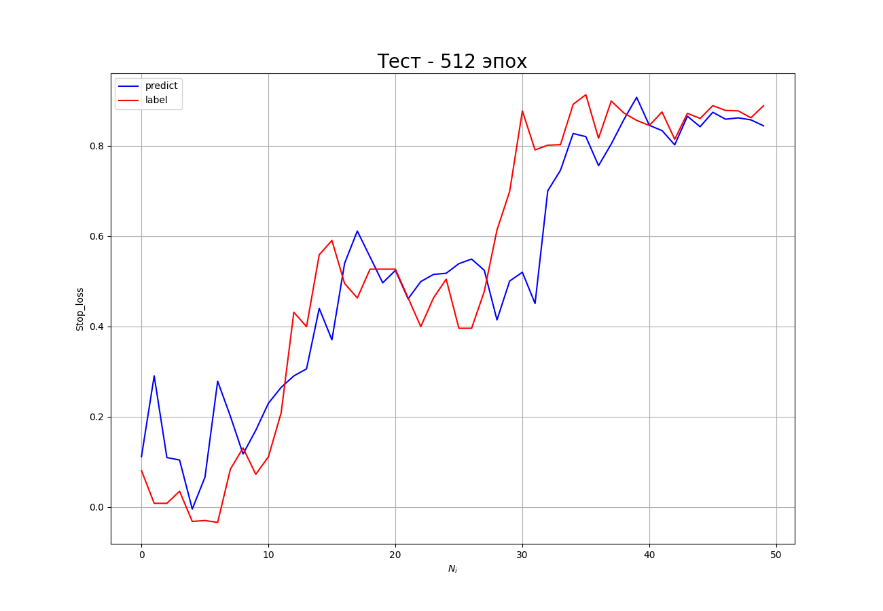
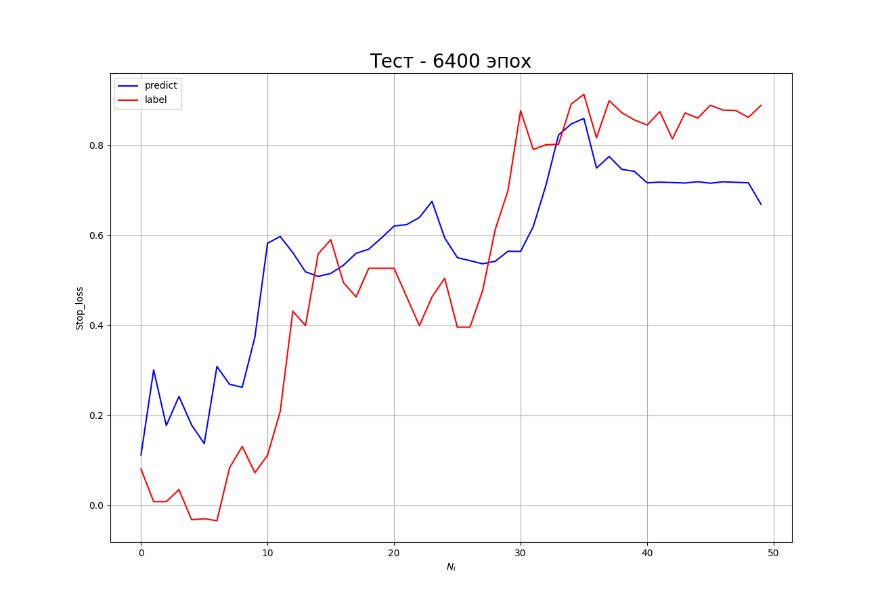


Рисунок 44 – Слева изображен график до дообучения, справа – после.

По графикам можно также заметить, что после дообучения, график предсказаний (выделен синим цветом) расположен более близко к графику истинных значений. Следовательно, является более точным.

* 1. Закономерности

При проведении исследования, удалось выяснить, что при малом количестве эпох на этапе обучения в левой части графика появляются большие осцилляции, связанные, вероятно, с недостаточным обучением НС. Начиная с 1600 эпох для этапа простого обучения осцилляции начинают проходить и появляется более плавный график.

На этапе дообучения достаточно хорошие результаты показывают метрики и графики для 512 эпох. Можно выбрать это число эпох, как оптимальное.

В целом, этап дообучения дает хорошую аппроксимацию истинного графика. Даже при условии, что на этапе простого обучения используют малое количество эпох, этап дообучения сглаживает график предсказаний, тем самым приближая его к истинному графику.

Также удалось выяснить, что этап дообучения позволяет повысить точность примерно на 25%, но при этом придется пожертвовать временем. Это время в масштабах целого дня не сильно большое и им можно пренебречь для такого рода задач.

* 1. Оптимальные настройки

Сложно подобрать оптимальные настройки под задачи похожего типа, но для рассматриваемого случая оптимальными настройками можно назвать:

1. Для этапа простого обучения CNN – 6400 эпох.
2. Для этапа дообучения CNN– 512 эпох.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были изучены различные архитектуры НС, такие как: MLP, CNN, ViT. Была изучена теоретическая часть по корректной реализации собственных датасетов и даталоадеров, используя модули PyTorch. Также была изучена и большая часть инструментария данного модуля. Написаны краткие обзоры полученных теоретических знаний.

Реализованы полномасштабные НС на изученных архитектурах нейронных сетей. Написаны модули для удобного выбора и использования различных функций оптимизации и функций потерь.

Проведено множество исследований на реализованных НС. Сделаны соответствующие выводы по их эффективности. Все результаты этих исследований представлены по ссылке: (<https://github.com/Relax-FM/Diploma/blob/main/Diploma/excel/DIPLOMA.xlsx>).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Pandas Documentation [Электронный ресурс] – URL: [https://pandas.pydata.org/docs/](https://pandas.pydata.org/docs/%20) (дата обращения: 09.02.2024 - 22.03.2024)
2. PyTorch Tutorials [Электронный ресурс] – URL: <https://pytorch.org/tutorials/> (дата обращения: 09.02.2024 - 10.05.2024).
3. PyTorch Documentation [Электронный ресурс] – URL: [https://pytorch.org/docs/stable/index.html](https://pytorch.org/docs/stable/index.html%20) (дата обращения: 09.02.2024 - 10.05.2024).
4. «Сверточная нейронная сеть с нуля» [Электронный ресурс] ‒ URL: <https://programforyou.ru/poleznoe/convolutional-network-from-scratch-part-zero-introduction> (дата обращения: 19.02.2024 - 24.03.2024).
5. Блог о сверточной нейронной сети [Электронный ресурс] ‒ URL: <https://habr.com/ru/companies/skillfactory/articles/565232/> (дата обращения: 20.03.2024).
6. CNN from scratch [электронный ресурс] – URL: <https://www.pycodemates.com/2023/07/build-a-cnn-from-scratch-using-python.html> (дата обращения: 23.03.2024)
7. Курс Deep Learning (семестр 1, весна 2023): продвинутый поток [Электронный ресурс] – URL: <https://stepik.org/course/135003/syllabus> (дата обращения: 25.03.2024 - 10.05.2024)
8. Vision transformer from scratch [электронный ресурс] – URL: <https://debuggercafe.com/vision-transformer-from-scratch/> (дата обращения: 20.04.2024)
9. Хабр статья про трансформер [электронный ресурс] – URL: <https://habr.com/ru/companies/mws/articles/770202/> (дата обращения: 24.04.2024 – 26.04.2024.)
10. Курс Deep Learning (семестр 2, весна 2024) [Электронный ресурс] – URL: <https://stepik.org/course/196142/syllabus> (дата обращения: 25.04.2024 - 25.05.2024)
11. Курс машинного обучения. К.В.Воронцов [Электронный ресурс] – URL: [http://www.machinelearning.ru/wiki/index.php?title=Машинное\_обучение\_(курс\_лекций,\_К.В.Воронцов)](http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_(курс_лекций,_К.В.Воронцов)%20) (дата обращения: 27.04.2024 - 29.04.2024)
12. Сайт с датасетами [Электронный ресурс] – URL: [https://www.kaggle.com/datasets/](https://www.kaggle.com/datasets/d4rklucif3r/cat-and-dogs?select=dataset) (дата обращения: 21.05.2024).

ПРИЛОЖЕНИЕ А. Графическая часть выпускной квалификационной работы

В графическую часть выпускной квалификационной работы входят:

Слайд 1. Цель работы

Слайд 2. MLP

Слайд 3. Результаты работы MLP

Слайд 4. CNN

Слайд 5. Результаты работы CNN

Слайд 6. ViT

Слайд 7. Результаты работы ViT

Слайд 8. Дообучение

Слайд 9-10. Результаты работы дообучения

Слайд 11. Заключение