МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Самарский национальный исследовательский университет

имени академика С.П. Королева»

Институт информатики и кибернетики

Факультет информатики

Кафедра программных систем

ОТЧЁТ

по лабораторной работе № 3

«Создание чат-бота с искусственным интеллектом»

по курсу «Нейронные сети»

Выполнил:

Каликин М.А.

гр. 6132-020402D

Проверила:

Жданова А.Н.

Самара 2022

СОДЕРЖАНИЕ

[1 Математическая постановка задачи 3](#_Toc122040527)

[2 Обоснование выбора нейронной сети 4](#_Toc122040528)

[3 Архитектура сети, параметры, методы обучения 5](#_Toc122040529)

[4 Структурная схема алгоритма 11](#_Toc122040530)

[5 Вычислительные эксперименты 14](#_Toc122040531)

[6 Результаты работы 16](#_Toc122040532)

[ЗАКЛЮЧЕНИЕ 17](#_Toc122040533)

[ССЫЛКИ 18](#_Toc122040534)

[Листинг программы 19](#_Toc122040535)

# **1 Математическая постановка задачи**

Добавить заранее заготовленные диалоги с ботом, обучить бота: создать сложные диалоги, распознавание смысла сообщения пользователя; работа с контекстом: поддержание ветки диалога на протяжении нескольких сообщений.

За основу будет использоваться набор данных по тематике киновсленной «Звездные войны»

# **2 Обоснование выбора нейронной сети**

В данной лабораторной работе будет использоваться Neural conversational model потому, что данная модель больше подходит для задачи написания чат-бота.

Есть два типа чат-ботов:

На основе команд: чат-боты, которые работают по заранее определенным правилам и могут отвечать только на ограниченные запросы или вопросы. Пользователям необходимо выбрать вариант, чтобы определить свой следующий шаг.

Интеллектуальные чат-боты/чат-боты с искусственным интеллектом: чат-боты, которые используют машинное обучение и понимание естественного языка, чтобы понимать язык пользователя, и достаточно умны, чтобы учиться на разговорах со своими пользователями. Вы можете общаться с помощью текста, речи или даже взаимодействовать с чат-ботом с помощью графических интерфейсов.

Все чат-боты подпадают под концепцию NLP (обработки естественного языка). NLP состоит из двух вещей:

NLU (понимание естественного языка): способность машин понимать человеческий язык, например русский.

NLG (Генерация естественного языка): способность машины генерировать текст, похожий на предложения, написанные человеком.

Например, пользователь задает вопрос чат-боту: «Эй, что сегодня в новостях?» Чат-бот разбивает предложение пользователя на две части: намерение и сущность. Целью этого предложения может быть get\_news, поскольку оно относится к действию, которое хочет выполнить пользователь. Сущность сообщает конкретные подробности о намерении, поэтому сущностью будет «сегодня». Таким образом, модель машинного обучения используется для распознавания намерений и объектов чата.

# **3 Архитектура сети, параметры, методы обучения**

Для выполнения работы использовались nltk, PyTorch а также вспомогательные библиотеки Python. С помощью данных фреймворков была построены модели для обработка естественного языка.

Далее был подготовлен файл с заранее подготовленными диалогами, т.н намерениями и ответами в формате JSON.

**Предварительная подготовка данных**

Импортируем все необходимые модули и загрузим файл данных в формате JSON.

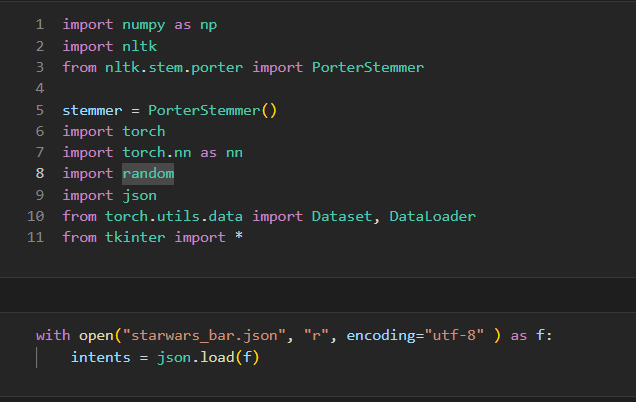


Рис 1. Загрузка файла с данными

Создадим пользовательские функции для работы с естественным языком с использованием nltk.

Для обучения использовался так называемый стемминг. Стемминг – используется для нормализации слов и подразумевает морфологический разбор слова с нахождением общей для всех его грамматических форм. Использовалась модель Портера Стеммера из библиотеки nlltk.

Разобьём каждое слово в предложениях и добавим его в массив, для этого использовалась nltk.word\_tokenize(), которая преобразовала строку одного предложения в список слов. Например, если передать «привет, как дела", будет возвращено ["привет", "как", "дела"]. Слова передавались в нижнем ригистре, это делалась для того ,чтобы такие слова как «Хорошо» и «хорошо» не были помечены как разные слова.

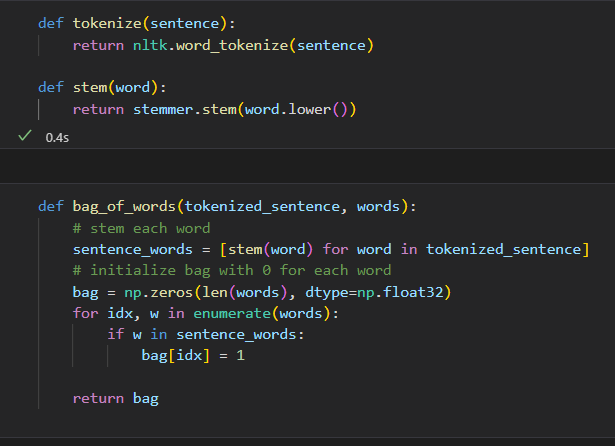
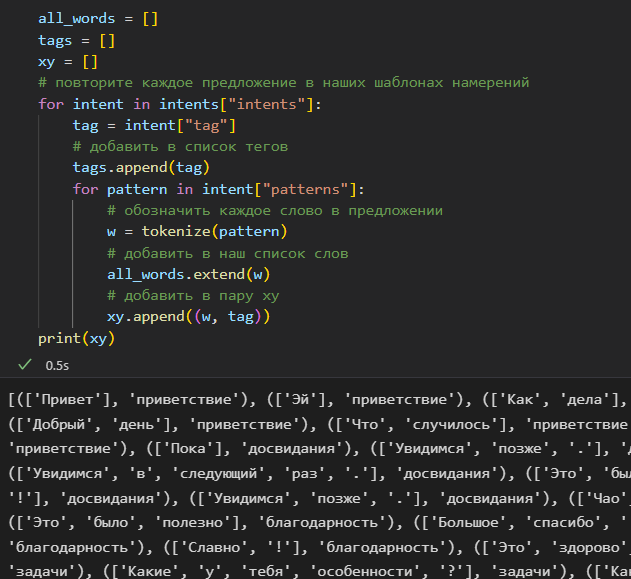


Рис 2. Использование nltk.word\_tokenize

Чтобы получить правильную информацию, мы будем распаковывать starwars\_bar.json с помощью следующего кода:

  
Рис 3. Распаковка starwars\_bar.json

Прим помощи стемминга разделим слова на отдельные списки, и тем самым получим 97 паттернов.

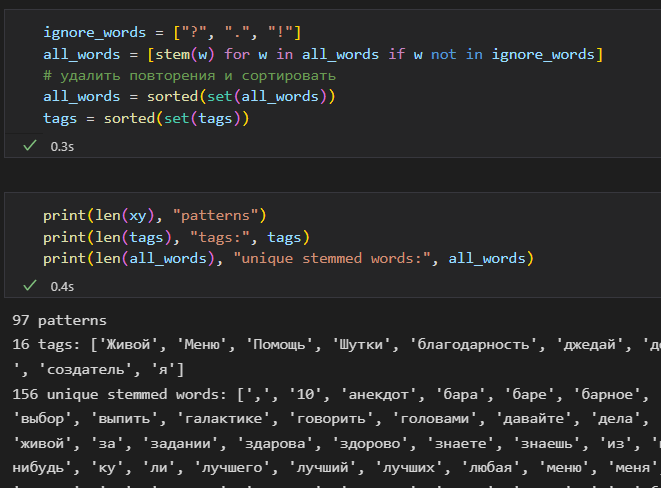


Рис 3. Стемминг и получение паттернов

**Модель PyTorch**Преобразуем данные в формат, который наша модель PyTorch может легко понять при помощи следующего кода

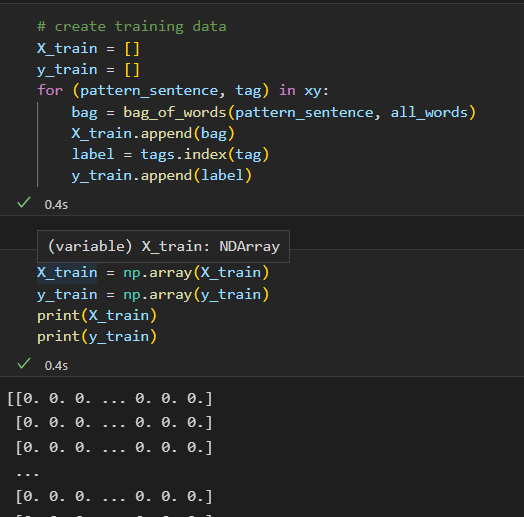
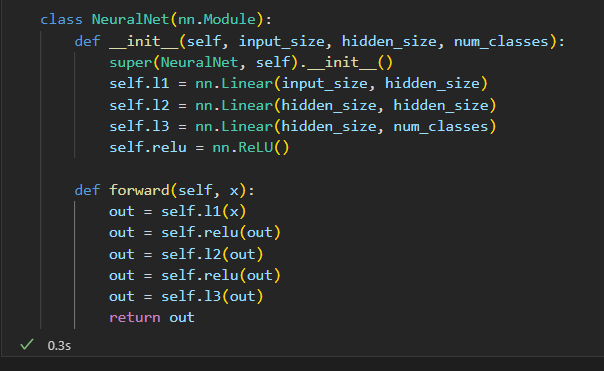


Рис 4. Преобразование формата данных для модели PyTorch

Создадим класс для реализации для нейронной сети. Это будет нейронная сеть с прямой связью, которая будет иметь 3 линейных слоя, и будем использовать функцию активации «ReLU». Также настроим модель и ее слои с использование класса NN.Module

  
Рис 5. Создание модели

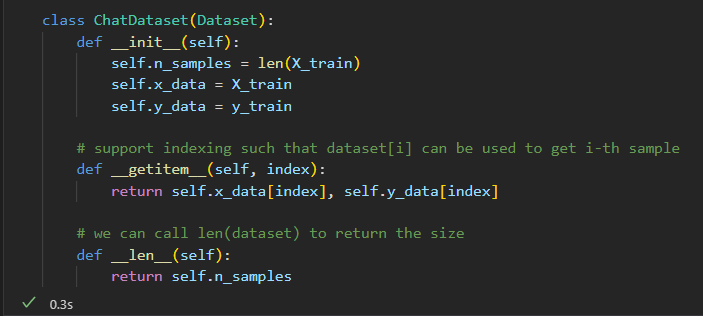
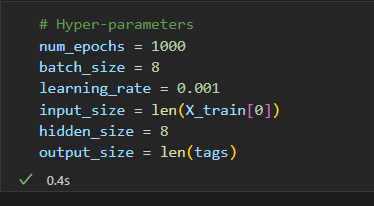
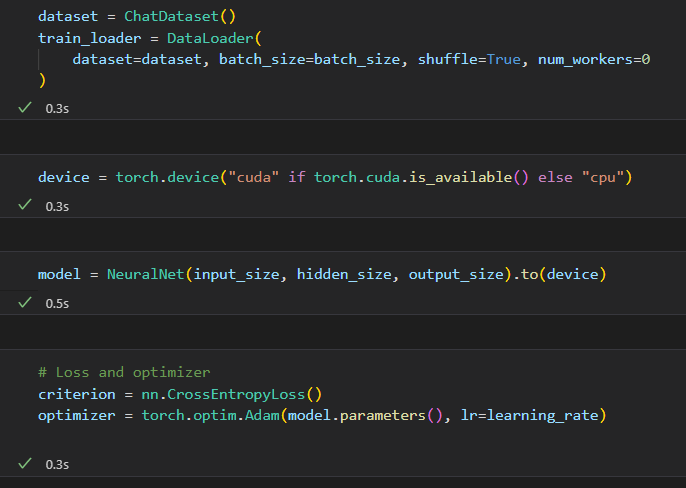
Напишем класс ChatDataset и используем функции getitem и setitem для доступа к элементам списка, словаря, массива.  


Рис 6. Класс с функциями getitem и setitem

Определим набор гиперпараметров эти параметры необходимо инициализировать перед обучением модели.   
  
Рис 6. Гиперпараметры

Создадим экземпляры модели, функций потерь и оптимизатор.

  
Рис 7. Экземпляры модели, функций потерь и оптимизатор

Выполним обучение модели   
Рис 8. Обучение модели

Сохраним данные и загрузим их для работы чат-бота  


Рис 9. Работа с данными

Обученная модель готова, но т.к данные о тренировках были ограничены, с чат-ботом можем обсуждать только несколько тем.  


Рис 10. Работа чат-бота на уровне кода

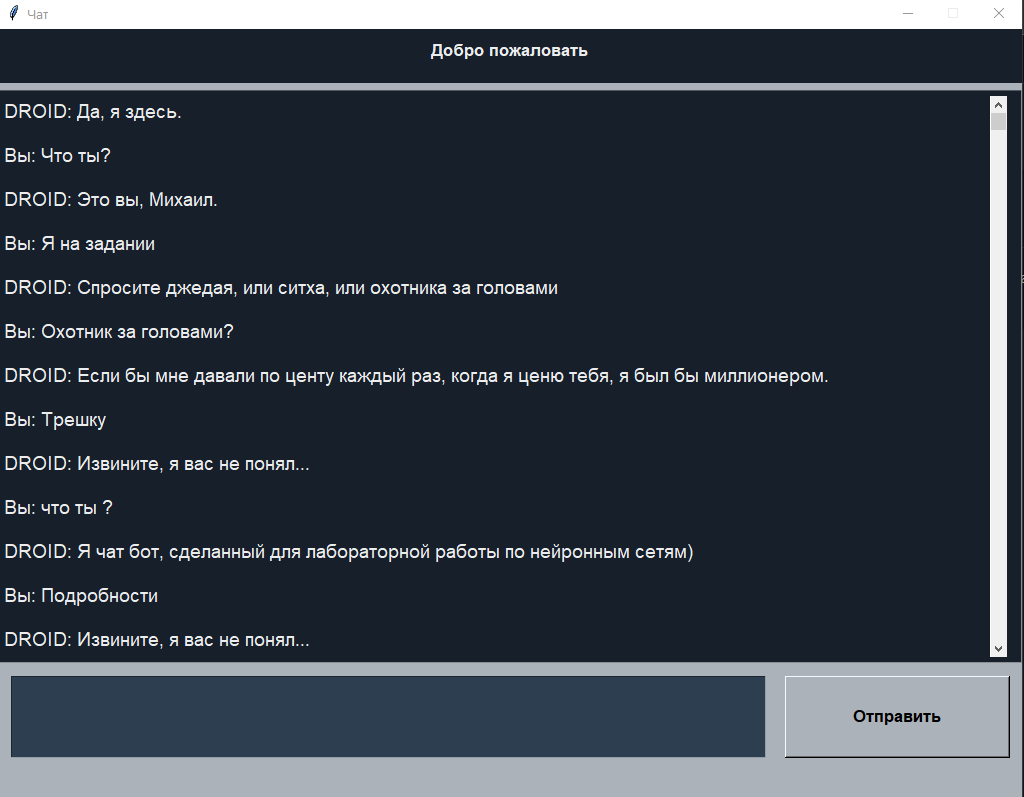


Рис 10. Интерфейс чат-бота

# **4 Структурная схема алгоритма**

Использовался NLP – обработка естественного языка для распознавания письменной речи, с соответствующей обработки текста.

Сначала необходимо преобразовать текст для восприятия компьютером выполнив следующие действия:

Очистка. Из текста удаляются бесполезные для машины данные. Это большинство знаков пунктуации, особые символы, скобки, теги и пр. Некоторые символы могут быть значимыми в конкретных случаях. Например, в тексте про экономику знаки валют несут смысл.

Препроцессинг. Дальше наступает большой этап предварительной обработки — препроцессинга. Это приведение информации к виду, в котором она более понятна алгоритму. Популярные методы препроцессинга:

* приведение символов к одному регистру, чтобы все слова были написаны с маленькой буквы;
* токенизация — разбиение текста на токены. Так называют отдельные компоненты — слова, предложения или фразы;
* тегирование частей речи — определение частей речи в каждом предложении для применения грамматических правил;
* лемматизация и стемминг — приведение слов к единой форме. Стемминг более грубый, он обрезает суффиксы и оставляет корни. Лемматизация — приведение слов к изначальным словоформам, часто с учетом контекста;
* удаление стоп-слов — артиклей, междометий и пр.;
* спелл-чекинг — автокоррекция слов, которые написаны неправильно.

Векторизация. После предобработки на выходе получается набор подготовленных слов. Но алгоритмы работают с числовыми данными, а не с чистым текстом. Поэтому из входящей информации создают векторы — представляют ее как набор числовых значений.

Популярные варианты векторизации — «мешок слов» и «мешок N-грамм». В «мешке слов» слова кодируются в цифры. Учитывается только количество слова в тексте, а не их расположение и контекст. N-граммы — это группы из N слов. Алгоритм наполняет «мешок» не отдельными словами с их частотой, а группами по несколько слов, и это помогает определить контекст.

Применение алгоритмов машинного обучения. С помощью векторизации можно оценить, насколько часто в тексте встречаются слова. Но большинство актуальных задач сложнее, чем просто определение частоты — тут нужны продвинутые алгоритмы машинного обучения. В зависимости от типа конкретной задачи создается и настраивается своя отдельная модель.

Алгоритмы обрабатывают, анализируют и распознают входные данные, делают на их основе выводы. Это интересный и сложный процесс, в котором много математики и теории вероятностей.

Методы выбирают согласно задаче.

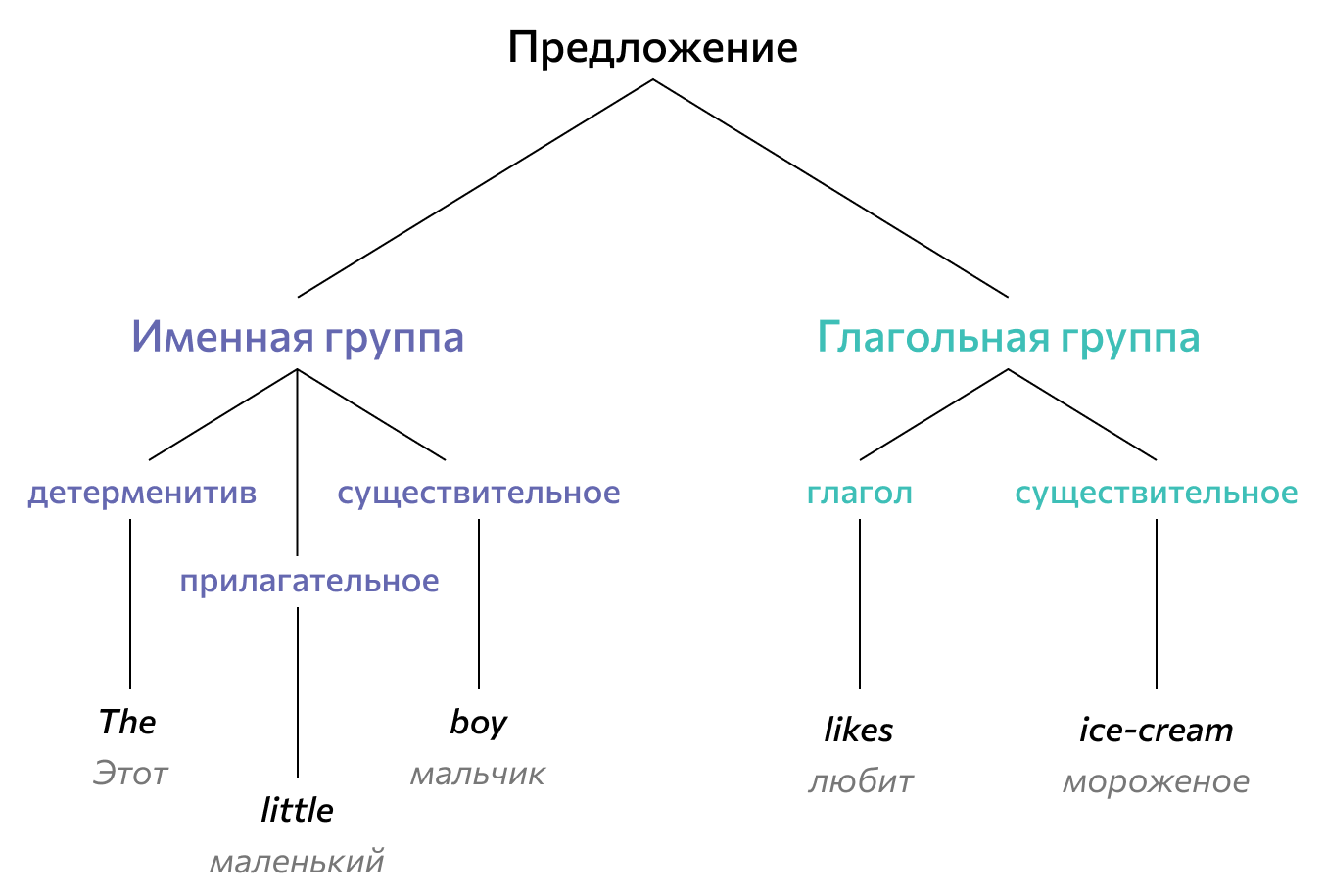
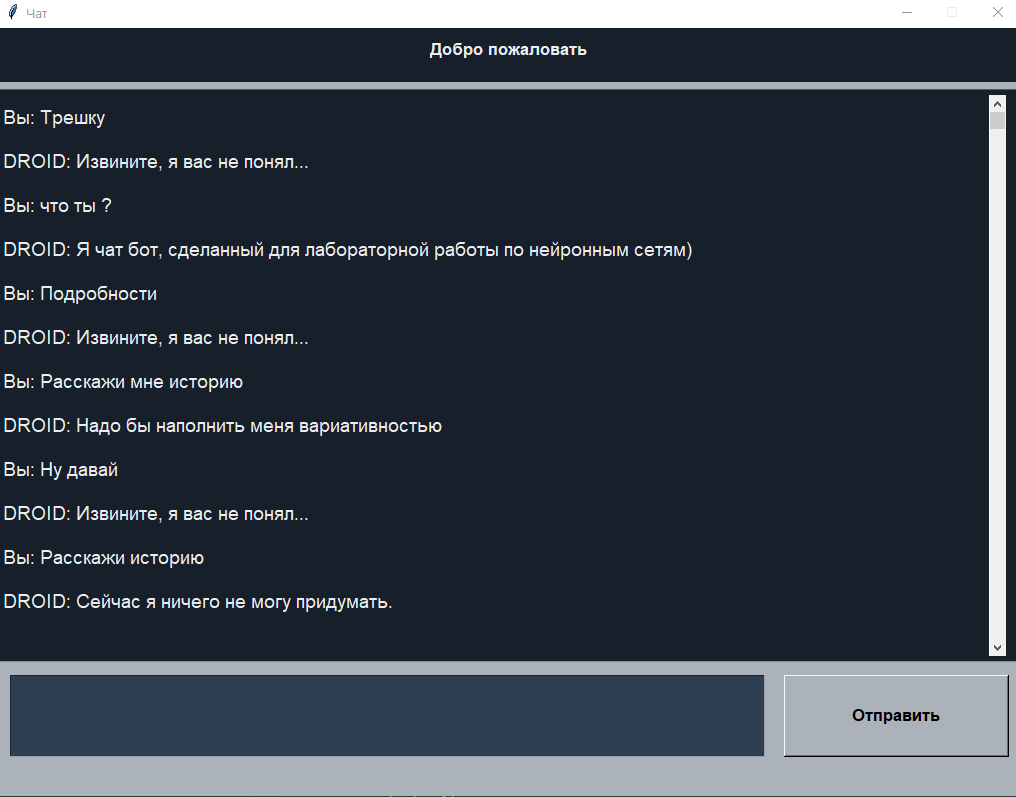


Рис 11. Методы для алгоритма

# **5 Вычислительные эксперименты**

**Проверим модель**Рис 12. Проверка модели

  
Рис 13. Проверка чат бота на вариативность ответов

# **6 Результаты работы**

Была обучена CNN сеть с использованием NLP.

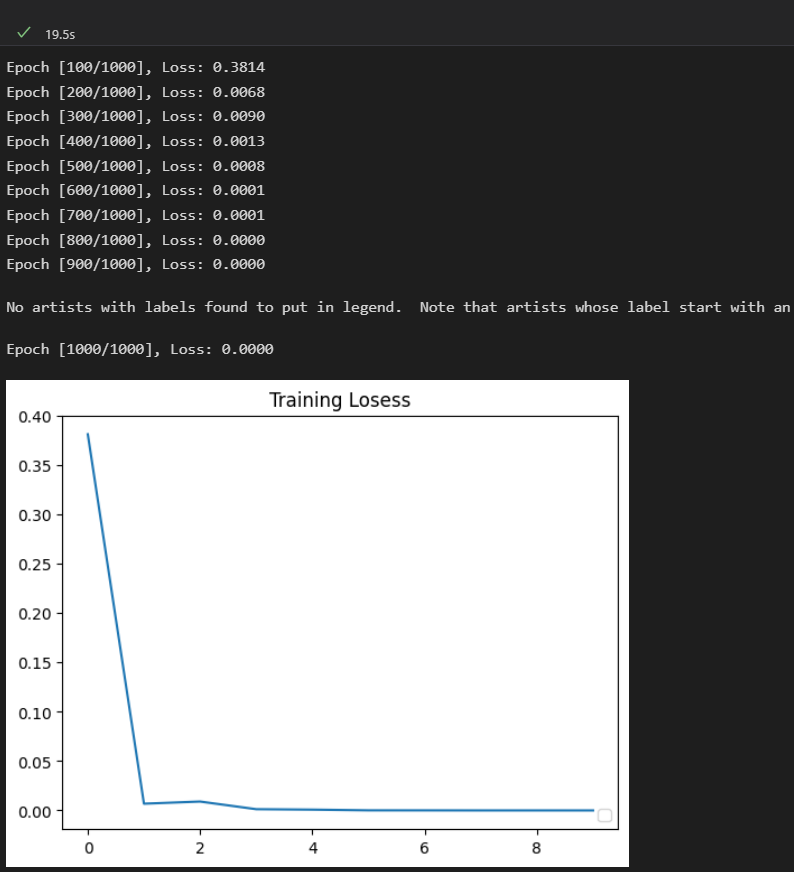


Рис 13. Показана функция потерь.

Модель можно переобучить, и сделать улучшение.

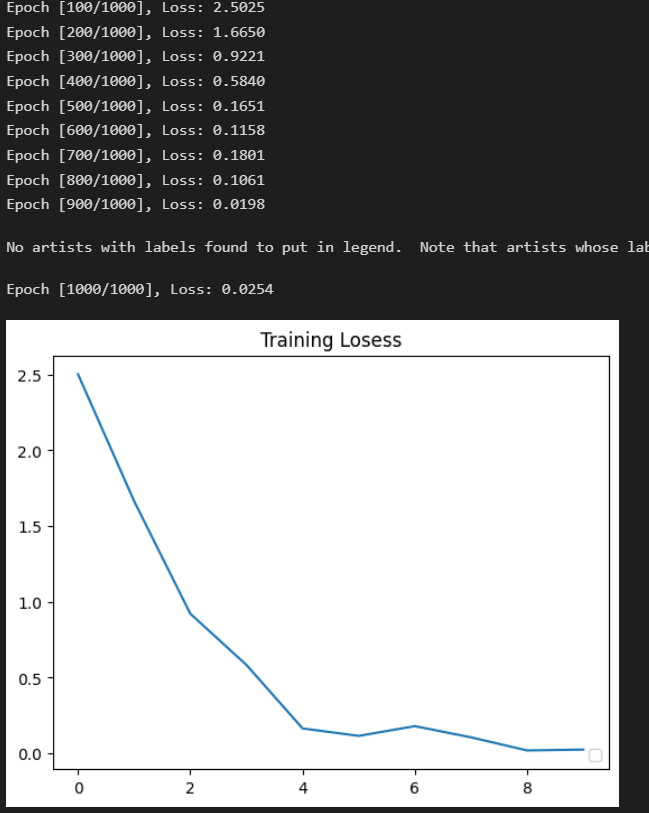


Рис 14. Показана функция потерь при использовании SigMoid

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы были изучены основные принципы работы с обучением нейронных сетей с распознаванием и обработкой естественного языка (Natural Language Processing, NLP). Обучена на подготовленном заранее специальном файле, состоящем из 1,593 слов, с применением намерений и ответов.

Проведены эксперименты с чат ботом, чат бот может отвечать вариативно.

В результате получилось приложение, которое можно адаптировать под свои нужды, например сделать чат бота с большей вариативностью ответов, и содержательными разговорами.

# **ССЫЛКИ**

Github: https://github.com/MikhailKalikin/Kalikin\_MA\_NN\_LAB3

# **Листинг программы**

import numpy as np

import nltk

from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

import torch

import torch.nn as nn

import random

import json

from torch.utils.data import Dataset, DataLoader

from tkinter import \*

from matplotlib import pyplot as plt

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('corpus')

with open("starwarsintents.json", "r") as f:

intents = json.load(f)

def tokenize(sentence):

return nltk.word\_tokenize(sentence)

def stem(word):

return stemmer.stem(word.lower())

def bag\_of\_words(tokenized\_sentence, words):

# stem each word

sentence\_words = [stem(word) for word in tokenized\_sentence]

# initialize bag with 0 for each word

bag = np.zeros(len(words), dtype=np.float32)

for idx, w in enumerate(words):

if w in sentence\_words:

bag[idx] = 1

return bag

all\_words = []

tags = []

xy = []

# loop through each sentence in our intents patterns

for intent in intents["intents"]:

tag = intent["tag"]

# add to tag list

tags.append(tag)

for pattern in intent["patterns"]:

# tokenize each word in the sentence

w = tokenize(pattern)

# add to our words list

all\_words.extend(w)

# add to xy pair

xy.append((w, tag))

print(xy)

# stem and lower each word

ignore\_words = ["?", ".", "!"]

all\_words = [stem(w) for w in all\_words if w not in ignore\_words]

# remove duplicates and sort

all\_words = sorted(set(all\_words))

tags = sorted(set(tags))

print(len(xy), "patterns")

print(len(tags), "tags:", tags)

print(len(all\_words), "unique stemmed words:", all\_words)

# create training data

X\_train = []

y\_train = []

for (pattern\_sentence, tag) in xy:

# X: bag of words for each pattern\_sentence

bag = bag\_of\_words(pattern\_sentence, all\_words)

X\_train.append(bag)

# y: PyTorch CrossEntropyLoss needs only class labels, not one-hot

label = tags.index(tag)

y\_train.append(label)

X\_train = np.array(X\_train)

y\_train = np.array(y\_train)

print(X\_train)

print(y\_train)

class NeuralNet(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size, num\_classes):

super(NeuralNet, self).\_\_init\_\_()

self.l1 = nn.Linear(input\_size, hidden\_size)

self.l2 = nn.Linear(hidden\_size, hidden\_size)

self.l3 = nn.Linear(hidden\_size, num\_classes)

self.relu = nn.ReLU()

def forward(self, x):

out = self.l1(x)

out = self.relu(out)

out = self.l2(out)

out = self.relu(out)

out = self.l3(out)

# no activation and no softmax at the end

return out

class ChatDataset(Dataset):

def \_\_init\_\_(self):

self.n\_samples = len(X\_train)

self.x\_data = X\_train

self.y\_data = y\_train

# support indexing such that dataset[i] can be used to get i-th sample

def \_\_getitem\_\_(self, index):

return self.x\_data[index], self.y\_data[index]

# we can call len(dataset) to return the size

def \_\_len\_\_(self):

return self.n\_samples

# Hyper-parameters

num\_epochs = 1000

batch\_size = 8

learning\_rate = 0.001

input\_size = len(X\_train[0])

hidden\_size = 8

output\_size = len(tags)

dataset = ChatDataset()

train\_loader = DataLoader(

dataset=dataset, batch\_size=batch\_size, shuffle=True, num\_workers=0

)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = NeuralNet(input\_size, hidden\_size, output\_size).to(device)

# Loss and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)

# Train the model

for epoch in range(num\_epochs):

for (words, labels) in train\_loader:

words = words.to(device)

labels = labels.to(dtype=torch.long).to(device)

# Forward pass

outputs = model(words)

# if y would be one-hot, we must apply

# labels = torch.max(labels, 1)[1]

loss = criterion(outputs, labels)

# Backward and optimize

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if (epoch + 1) % 100 == 0:

print(f"Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}")

data = {

"model\_state": model.state\_dict(),

"input\_size": input\_size,

"hidden\_size": hidden\_size,

"output\_size": output\_size,

"all\_words": all\_words,

"tags": tags,

}

FILE = "data.pth"

torch.save(data, FILE)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

with open("starwarsintents.json", "r") as json\_data:

intents = json.load(json\_data)

FILE = "data.pth"

data = torch.load(FILE)

input\_size = data["input\_size"]

hidden\_size = data["hidden\_size"]

output\_size = data["output\_size"]

all\_words = data["all\_words"]

tags = data["tags"]

model\_state = data["model\_state"]

model = NeuralNet(input\_size, hidden\_size, output\_size).to(device)

model.load\_state\_dict(model\_state)

model.eval()

bot\_name = "MR.ROBOT1"

def get\_response(msg):

sentence = tokenize(msg)

X = bag\_of\_words(sentence, all\_words)

X = X.reshape(1, X.shape[0])

X = torch.from\_numpy(X).to(device)

output = model(X)

\_, predicted = torch.max(output, dim=1)

tag = tags[predicted.item()]

probs = torch.softmax(output, dim=1)

prob = probs[0][predicted.item()]

if prob.item() > 0.75:

for intent in intents["intents"]:

if tag == intent["tag"]:

return random.choice(intent["responses"])

return "Не понимаю о чем ты говоришь?..."

BG\_GRAY = "#ABB2B9"

BG\_COLOR = "#17202A"

TEXT\_COLOR = "#EAECEE"

FONT = "Helvetica 14"

FONT\_BOLD = "Helvetica 13 bold"

class ChatApplication:

def \_\_init\_\_(self):

self.window = Tk()

self.\_setup\_main\_window()

def run(self):

self.window.mainloop()

def \_setup\_main\_window(self):

self.window.title("Chat")

self.window.resizable(width=False, height=False)

self.window.configure(width=470, height=550, bg=BG\_COLOR)

# head label

head\_label = Label(

self.window,

bg=BG\_COLOR,

fg=TEXT\_COLOR,

text="Добро пожаловать!",

font=FONT\_BOLD,

pady=10,

)

head\_label.place(relwidth=1)

# tiny divider

line = Label(self.window, width=450, bg=BG\_GRAY)

line.place(relwidth=1, rely=0.07, relheight=0.012)

# text widget

self.text\_widget = Text(

self.window,

width=20,

height=2,

bg=BG\_COLOR,

fg=TEXT\_COLOR,

font=FONT,

padx=5,

pady=5,

)

self.text\_widget.place(relheight=0.745, relwidth=1, rely=0.08)

self.text\_widget.configure(cursor="arrow", state=DISABLED)

# scroll bar

scrollbar = Scrollbar(self.text\_widget)

scrollbar.place(relheight=1, relx=0.974)

scrollbar.configure(command=self.text\_widget.yview)

# bottom label

bottom\_label = Label(self.window, bg=BG\_GRAY, height=80)

bottom\_label.place(relwidth=1, rely=0.825)

# message entry box

self.msg\_entry = Entry(bottom\_label, bg="#2C3E50", fg=TEXT\_COLOR, font=FONT)

self.msg\_entry.place(relwidth=0.74, relheight=0.06, rely=0.008, relx=0.011)

self.msg\_entry.focus()

self.msg\_entry.bind("<Return>", self.\_on\_enter\_pressed)

# send button

send\_button = Button(

bottom\_label,

text="Send",

font=FONT\_BOLD,

width=20,

bg=BG\_GRAY,

command=lambda: self.\_on\_enter\_pressed(None),

)

send\_button.place(relx=0.77, rely=0.008, relheight=0.06, relwidth=0.22)

def \_on\_enter\_pressed(self, event):

msg = self.msg\_entry.get()

self.\_insert\_message(msg, "You")

def \_insert\_message(self, msg, sender):

if not msg:

return

self.msg\_entry.delete(0, END)

msg1 = f"{sender}: {msg}\n\n"

self.text\_widget.configure(state=NORMAL)

self.text\_widget.insert(END, msg1)

self.text\_widget.configure(state=DISABLED)

msg2 = f"{bot\_name}: {get\_response(msg)}\n\n"

self.text\_widget.configure(state=NORMAL)

self.text\_widget.insert(END, msg2)

self.text\_widget.configure(state=DISABLED)

self.text\_widget.see(END)

if \_\_name\_\_ == "\_\_main\_\_":

app = ChatApplication()

app.run()