|  |  |
| --- | --- |
| **К Г ЭУ** | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  «КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ» |

Отчет по лабораторной работе №10

по дисциплине «Проектный практикум по управлению разработкой и разработке программного обеспечения»

**«Сокращение полного перебора»**

**Выполнил:** Конюхов А.И.

**Проверил:**

доцент кафедры ИТИС, к.ф.-м.н. Хуснутдинов Р. М.

Казань, 2024г.

**Техническое Задание**

**1. Общая информация**

* **Название проекта:** Программа для генерации паролей на основе нейронной сети.
* **Цель проекта:** разработать программу, которая использует методы машинного обучения для генерации паролей, основываясь на обучении на наборе данных и проверяет сгенерированные пароли с заданным целевым паролем.
* **Целевая аудитория:** Разработчики, исследователи в области машинного обучения, а также специалисты по информационной безопасности, заинтересованные в тестировании систем безопасности.

**2. Функциональные требования**

* **2.1 Генерация паролей:**
  + **2.1.1. Настройка параметров:** Пользователь должен иметь возможность настроить следующие параметры:
    - Длина пароля (password\_length).
    - Количество обучающих паролей (num\_passwords).
    - Максимальное количество попыток для поиска пароля (max\_attempts).
    - Целевой пароль (target\_password).
    - Температуру (temperature) для генерации.
    - Набор используемых символов (chars).
  + **2.1.2. Генерация паролей:** Программа должна генерировать пароли с помощью рекуррентной нейронной сети (LSTM) на основе заданного набора символов.
  + **2.1.3. Разнообразие генерации:** Программа должна генерировать разнообразные пароли, используя случайный выбор символов на основе вероятностей, предсказываемых нейросетью и с помощью параметра температуры.
* **2.2. Обучение нейронной сети:**
  + **2.2.1. Сбор обучающих данных:** Программа должна генерировать обучающие данные (пароли) для обучения нейросети.
  + **2.2.2. Предобработка данных:** Программа должна преобразовывать символы в числовые последовательности и дополнять последовательности до максимальной длины для обучения нейросети.
  + **2.2.3. Обучение нейросети:** Программа должна обучать рекуррентную нейросеть (LSTM) на подготовленных данных.
    - Использовать функцию потерь sparse\_categorical\_crossentropy и оптимизатор Adam.
    - Использовать EarlyStopping, чтобы предотвратить переобучение.
* **2.3. Проверка паролей:**
  + **2.3.1. Сравнение с целевым паролем:** Программа должна сравнивать сгенерированные пароли с заданным целевым паролем.
  + **2.3.2. Вывод результата:** Программа должна выводить сообщение об успехе и количестве попыток, если пароль найден, или сообщение о неудаче, если пароль не найден после заданного количества попыток.

**3. Технические требования**

* **Язык программирования:** Python 3.6 или выше.
* **Библиотеки:**
  + TensorFlow/Keras для работы с нейронными сетями.
  + NumPy для работы с массивами.
  + scikit-learn для разбиения данных на тренировочные и валидационные выборки.
* **Платформа:** Программа должна работать на любой операционной системе (Windows, macOS, Linux) с установленным Python и необходимыми библиотеками.

**4. Требования к интерфейсу пользователя**

* **Интерфейс:** Программа должна быть консольным приложением, не требующим графического интерфейса пользователя.
* **Ввод параметров:** Пользователь должен иметь возможность задавать параметры программы через переменные в коде (например, password\_length, num\_passwords, target\_password и др.).
* **Вывод информации:** Программа должна выводить:
  + Сгенерированные пароли (если целевой пароль не найден).
  + Сообщение об успехе и количестве попыток, если пароль найден.
  + Сообщение о неудаче, если пароль не найден после заданного количества попыток.

**5. Требования к нефункциональности**

* **Производительность:** Программа должна работать достаточно быстро для обработки разумных объемов данных и генерации паролей.
* **Читаемость кода:** Код должен быть написан понятно и структурировано, с использованием комментариев для пояснения логики.
* **Устойчивость к ошибкам:** Программа должна корректно обрабатывать ошибки (например, некорректные параметры, сбои во время обучения).
* **Модифицируемость:** Код должен быть легко модифицируемым для изменения параметров, добавления новых функций, или тестирования других архитектур.

**Справочная документация:**

**Использование программы:**

* 2.1. Запуск программы: Программа запускается из командной строки или терминала с помощью интерпретатора Python (python your\_program\_name.py).
* 2.2. Настройка параметров: Основные параметры программы настраиваются в начале if \_\_name\_\_ == "\_\_main\_\_": блока кода, перед вызовом функций.
  + password\_length:
    - Определение: Длина пароля, который будет сгенерирован и проверен.
    - Значение по умолчанию: 8.
    - Тип: Целое число (integer).
    - Пример использования: password\_length = 12 (установит длину пароля в 12 символов).
  + num\_passwords:
    - Определение: Количество паролей для обучения модели.
    - Значение по умолчанию: 10000.
    - Тип: Целое число (integer).
    - Пример использования: num\_passwords = 20000 (увеличит количество обучающих паролей).
  + target\_password:
    - Определение: Целевой пароль, который программа будет пытаться сгенерировать.
    - Значение по умолчанию: "password".
    - Тип: Строка (string).
    - Пример использования: target\_password = "MySecret123" (установит целевой пароль в MySecret123).
  + max\_attempts:
    - Определение: Максимальное количество попыток генерации, которые будет предпринимать программа, прежде чем остановится в поиске целевого пароля.
    - Значение по умолчанию: 100
    - Тип: Целое число (integer).
    - Пример использования: max\_attempts = 200 (увеличит количество попыток до 200).
  + chars:
    - Определение: Набор символов, которые будут использованы при генерации паролей.
    - Значение по умолчанию: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&\*".
    - Тип: Строка (string).
    - Пример использования: chars = "abc123" (установит символы только abc123).
  + temperature:
    - Определение: Параметр, который влияет на случайность выбора символов при генерации пароля. Меньшее значение (например, 0.5) делает выбор более предсказуемым, большее значение (например, 1.5) более случайным.
    - Значение по умолчанию: 0.7
    - Тип: Вещественное число (float)
    - Пример использования: temperature = 1.2 (увеличивает температуру, делает генерацию более случайной).
* 2.3. Вывод программы:
  + Успешный поиск пароля: Если целевой пароль найден, программа выведет сообщение следующего вида:

«Пароль найден за N попыток»

Неудачный поиск выведет сообщение:

«Целевой пароль не был найден за 100 попыток»

Алгоритм работы программы:

1. Программа генерирует случайные пароли и добавляет несколько популярных паролей
2. Пароли преобразуются в числовые последовательности и подготавливаются для обучения нейросетью
3. LSTM-модель обучается на подготовленных данных
4. Программа генерирует пароли с использованием обученной нейросети, выбирая символы на основе предсказаний модели с использованием температуры.
5. Сгенерированные пароли сравниваются с целевым паролем.
6. Выводится сообщение о том, найден пароль или нет, и, если найден, выводится количество попыток.

Код программы:

import numpy as np  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import LSTM, Dense, Embedding, Input  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import EarlyStopping  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import OneHotEncoder  
import random  
  
# 1. Генерация данных  
def generate\_passwords(num\_passwords, password\_length):  
 chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&\*"  
 passwords = []  
 for \_ in range(num\_passwords):  
 password = ''.join(random.choice(chars) for \_ in range(password\_length))  
 passwords.append(password)  
  
 # Добавляем часто встречающиеся пароли  
 common\_passwords = ["password123", "12345678", "qwert", "Abcde123!", "Date12345"]  
 passwords += common\_passwords  
  
 return passwords  
  
# 2. Преобразование данных  
def prepare\_data(passwords, chars):  
 char\_to\_int = dict((c, i) for i, c in enumerate(chars))  
 int\_to\_char = dict((i, c) for i, c in enumerate(chars))  
 num\_chars = len(chars)  
  
 encoded\_passwords = []  
 encoded\_target = []  
 for password in passwords:  
 for i in range(len(password) - 1):  
 encoded\_pass = [char\_to\_int[char] for char in password[:i+1]]  
 encoded\_passwords.append(encoded\_pass)  
 encoded\_target.append(char\_to\_int[password[i+1]])  
  
  
 max\_len = max([len(x) for x in encoded\_passwords])  
  
 padded\_inputs = tf.keras.preprocessing.sequence.pad\_sequences(encoded\_passwords, maxlen=max\_len, padding='pre')  
  
 return padded\_inputs, np.array(encoded\_target), num\_chars, int\_to\_char, max\_len, char\_to\_int  
  
# 3. Построение нейросети  
def create\_model(num\_chars, max\_len):  
 model = Sequential()  
 model.add(Embedding(num\_chars, 32, input\_length = max\_len))  
 model.add(LSTM(128, return\_sequences=False))  
 model.add(Dense(num\_chars, activation='softmax')) # Выходной слой с softmax  
 optimizer = Adam(learning\_rate=0.001)  
 model.compile(loss='sparse\_categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])  
 return model  
  
# 4. Обучение нейросети  
def train\_model(model, X\_train, y\_train, X\_val, y\_val):  
 early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)  
 model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_val, y\_val), callbacks=[early\_stopping])  
 return model  
  
# 5. Генерация паролей  
def generate\_password\_with\_lstm(model, chars, int\_to\_char, password\_length, max\_len, char\_to\_int, temperature = 1.0):  
 generated\_password = ""  
 for \_ in range(password\_length):  
 encoded\_pass = [char\_to\_int[char] for char in generated\_password]  
 encoded\_pass = tf.keras.preprocessing.sequence.pad\_sequences([encoded\_pass], maxlen=max\_len, padding='pre')  
  
 pred = model.predict(encoded\_pass)  
 pred = np.asarray(pred).astype('float64')  
  
 # Apply temperature  
 pred = np.log(pred) / temperature  
 exp\_preds = np.exp(pred)  
 pred = exp\_preds / np.sum(exp\_preds)  
  
 next\_char\_index = np.random.choice(len(chars), p=pred.flatten())  
 generated\_password += int\_to\_char[next\_char\_index]  
  
 return generated\_password  
  
def check\_password(generated\_password, target\_password):  
 return generated\_password == target\_password  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 password\_length = 8  
 num\_passwords = 10000  
 target\_password = "password" # Целевой пароль  
  
 chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&\*"  
  
 # 1. Генерация данных  
 passwords = generate\_passwords(num\_passwords, password\_length)  
  
 # 2. Преобразование данных  
 X, y, num\_chars, int\_to\_char, max\_len, char\_to\_int = prepare\_data(passwords, chars)  
  
 X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
 # 3. Построение нейросети  
 model = create\_model(num\_chars, max\_len)  
 # 4. Обучение нейросети  
 model = train\_model(model, X\_train, y\_train, X\_val, y\_val)  
  
 # 5. Генерация и проверка паролей  
 max\_attempts = 100  
 for attempt in range(max\_attempts):  
 generated\_password = generate\_password\_with\_lstm(model, chars, int\_to\_char, password\_length, max\_len, char\_to\_int, temperature=0.7)  
 if check\_password(generated\_password, target\_password):  
 print(f"Пароль найден! '{generated\_password}' за {attempt+1} попыток")  
 break  
  
 if attempt == max\_attempts -1:  
 print("Целевой пароль не был найден за 100 попыток")