МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра Вычислительной техники

Машинное обучение.

Лабораторная работа №4

Практика 7-8

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АПИМ-24  Выполнил:  Разуваев В. В. | Проверил:  Гаврилов А.В. |

**Содержание**

[Цель работы 3](#_Toc184590788)

[Задание 3](#_Toc184590789)

[Ход работы 3](#_Toc184590790)

[Задача 1 3](#_Toc184590791)

[Задача 2 8](#_Toc184590792)

[Заключение 13](#_Toc184590793)

# Цель работы

Мониторинг моделей глубокого обучения средствами библиотеки Keras. Ансамблирование моделей нейронных сетей с использованием библиотеки Keras.

# Задание

1. Построить рекуррентную нейронную сеть, которая будет прогнозировать значение некоторого периодического сигнала (Вариант 4):

Преобразовать последовательность в датасет, который можно подавать на вход нейронной сети (можно использовать функцию gen\_data\_from\_sequence из примера). Разбить датасет на обучающую, контрольную и тестовую выборку. Построить и обучить модель. Построить график последовательности, предсказанной на тестовой выборке (пример построения также есть в примере). Данный график необходимо также добавить в pr.

2. Реализовать собственный CallBack, и провести обучение модели из практического занятия №6 с написанным CallBack’ом. То, какой CallBack необходимо реализовать определяется вариантом.

Вариант 4: Сохранение моделей на заданных пользователем эпохах. Название файлов с моделями должна иметь следующий вид <текущая дата>\_<префикс, задаваемый пользователем>\_<номер эпохи>.

# Ход работы

## Задача 1

*Листинг 1 –Код с генерацией последовательности для 4 варианта:*

import numpy as np

import random

import math

import matplotlib.pyplot as plt

def func(i):

i = i % 31

return ((i-15) \*\* 2)/100 - 4

def gen\_sequence(seq\_len = 1000):

seq = [math.cos(i/2) + func(i) + random.normalvariate(0, 0.04) for i in range(seq\_len)]

return np.array(seq)

def draw\_sequence():

seq = gen\_sequence(250)

plt.plot(range(len(seq)),seq)

plt.show()

draw\_sequence()

Структура программы:

Генерация данных:

* Последовательность создается из функции math.cos, функции func и добавленного шума (имитация реального сигнала, содержащего закономерности и случайные отклонения).
* Данные разбиваются на окна длиной lookback для предсказания следующего значения.

Разделение данных:

* Разделение на тренировочные, валидационные и тестовые данные.

Архитектура модели:

* GRU и LSTM слои используют память для учета временной зависимости данных.
* Dropout и Recurrent Dropout добавлены для предотвращения переобучения.

Обучение модели:

* На каждой эпохе модель обновляет веса для минимизации ошибки.

Оценка и визуализация:

* Строится график снижение функции потерь для обучающей и валидационной выборок.
* Строится график реальной и предсказанной последовательности.

*Листинг 2 –Код для задания 1:*

from tensorflow.keras.models import Sequential

from tensorflow.keras import layers

import numpy as np

import random

import math # Для работы с математическими функциями

import matplotlib.pyplot as plt

# Определяем вспомогательную функцию для создания сигнала (функция из 4 варианта)

def func(i):

i = i % 31

return ((i - 15) \*\* 2) / 100 - 4

# Генерация последовательности с заданной длиной

def gen\_sequence(seq\_len=1000):

seq = [math.cos(i / 2) + func(i) + random.normalvariate(0, 0.04) for i in range(seq\_len)]

return np.array(seq) # Возвращаем последовательность в виде массива numpy

# Функция для подготовки данных из последовательности

def gen\_data\_from\_sequence(seq\_len=1006, lookback=10):

"""

- seq\_len: длина последовательности

- lookback: размер окна данных (количество прошлых значений для предсказания следующего)

"""

seq = gen\_sequence(seq\_len) # Генерируем последовательность

# Создаем входные данные (прошлые значения для каждого окна)

past = np.array([[[seq[j]] for j in range(i, i + lookback)] for i in range(len(seq) - lookback)])

# Создаем ожидаемые результаты (следующее значение после окна)

future = np.array([[seq[i]] for i in range(lookback, len(seq))])

return past, future # Возвращаем входы и выходы

# Генерация данных и их разбиение

data, res = gen\_data\_from\_sequence() # Генерируем данные (входы и целевые значения)

dataset\_size = len(data)

train\_size = (dataset\_size // 10) \* 7 # 70% данных - тренировочная выборка

val\_size = (dataset\_size - train\_size) // 2 # 15% - валидационная выборка

# Оставшиеся 15% - тестовая выборка

# Разбиваем данные на тренировочные, валидационные и тестовые

train\_data, train\_res = data[:train\_size], res[:train\_size] # Тренировочная выборка

val\_data, val\_res = data[train\_size:train\_size + val\_size], res[train\_size:train\_size + val\_size] # Валидация

test\_data, test\_res = data[train\_size + val\_size:], res[train\_size + val\_size:] # Тестовая выборка

# Создание нейросети

model = Sequential() # Инициализация последовательной модели

# Добавление первого слоя GRU

model.add(layers.GRU(32, recurrent\_activation='sigmoid', input\_shape=(None, 1), return\_sequences=True))

# Добавление слоя LSTM

model.add(layers.LSTM(32, activation='relu', return\_sequences=True, dropout=0.2))

# Добавление слоя GRU

model.add(layers.GRU(32, recurrent\_dropout=0.2))

# Выходной слой

model.add(layers.Dense(1))

# Компиляция модели

model.compile(optimizer='nadam', loss='mse')

# Обучение модели

history = model.fit(train\_data, train\_res, epochs=20, validation\_data=(val\_data, val\_res))

# Построение графика функции потерь

loss = history.history['loss'] # История ошибки на тренировочных данных

val\_loss = history.history['val\_loss'] # История ошибки на валидации

plt.plot(range(len(loss)), loss, label='Train Loss') # График ошибки на тренировке

plt.plot(range(len(val\_loss)), val\_loss, label='Validation Loss') # График ошибки на валидации

plt.legend()

plt.show()

# Предсказание на тестовой выборке

predicted\_res = model.predict(test\_data) # Предсказания модели

# Построение графика предсказанных значений и реальных значений

pred\_length = range(len(predicted\_res)) # Диапазон значений для оси X

plt.plot(pred\_length, predicted\_res, label='Predicted') # Предсказанные значения

plt.plot(pred\_length, test\_res, label='Actual') # Реальные значения

plt.legend()

plt.show()

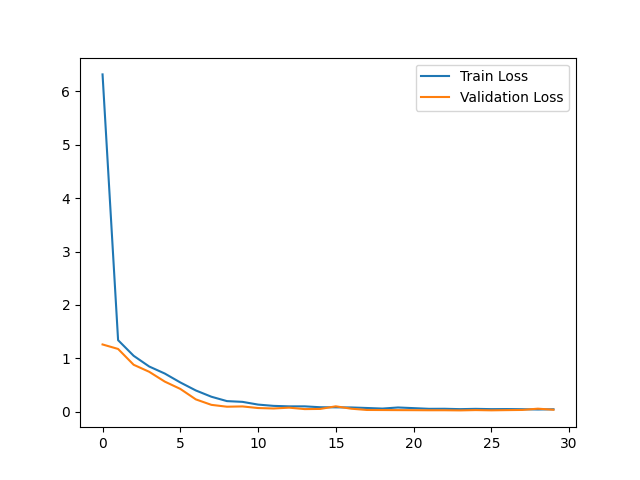


Рисунок 1 – График снижение функции потерь для обучающей и валидационной выборок

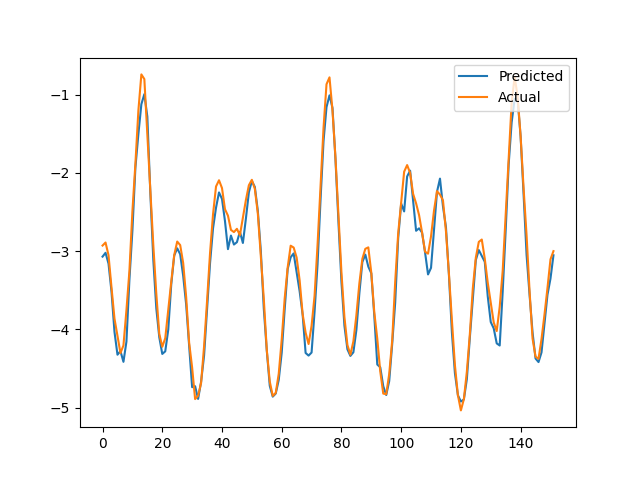


Рисунок 2 – График реальной и предсказанной последовательности

График функции потерь показывает стабильное уменьшение ошибки, отсутствие значительного переобучения. На тестовой выборке предсказанные значения модели достаточно близко повторяют реальный сигнал, что подтверждает способность модели корректно прогнозировать данные. Модель успешно справилась с задачей прогнозирования значений периодического сигнала, несмотря на наличие шума.

## Задача 2

Структура программы:

Генерация данных: Используется функция gen\_data, которая создает изображения и их метки.

Нормализация данных: Нормализация уменьшает влияние масштаба пикселей и улучшает обучение.

One-hot кодировка меток: Преобразование меток в формат, понятный модели.

Архитектура модели: Используются сверточные слои для извлечения признаков, пулинг для уменьшения размерности, и полносвязные слои для классификации.

Сохранение модели на заданных эпохах: запуск кастомного Callback для сохранения модели.

Визуализация: Построение графика точности (Accuracy), показывающего, как изменялась точность модели на тренировочных и валидационных данных в процессе обучения, а также графика функции потерь (Loss), показывающий изменение ошибки модели (функции потерь) на тренировочных и валидационных данных.

Сохранение модели: Сохранение модели в формате .h5 для дальнейшего использования.

*Листинг 3 – Реализация кастомного Callback для сохранения модели (pr8.py).*

import os

import datetime

from tensorflow.keras.callbacks import Callback

class CustomModelSaver(Callback):

def \_\_init\_\_(self, save\_epochs, prefix="model", directory="models"):

"""

Инициализация кастомного CallBack для сохранения моделей.

:param save\_epochs: список эпох, на которых нужно сохранять модель.

:param prefix: префикс для имени файла.

:param directory: папка для сохранения моделей.

"""

super().\_\_init\_\_()

self.save\_epochs = save\_epochs

self.prefix = prefix

self.directory = directory

os.makedirs(self.directory, exist\_ok=True) # Создать папку, если её нет

def on\_epoch\_end(self, epoch, logs=None):

"""

Сохраняет модель на указанных эпохах.

:param epoch: текущая эпоха.

:param logs: данные, собранные за текущую эпоху.

"""

if (epoch + 1) in self.save\_epochs: # Проверяем, входит ли эпоха в список

current\_date = datetime.datetime.now().strftime("%Y-%m-%d")

filename = f"{current\_date}\_{self.prefix}\_epoch-{epoch+1}.h5"

filepath = os.path.join(self.directory, filename)

self.model.save(filepath)

print(f"Модель сохранена: {filepath}")

*Листинг 4 – Код 6го практического задания с добавлением Callback.*

import numpy as np # Для работы с массивами

from tensorflow.keras.utils import to\_categorical # Для преобразования меток в one-hot кодировку

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from sklearn.model\_selection import train\_test\_split # Для разделения данных на тренировочную и тестовую выборки

from var4 import gen\_data # Импорт функции для генерации данных

import pr8

data, labels = gen\_data(size=1000, img\_size=50) # Генерация 1000 изображений размером 50x50 пикселей

# Преобразование строковых меток в числовые

class\_map = {'Cross': 0, 'Line': 1} # Определяем соответствие между строковыми и числовыми метками

numeric\_labels = np.vectorize(class\_map.get)(labels.flatten()) # Преобразуем метки в числовой формат

# Нормализация данных

data = data / 255.0 # Приводим значения пикселей к диапазону [0, 1]

data = data[..., np.newaxis] # Добавляем измерение для канала (1 для черно-белых изображений)

# Проверяем распределение классов

unique, counts = np.unique(numeric\_labels, return\_counts=True)

print(f"Распределение классов: {dict(zip(unique, counts))}")

# Разделение данных на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data, numeric\_labels, test\_size=0.2, random\_state=42

)

# Преобразование меток в формат one-hot encoding

y\_train = to\_categorical(y\_train, num\_classes=2)

y\_test = to\_categorical(y\_test, num\_classes=2)

# Импорт необходимых классов для построения модели

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

# Создание модели сверточной нейронной сети

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(50, 50, 1)), # Сверточный слой с 32 фильтрами

MaxPooling2D((2, 2)), # Слой подвыборки (пулинга) для уменьшения размерности

Dropout(0.25), # Слой регуляризации для предотвращения переобучения

Conv2D(64, (3, 3), activation='relu'), # Еще один сверточный слой с 64 фильтрами

MaxPooling2D((2, 2)), # Еще один слой подвыборки

Dropout(0.25), # Регуляризация

Conv2D(128, (3, 3), activation='relu'), # Углубляем сеть (128 фильтров)

MaxPooling2D((2, 2)), # Подвыборка

Dropout(0.25),

Flatten(), # Преобразуем данные в одномерный массив для Dense-слоев

Dense(256, activation='relu'), # Полносвязный слой с 256 нейронами

Dropout(0.5), # Регуляризация

Dense(2, activation='softmax') # Выходной слой с 2 классами

])

# Компиляция модели

model.compile(

optimizer=Adam(learning\_rate=0.0001), # Используем оптимизатор Adam с пониженной скоростью обучения

loss='categorical\_crossentropy', # Функция потерь для задачи классификации

metrics=['accuracy'] # Метрика для отслеживания точности

)

# # Использование кастомного CallBack

save\_epochs = [5, 25, 45] # Эпохи, на которых нужно сохранять модели

custom\_callback = pr8.CustomModelSaver(save\_epochs, prefix="cross\_line\_model", directory="saved\_models")

# Обучение модели

history = model.fit(

X\_train, y\_train, # Обучающие данные

validation\_split=0.2, # Используем 20% обучающих данных для проверки

epochs=50, # Количество эпох

batch\_size=32, # Размер батча

verbose=1, # Показ прогресса обучения

callbacks=[custom\_callback]

)

# Оценка модели на тестовой выборке

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f"Точность на тестовой выборке: {test\_accuracy \* 100:.2f}%")

Сохранение модели осуществляется на 5, 25 и 45 эпохе. Соответствующие точности 52%, 85%, 100%.

*Листинг 5 – Код для визуализации степени обучения моделей.*

# Функция для построения графиков истории обучения

def plot\_training\_history(history):

loss = history.history['loss']

val\_loss = history.history['val\_loss']

accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

epochs = range(1, len(loss) + 1)

plt.figure(figsize=(12, 5))

# График потерь

plt.subplot(1, 2, 1)

plt.plot(epochs, loss, label='Training Loss', color='blue')

plt.plot(epochs, val\_loss, label='Validation Loss', color='orange')

plt.title('Loss During Training')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

# График точности

plt.subplot(1, 2, 2)

plt.plot(epochs, accuracy, label='Training Accuracy', color='blue')

plt.plot(epochs, val\_accuracy, label='Validation Accuracy', color='orange')

plt.title('Accuracy During Training')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

# Построение графиков обучения

plot\_training\_history(history)

# Функция для оценки сохранённых моделей

def evaluate\_saved\_models(directory, X\_test, y\_test):

models = sorted(os.listdir(directory))

epochs = []

accuracies = []

for model\_file in models:

filepath = os.path.join(directory, model\_file)

model = load\_model(filepath)

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

epochs.append(int(model\_file.split('\_')[-1].split('-')[-1].split('.')[0]))

accuracies.append(accuracy)

print(f"Модель {model\_file} -> Точность: {accuracy \* 100:.2f}%")

epochs.sort()

accuracies.sort()

plt.figure(figsize=(8, 6))

plt.plot(epochs, accuracies, marker='o', linestyle='-', color='blue')

plt.title('Test Accuracy of Saved Models')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.grid(True)

plt.show()

# Оценка сохранённых моделей

evaluate\_saved\_models("saved\_models", X\_test, y\_test)

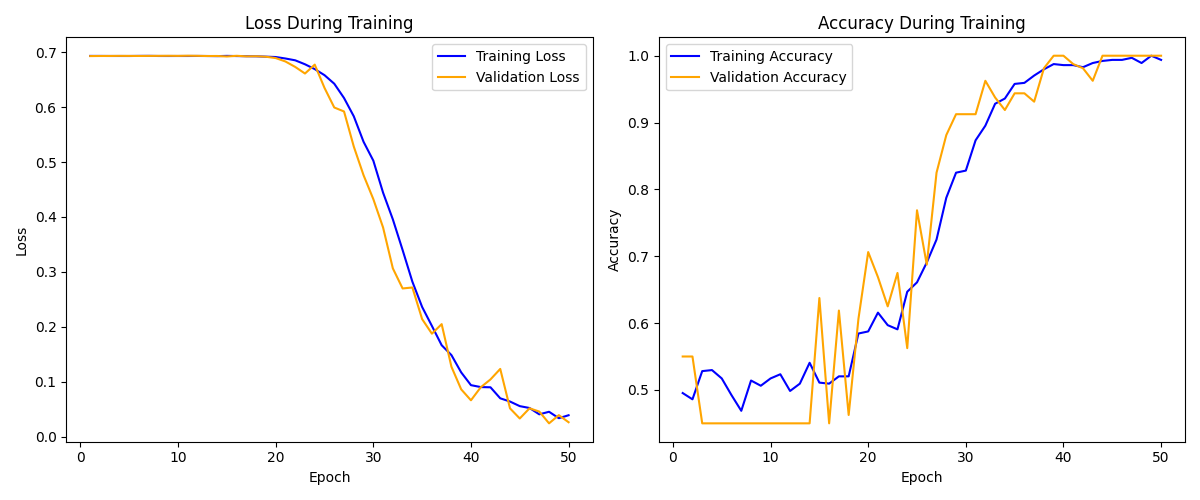


Рисунок 3 – Графики точности и функции потерь

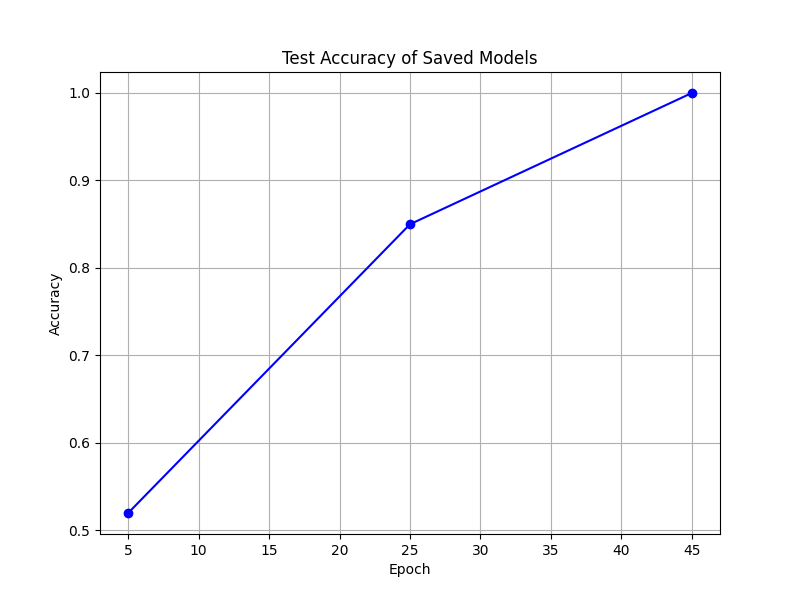


Рисунок 4 – Точности сохраненных моделей

# Заключение

В ходе выполнения лабораторной работы была реализована рекуррентная нейронная сеть на базе библиотеки Keras для прогнозирования значений заданной функции. Также было произведено обучение модели с реализованным собственным CallBack. Точность модели стабильно растёт с увеличением эпох, это хороший признак, но есть момент, в котором точность достигает 100% (как на эпохе 45 в этом случае), это может быть связано с переобучением или несбалансированным набором данных.