Министерство науки и высшего образования РФ

Томский государственный университет
систем управления и радиоэлектроники

Е.Ю. Костюченко, А.Ю. Якимук

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Учебно-методическое пособие
для студентов направлений подготовки
10.00.00 Информационная безопасность

Томск
2022

УДК 004.056
ББК 32.973.26-018.2
К 64

Костюченко, Евгений Юрьевич
К 64 Искусственный интеллект: учебно-методическое пособие / Е.Ю. Костюченко, А.Ю. Якимук. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2022. – 30 с.

Настоящее учебно-методическое пособие содержит описания лабораторных и самостоятельных работ по дисциплине «Искусственный интеллект» для направлений подготовки, входящих в укрупненную группу специальностей и направлений 10.00.00 Информационная безопасность.

УДК 004.056
ББК 32.973.26-018.2

ё

СОДЕРЖАНИЕ

ё

ВВЕДЕНИЕ

Целью преподавания дисциплины является изучение принципов работы нейронных сетей и примеров применения их на практике.

Задачи изучения дисциплины:

– Изучить основные принципы машинного обучения и нейронных сетей.

– Рассмотреть принципы применения машинного обучения и нейронных сетей для задач компьютерного зрения.

– Рассмотреть принципы применения машинного обучения и нейронных сетей для задач обработки естественного языка.

ё

ЛАБОРАТОРНАЯ РАБОТА №1
Знакомство нейронными сетями

Цель работы

Целью данной лабораторной работы является получение навыков построения нейронной сети, подбор параметров и оценки качества работы на обучающей выборке.

Ход работы

Для создания нейронной сети и дальнейшей ее работы необходимо подключить следующие библиотеки (рис. 1.1):

— numpy – поддержка больших многомерных массивов и матриц, векторизация, большая библиотека высокоуровневых математических функций;

— sklearn – библиотека машинного обучения.

— pandas – библиотека препроцессинга, анализа и иной обработки данных;

— matplotlib – построение графиков;

— imblearn – библиотека для борьбы с проблемами несбалансированных наборов данных;

— random;

— tensorflow – библиотека для решения задач построения и тренировки нейронной сети.

Второй шаг в работе – подключение, извлечение данных из файла, в котором они представлены, и разбиение на параметры и классы. Поскольку именования классов представлены в строковом типе, необходимо преобразовать их таким образом, чтобы они имели числовой формат, но не теряли логического смысла (рис. 1.2).

После подключения входных данных необходимо произвести их нормализацию, а также разбиение на обучающую и тестовые выборки для этого следует применить следующие команды:

— X = MinMaxScaler().fit_transform(X.to_numpy());

— X_train,X_test,Y_train,Y_test = train_test_split(X,Y).

ё

```
import numpy as np
import tensorflow as tf
import random as rn
import pandas as pd
from tensorflow.keras.utils import to_categorical
from keras import backend as K
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from imblearn.pipeline import Pipeline
from keras.wrappers.scikit_learn import KerasClassifier
```

Рисунок 1.1 – Подключение библиотек

```
df = pd.read_csv('abalone.dat')
print(df.columns)
X = df.drop([' Rings'], axis = 1)
temp = to_categorical(df['Sex'].astype('category').cat.codes)
#temp = pd.DataFrame(temp)
X = X.drop(['Sex'], axis = 1)
X[["SF","SI","SM"]]=temp
Y = to_categorical(df[' Rings'].astype('category').cat.codes)
print(X)
```

Рисунок 1.2 – Подключение входных данных

По завершении подготовительных работ можно приступать к непосредственному созданию нейронной сети. Для этого необходимо задать количество слоев и количество нейронов в каждом слое (рис. 1.3).

Дальнейшая работа заключается в обучении нейронной сети. Для этого необходимо подобрать наиболее оптимальные значения для следующих параметров (рис. 1.4):

— batchsize – размер одной части обрабатываемых данных;

— epochs – количество циклов обучения.

По окончании работы программы должен быть получен график зависимости значения точности (acc) и потерь (loss) от эпохи (рис. 1.5)

ё

```
#Создание нейронной сети
def my_model(my_X, my_Y):
  model = Sequential()#Сеть с последовательным послойным распространением сигнала
  model.add(Dense(50,activation='tanh',input_shape=(my_X.shape[1],)))#Добавили полносвязный слой. 10 нейронов
  #активация - гиперболический тангенс tanh, число входов - 144, по числу параметров
  #Возможные функции активации
  #https://keras.io/api/layers/activations/
  model.add(Dense(40,activation='relu'))
  #model.add(Dense(30,activation='tanh'))
  #model.add(Dense(3,activation='tanh'))
  model.add(Dense(my_Y.shape[1],activation='softmax'))#Добавили полносвязный слой. Число нейронов - число выходов (пользователей)
  ##функция активации - софтмакс, для нормировки
  model.compile(
    #Возможные loss-функции
    #https://keras.io/api/losses/probabilistic_losses/#binarycrossentropy-class
    loss = tf.keras.losses.CategoricalCrossentropy(),
    #loss='mse',#потери - среднеквадратичная ошибка
    optimizer=tf.optimizers.Adam(learning_rate=1e-2),#оптимизатор adam
    #optimizer = 'adam',
    metrics=['acc']#метрики - доля верных ответов, F1-мера, точность, полнота
  )
  return(model)
```

Рисунок 1.3 – Создание нейронной сети

```
def train_model(model, feature, label, epochs, batch_size):
  """Обучение модели"""

  history = model.fit(x=feature,
                      y=label,
                      batch_size=batch_size,
                      epochs=epochs)

  trained_weight = model.get_weights()[0]
  trained_bias = model.get_weights()[1]

  epochs = history.epoch

  hist = pd.DataFrame(history.history)

  acc = hist["accuracy"]

  return trained_weight, trained_bias, epochs, acc

print("Defined create_model and train_model")
```

```
Defined create_model and train_model
```

```
my_features = data_scaled
my_labels = Y.astype('category').cat.codes
#my_labels = pd.get_dummies(Y)
print(my_labels)
```

```
learning_rate=0.05
epochs=100
my_batch_size= 19020

my_model = build_model(learning_rate)
trained_weight, trained_bias, epochs, acc = train_model(my_model, my_features,
                                                        my_labels, epochs,
                                                        my_batch_size)
plot_the_loss_curve(epochs, acc)
```

Рисунок 1.4 – Обучение нейронной сети

ё

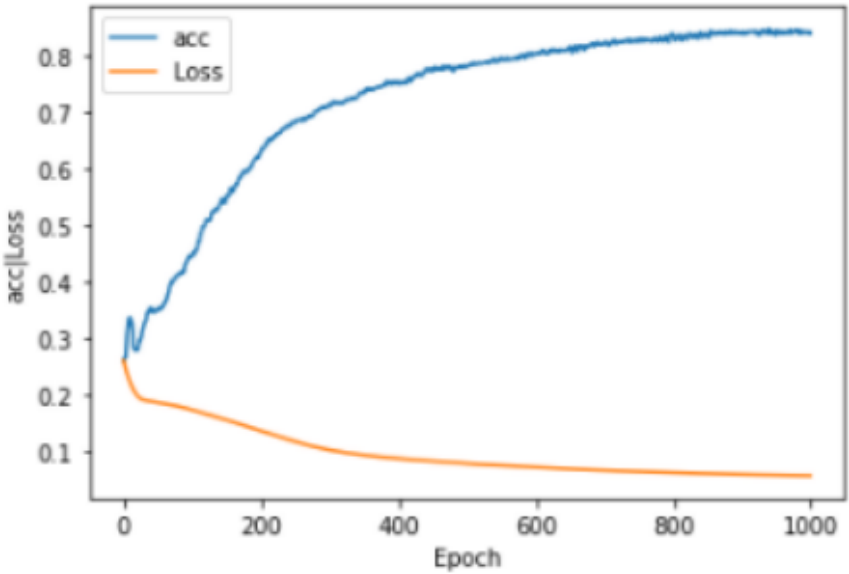


Рисунок 1.5 – Пример графика обучения

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Описание нейронной сети.
4. Подбор параметров обучения.
5. Заключение.

ё

ЛАБОРАТОРНАЯ РАБОТА №2
Архитектура, кроссвалидация

Цель работы

Целью данной лабораторной работы является получение навыков работы с метриками качества нейронных сетей на языке программирования Python.

Ход работы

Для выполнения лабораторной работы применяется готовая программа на языке программирования Python, реализующая оценивание качества обучения нейронной сети.

Для создания нейронной сети подключаются необходимые для ее работы библиотеки (рис. 2.1) и реализуется открытие заданного набора данных в переменную (рис. 2.2).

```
import numpy as np
import tensorflow as tf
import random as rn
import pandas as pd
from tensorflow.keras.utils import to_categorical
from keras import backend as K
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from imblearn.pipeline import Pipeline
from keras.wrappers.scikit_learn import KerasClassifier
```


ё

Рисунок 2.1 – Подключение библиотек

```
#Открыть набор данных в переменную
df = pd.read_csv('heart.dat')
print(df.columns)
X = df.drop([' Thal Class'], axis = 1)
Y = to_categorical(df[' Thal Class'].astype('category').cat.codes)
```

Рисунок 2.2 – Открытие файла с набором данных

Далее необходимо провести нормализацию данных – процедуру предобработки входной информации (обучающих, тестовых и валидационных выборок, а также реальных данных) (рис. 2.3).

```
X = MinMaxScaler().fit_transform( X.to_numpy())
```

Рисунок 2.3 – Нормализация входных данных

Далее реализуются функции, определяющие метрики качества работы системы - полнота, точность и F-мера (рис. 2.4).

ё

```
def recall_m(y_true, y_pred):
    print(type(y_true))
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):#Точность
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):#F1-мера
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Рисунок 2.4 – Задание функций

По завершении описанных действий происходит создание нейронной сети (рис. 2.5).

```
#Создание нейронной сети
def my_model(my_X, my_Y, activation_1='tanh', activation_2='relu',
             neurons_input = 10, neurons_hidden_1=10, loss = 'mse',
             optimizer='adam'):
  model = Sequential()#Сеть с последовательным послойным распространением сигнала
  model.add(Dense(neurons_input,activation=activation_1,input_shape=(my_X.shape[1],)))
  #активация - гиперболический тангенс tanh, число входов - 144, по числу параметров
  #Возможные функции активации
  #https://keras.io/api/layers/activations/
  model.add(Dense(neurons_hidden_1,activation=activation_2))
  model.add(Dense(my_Y.shape[1],activation='sigmoid'))
  #Функция активации - софтмакс, для нормировки
  model.compile(
    #Возможные loss-функции
    #https://keras.io/api/losses/probabilistic_losses/#binarycrossentropy-class
    #loss = tf.keras.losses.BinaryCrossentropy(),
    loss=loss,#потери - среднеквадратичная ошибка
    #optimizer=tf.optimizers.Adam(learning_rate=1e-3),#оптимизатор adam
    optimizer = optimizer,
    metrics=['acc',f1_m,precision_m, recall_m]#метрики-доля верных ответов, F1-мера, точность,
  )
  return(model)
```

Рисунок 2.5 – Создание нейронной сети

ё

Следующим шагом происходит обучение нейронной сети, а также анализируется качества ее работы с помощью метрик на каждом этапе кроссвалидации. Результаты выводятся, строится график (рис. 2.6, 2.7, 2.8).

```
# Define the K-fold Cross Validator
num_folds = 3
batch_size = 100
acc_per_fold = []
loss_per_fold = []
f1_per_fold = []
prec_per_fold = []
recall_per_fold = []
res_f1_per_fold = []
res_loss_per_fold = []

kfold = KFold(n_splits=num_folds, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 1
for train, test in kfold.split(X, Y):
  model = my_model( X, Y)
  # Compile the model
  # Generate a print
  print('------------------------------------------------------------------------')
  print(f'Training for fold {fold_no} ...')
```

Рисунок 2.6 – Работа с нейронной сетью (часть 1)

ё

```
results = model.fit(X[train],Y[train],
                    batch_size=batch_size,
                    validation_data = (X[test],Y[test]),
                    epochs=50,verbose=1)
# Generate generalization metrics
scores = model.evaluate(X[test], Y[test], verbose=0)
print(scores)
print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]};
{model.metrics_names[1]} of {scores[1]*100}%; {model.metrics_names[2]} of {scores[2]*100}%;
{model.metrics_names[3]} of {scores[3]*100}%; {model.metrics_names[4]} of {scores[4]*100}%;')

acc_per_fold.append(scores[1] * 100)
f1_per_fold.append(scores[2] * 100)
prec_per_fold.append(scores[3] * 100)
recall_per_fold.append(scores[4] * 100)
loss_per_fold.append(scores[0])
res_f1_per_fold.append(results.history['val_f1_m'])
res_loss_per_fold.append(results.history['loss'])

# Increase fold number
fold_no = fold_no + 1
```

Рисунок 2.7 – Работа с нейронной сетью (часть 2)

```
# == Provide average scores ==
print('------------------------------------------------------------------------')
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('------------------------------------------------------------------------')
  print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}% - F1: {f1_per_fold[i]}% - Precision: {prec_per_fold[i]}% - Recall: {recall_per_fold[i]}%')
print('------------------------------------------------------------------------')
print('Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)} (+- {np.std(loss_per_fold)})')
print(f'> F1: {np.mean(f1_per_fold)} (+- {np.std(f1_per_fold)})')
print(f'> Precision: {np.mean(prec_per_fold)} (+- {np.std(prec_per_fold)})')
print(f'> Recall: {np.mean(recall_per_fold)} (+- {np.std(recall_per_fold)})')
print('------------------------------------------------------------------------')
```

Рисунок 2.8 – Организация вывода результатов

Результат обучения нейронной сети также должен быть представлен графиком (рис. 2.9).

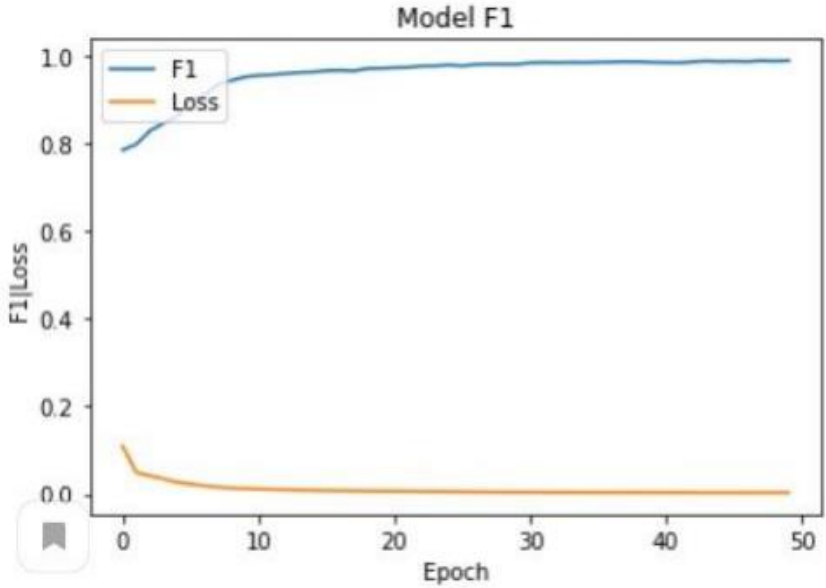
ё

```
#Находим среднюю F-меру
f1_line = [sum(x) / len(x) for x in zip(*res_f1_per_fold)]
#Средняя лосс-функция
loss_line = [sum(x) / len(x) for x in zip(*res_loss_per_fold)]

#Вывод графиков
plt.plot(f1_line)#выводим F1-меру для обучающей выборки
plt.plot(loss_line)#выводим F1-меру для тестовой выборки
plt.title('Model F1')
plt.ylabel('F1|Loss')
plt.xlabel('Epoch')
plt.legend(['F1', 'Loss'], loc='upper left')
plt.show()
```

Рисунок 2.9 – Вывод графика

Дальнейшая работа заключается в подборе оптимальных параметров нейронной сети, и получении результатов ее обучения (рис. 2.10, 2.11).



ё

Рисунок 2.10 – Пример графика

```
Average scores for all folds:
> Accuracy: 99.0120530128479 (+- 0.49926374332220835)
> Loss: 0.0029358278261497617 (+- 0.0009206113442892318)
> F1: 99.00430043538411 (+- 0.4799972317671846)
> Precision: 98.9922006924947 (+- 0.46032841652521855)
> Recall: 99.02203877766927 (+- 0.49583679284937826)
------------------------------------------------------------------------
```

Рисунок 2.11 – Пример оценок нейронной сети

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Описание нейронной сети.
4. Подбор параметров обучения.
5. Заключение.
6. Приложение А (листинг программы).

ё

ЛАБОРАТОРНАЯ РАБОТА №3
Введение в сверточные нейронные сети

Цель работы

Целью данной лабораторной работы является овладение основными навыками для создания сверточных нейронных сетей, обрабатывающих изображения, и работы с ними на языке программирования Python.

Ход работы

Для выполнения лабораторной работы применяется готовая программа на языке программирования Python, реализующая обработку изображений.

Перед началом работы с нейронными сетями к среде исполнения подключаются все необходимые библиотеки и загружается индивидуальный файл с данными (рис. 3.1).

```
import tensorflow as tf
import numpy as np
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Рисунок 3.1 – Подключение библиотек

После этого в программе загружаются списки с тренировочными и тестовыми изображениями и списками, соответствующими их классам. Далее в списке variant загружаются номера классов для индивидуального варианта и производится выборка этих классов изображений среди загруженных (рис. 3.2).

ё

```
# Входные данные:
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
variant = [16, 35, 66, 14, 98, 89, 31, 55, 57, 80, 67, 92, 54, 76, 7, 69, 85, 75, 2, 27]
new_train_images = []
new_train_labels = []
for i in range(train_labels.size):
  if train_labels[i] in variant:
    new_train_images.append(train_images[i])
    i, = np.where(variant == train_labels[i])
    new_train_labels.append(i)
new_train_images = np.asarray(new_train_images)
new_train_labels = np.asarray(new_train_labels)
new_test_images = []
new_test_labels = []
for i in range(test_labels.size):
  if test_labels[i] in variant:
    new_test_images.append(test_images[i])
    i, = np.where(variant == test_labels[i])
    new_test_labels.append(i)
new_test_images = np.asarray(new_test_images)
new_test_labels = np.asarray(new_test_labels)
```

Рисунок 3.2 – Загрузка и предобработка массива данных

После реализации загрузки и предобработки, представленной выше, происходит задание параметров картинок из массива train_images и их вывод (рис. 3.3).

Далее производится задание параметров нейронной сети и вывод сводного представления модели системы (рис. 3.4). После создания модели нейронной сети производится ее обучение (рис. 3.5).

Результаты обучения нейронной сети представляются в виде графиков (рис. 3.6).

ё

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

Рисунок 3.3 – Вывод рисунков

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), padding = 'same', activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), padding = 'same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), padding = 'same', activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(120, activation='tanh'))
model.add(layers.Dense(20))
```

Рисунок 3.4 – Создание модели нейронной сети

ё

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

Рисунок 3.5 – Обучение нейронной сети

```
[56] plt.plot(history.history['accuracy'], label='accuracy')
     plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.ylim([0.5, 1])
     plt.legend(loc='lower right')

     test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 1.1475 - accuracy: 0.7144 - 1s/epoch - 4ms/step
```

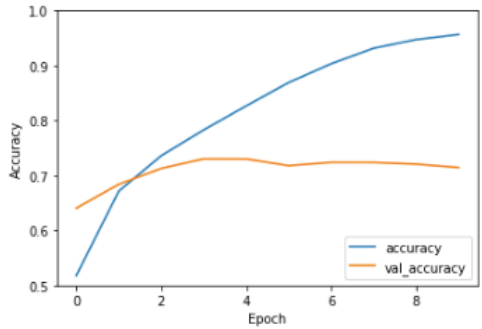


Рисунок 3.6 – Вывод графика

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Описание нейронной сети.
4. Подбор параметров обучения.
5. Заключение.
6. Приложение А (листинг программы).

ё

ЛАБОРАТОРНАЯ РАБОТА №4
Продвинутые сверточные нейронные сети, аугментация

Цель работы

Целью данной лабораторной работы является получение знаний и навыков для реализации аугментации в нейронных сетях, осуществляющих обработку изображений.

Ход работы

Для выполнения лабораторной работы применяется готовая программа на языке программирования Python, реализующая обработку изображений. Программу необходимо доработать путем добавления аугментации.

Перед началом работы с нейронными сетями к среде исполнения подключаются все необходимые библиотеки и загружается индивидуальный файл с данными.

Среди подключаемых библиотек целесообразно выделить:

— "numpy" – многофункциональная математическая библиотека, обеспечивающая работу с большими многомерными массивами и метрицами, векторизацию операций и т.д.;

— "matplotlib" – библиотека для визуализации данных (т.е. построения графиков);

— "sklearn" – библиотека, реализующая функционал машинного обучения.

После этого осуществляется загрузка и предобработка массива данных.

Массив данных делится на две части – тренировочную и валидационную; валидационная выборка составляет 1/5 часть первоначального массива данных, тренировочная – 4/5 соответственно.

Для улучшения качества работы нейронной сети в исходную программу следует внести изменения, среди которых возможны:

— изменение числа классов;

ё

— добавление слоя случайного контраста;
— добавление слоя случайного поворота изображения;
— добавление слоя случайного отражения, а также иных дополнительных слоев (рис. 4.1).

Внесенные в программу изменения должны улучшить работу нейронной сети путем уменьшения значения потерь, повышения точности и снижения вероятности переобучения нейросети (рис. 4.2, 4.3).

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 50, 50, 32)        896

 conv2d_4 (Conv2D)           (None, 50, 50, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 25, 25, 64)       0
 2D)

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 12, 12, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_6 (Conv2D)           (None, 6, 6, 128)         73856

 random_flip (RandomFlip)    (None, 6, 6, 128)         0

 random_rotation (RandomRota  (None, 6, 6, 128)        0
 tion)

 rescaling (Rescaling)       (None, 6, 6, 128)         0

 flatten_1 (Flatten)         (None, 4608)              0

 dense_2 (Dense)             (None, 512)               2359808

 dense_3 (Dense)             (None, 256)               131328

 dense_4 (Dense)             (None, 130)               33410

=================================================================
Total params: 2,654,722
Trainable params: 2,654,722
Non-trainable params: 0
```

Рисунок 4.1 – Пример описания архитектуры нейронной сети

ё

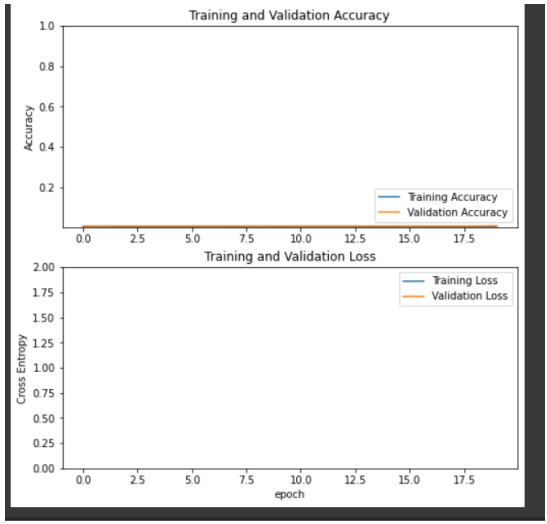
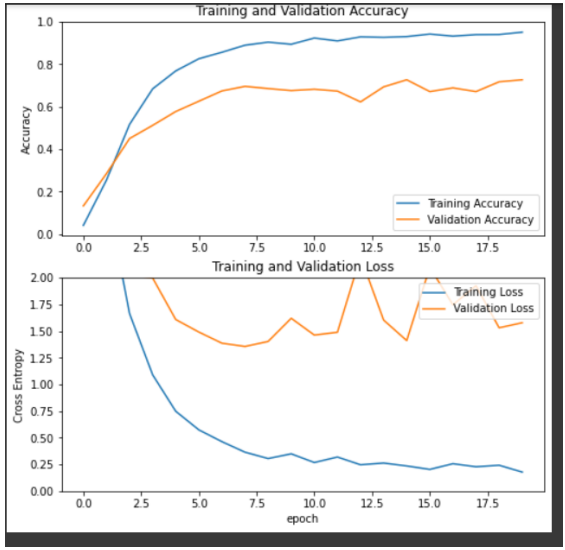


Рисунок 4.2 – Пример результата работы исходной программы



ё

Рисунок 4.3 – Пример результата работы модифицированной программы

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Описание нейронной сети.
4. Подбор параметров обучения.
5. Заключение.
6. Приложение А (листинг программы).

ё

ЛАБОРАТОРНАЯ РАБОТА №5
Введение в обработку текста

Цель работы

Целью данной лабораторной работы является получение знаний и навыков для реализации обработки текста на языке программирования Python.

Ход работы

Для выполнения лабораторной работы применяется готовая программа на языке программирования Python, реализующая обработку текста. Также в программе находятся незаконченные части, завершение которых является заданием на данную лабораторную работу.

В процессе выполнения лабораторной работы необходимо выполнить следующие задания:

1. Найти все номера телефонов в тексте, содержащемся в файле "task1.txt", обращая внимания на всевозможные форматы написания телефонных номеров в его рамках.
2. Посчитать количество предложений, токенов и типов в тексте, содержащемся в файле "task2.txt", сохранив список токенов в массив (прим.: список Python) "tokens".
3. Посчитать количество слов в тексте, содержащемся в файле "task2.txt", встречающихся более одного раза, и слов, состоящих из пяти и более букв.
4. Найти все женские имена в списке персонажей романа Л. Н. Толстого "Война и мир", имеющем место быть в виде текстового файла "task3.txt".

Содержание отчета

1. Титульный лист.
2. Цель работы.

ё

3. Описание нейронной сети.
4. Выполнение задания №1.
5. Выполнение задания №2.
6. Выполнение задания №3.
7. Выполнение задания №4.
8. Заключение.
9. Приложение А (листинг программы).

ё

ЛАБОРАТОРНАЯ РАБОТА №6
Обработка текста с применением нейронных сетей

Цель работы

Целью данной лабораторной работы является получение знаний и навыков для реализации обработки текста с помощью нейронных сетей.

Ход работы

Для выполнения лабораторной работы применяются готовые программы на языке программирования Python, реализующая обработку текста посредством нейронных сетей.

Для выполнения первого задания необходимо экспериментальным путем найти способ для улучшения результата работы нейронной сети, реализованной в первой программе. Для этого могут быть осуществлены следующие действия:

— Изменить способ взвешивания признаков;

— Реализовать взвешивание признаков с помощью точечной взаимной информации (PMI);

— Изменить способ стандартизации данных (см. начиная с 4:25 на шаге 6), например, запоминая сдвиг и масштаб с обучающей выборки и применяя эти параметры для стандартизации тестовой выборки; и/или стандартизируя каждый столбец по отдельности; 4. Добавить регуляризацию;

— Извлекать признаки не через токены, а через N-граммы;

— Добавить стемминг или простую лемматизацию;

— Изменить архитектуру нейросети, например, сделав два слоя вместо одного:

— Проанализировать, как сильно падает качество классификации с уменьшением размера словаря (для фильтрации словаря можно использовать разные эвристики, например, тот же PMI).

ё

В процессе работы необходимо фиксировать все внесенные изменения и их влияние на работу нейронной сети.

Для выполнения второго задания необходимо так же, как и в первом, поэкспериментировать с программой, реализующей нейронную сеть. Это нейросеть выполняет задачу дистрибутивной семантики.

Для достижения результат можно провести следующие действия:

— Поиграться с параметрами - количеством отрицательных слов, размером батча, скоростью обучения, размером окна;

— Убрать разбиение текстов на предложения и увеличить окно;

— Изменить токенизацию, например, разобравшись с библиотекой SpaCy и подключив лемматизацию и POS-теггинг, чтобы строить эмбеддинги не для словоформ, а для лемм (например, chicked_NOUN);

— Реализовать FastText и сравнить, как отличаются списки похожих документов, получаемых с помощью Word2Vec и FastText;

— Усложнить алгоритм оценки вероятности совместной встречаемости слов, например, заменив скалярное произведение на нейросеть с парой слоёв.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Выполнение задания №1.
4. Выполнение задания №2.
5. Выполнение задания №3.
6. Заключение.
7. Приложение А (листинг программы №1).
8. Приложение Б (листинг программы №2).

ё

КУРСОВАЯ РАБОТА

1. Выбор и согласование темы

Тема курсовой работы может быть, как выбрана из рекомендованного списка типовых проектов, так и определена в индивидуальном порядке. Тема курсовой работы определяется обучающимися по согласованию с преподавателем.

После выбора темы определяется детальное задание на курсовую работу

Каждый студент регистрирует в Системе обучения согласованную тему и описание задания на свою курсовую работу в элементе курса «Темы курсовых работ».

2. Выполнение и оформление курсовой работы

В комплект обязательных материалов, который представляются учащимися до начала защиты проекта, входит:

1. Исходный код программы в одном архиве.
2. Презентация проекта, оформленная на основе шаблонов, выложенных в Системе обучения.
3. Пояснительная записка.
4. Ссылка на хранилище с датасетом.

Студент обязан гарантировать, что предоставленные им на защиту материалы:

— разработаны студентом единолично под руководством преподавателя;

— не являются клоном (копией) чужих материалов;

— не нарушают лицензионные и авторские права третьих лиц;

— а также, что исходный код приложения не содержит материалов, размещенных в сети Интернет или опубликованных в литературе, в объеме, не позволяющем участнику претендовать на авторство и уникальность своего приложения.

ё

3. Защита курсовой работы

Защита курсовой работы учащимися проводится в публичном режиме и открыто для посещения всеми желающими.

3.1. Общая процедура защиты

1. Не позднее 2 недель до даты защиты деканат сообщает студентам дату, время и место проведения защиты.
2. До начала защиты студент отправляет преподавателю сформированный комплект материалов (п.2).
3. В назначенные день и время защиты, студент представляет перед комиссией выполненный проект с использованием ранее присланного комплекта материалов. Быть готовым продемонстрировать работу разработанной системы.
4. Комиссия заслушивает доклады и оценивает проекты.
5. Оценки за проекты проставляются в ведомости. Ведомость подписывается всеми членами комиссии.

3.2. Доклад студента при защите

Рекомендуемый регламент: презентация – 5-7 минут, затем ответы на вопросы комиссии – 5-10 минут. По просьбе комиссии студент должен быть готов продемонстрировать работу своей системы в действии.

Рекомендуемая структура доклада:

1. Введение: идея проекта, задачи, которые решает проект, существующие аналоги, целевая аудитория.
2. Демонстрация проекта: вживую, либо скриншоты/видео.
3. Описание реализации: архитектура проекта; описание датасета и технология его формирования; описание архитектуры нейронной сети с обоснованием выбора, описание стратегии обучения; интеграция нейросетевой модели в пользовательский продукт; сравнение с аналогами.
4. Заключение: планы по развитию и продвижению (в зависимости от научной, бизнес- или социальной ориентации проекта).

ё

3.3. Оценка

Курсовая работа оценивается комиссией в 10-бальной шкале.

Курсовая работа считается защищенной при получении оценки 4 и более баллов.

Решение принимается комиссией коллегиально.

При оценке презентации следует уделить внимание следующим критериям: грамотность и связность изложения, оформление слайдов, использование наглядных форм представления результатов (видео, демонстрация работы устройства/программы), соблюдение регламента.

ё

Литература

1. Нечеткая логика и нейронные сети: Учебное пособие / Н. В. Замятин - 2014. 292 с

2. Системы искусственного интеллекта: Учебное пособие / Н. В. Замятин - 2018. 244 с

3. Филипова, И. А. Правовое регулирование искусственного интеллекта : учебное пособие / И. А. Филипова. — Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2020. — 90 с.

4. Джонс, М. Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс. — Москва : ДМК Пресс, 2011. — 312 с

ё