Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.04.02 Информационные технологии и системная инженерия

Профиль: «Информационные технологии и системная инженерия»

**Индивидуальное задание**

по дисциплине

**«Искусственный интеллект и машинное обучение»**

по теме

**«Решить задачу классификации рукописных цифр (по базе MNIST). Использовать сети MLP, Хопфилда или Хемминга, CNN. Провести сравнительный анализ эффективности работы данных сетей»**

**Выполнил**

студент гр. ИТСИ-24-1м

**Юнусов Василь Равилевич**

**Принял** Доцент кафедры ВММБ

**Клюев Андрей Владимирович**

**Пермь 2024**

Введение

Нейронные сети представляют собой один из основных инструментов в области искусственного интеллекта и машинного обучения. Они имитируют работу биологических нейронных сетей, находящихся в мозге, и способны решать сложные задачи, такие как классификация данных, прогнозирование временных рядов, распознавание изображений и обработка текста. Основой теории нейронных сетей является концепция обучения через оптимизацию, что позволяет моделям улучшать свои прогнозы на основе предоставленных данных.

**Основные компоненты нейронной сети**

Искусственный нейрон

Искусственный нейрон — это математическая модель, которая принимает на вход несколько сигналов (данные), обрабатывает их с помощью весов, добавляет смещение и применяет функцию активации. Основное уравнение нейрона можно записать следующим образом:

где:

* xi — входные данные,
* wi — веса нейрона,
* b — смещение,
* f — функция активации,
* a — выход нейрона.

Функция активации определяет, как сигнал нейрона будет передан дальше в сеть. Популярные функции активации включают сигмоиду, ReLU (Rectified Linear Unit) и tanh.

Слои нейронной сети

Нейронные сети состоят из слоев:

1. Входной слой — принимает исходные данные.
2. Скрытые слои — обрабатывают данные, выделяя сложные зависимости и паттерны.
3. Выходной слой — возвращает результат в виде предсказания или классификации.

**Архитектура нейронной сети**

Архитектура определяет, сколько слоев и нейронов в каждом слое содержится в сети. Сети бывают:

* Прямого распространения (Feedforward): данные проходят от входного слоя к выходному без обратных связей.
* Рекуррентные (RNN): имеют обратные связи, что позволяет работать с последовательностями данных, например, временными рядами или текстом.
* Сверточные (CNN): оптимизированы для обработки изображений.

**Процесс обучения**

Обучение нейронной сети сводится к минимизации функции ошибки (или потерь), которая измеряет, насколько предсказания сети отличаются от реальных значений. Основные этапы:

1. Прямой проход (Forward Propagation)

Данные проходят через сеть, и вычисляется ошибка на выходе с использованием функции потерь, например, Mean Squared Error (MSE) для регрессии или Cross-Entropy для классификации.

1. Обратное распространение ошибки (Backpropagation)

Сеть корректирует свои веса, вычисляя градиенты ошибки с помощью метода обратного распространения и оптимизатора, например, стохастического градиентного спуска (SGD) или Adam.

1. Обновление весов

Веса корректируются по направлению уменьшения функции ошибки.

**Применение нейронных сетей**

Нейронные сети нашли широкое применение в различных областях:

* Распознавание образов — анализ изображений, например, классификация фотографий и распознавание лиц.
* Обработка текста и речи — машинный перевод, чат-боты, анализ тональности текста.
* Прогнозирование временных рядов — финансовые рынки, погодные условия.
* Автоматизация — управление роботами и автономными транспортными средствами.

Ход работы

Датасет

Для данной работы был использован датасет рукописных цифр из базы MNIST. Это уже обработанный и хорошо подготовленный датасет для изучения работы нейронных сетей. Он практически не требует никакой дополнительной обработки, более того, значения в нём уже нормализованы от 0 до 255. Всего в датасете 70 тыс. изображений, где 60 тыс. из них – тренировочные, а остальные 10 тыс. – тестовые. Каждый класс этого датасета имеет примерно одинаковое количество изображений, однако сбалансирован не идеально и разница в количестве изображений разных классов может доходить до 1,5 тыс.:

0: 6903

1: 7877

2: 6990

3: 7141

4: 6824

5: 6313

6: 6876

7: 7293

8: 6825

9: 6958

Ниже приведены случайные экземпляры обучающего и тестового множеств датасета:

|  |
| --- |
|  |
| Рис. 1. Экземпляры обучающего и тестового множеств датасета |

Построение и обучение моделей

В рамках данной работы были построены три модели нейронных сетей: MLP, Сеть Хемминга и CNN. Модель MLP является последовательной и состоит из одного скрытого слоя из 64 нейронов:

|  |
| --- |
| models.Sequential([  layers.Flatten(input\_shape=(28, 28, 1)),  layers.Dense(64, activation='relu'),  layers.Dropout(0.2),  layers.Dense(10, activation='softmax') ]) |

Модель свёрточной сети CNN состоит из 2 свёрточных блоков – из 4 слоёв:

|  |
| --- |
| models.Sequential([  layers.Conv2D(32, (3, 3), activation='relu', padding='same', input\_shape=(28, 28, 1)),  layers.MaxPooling2D((2, 2)),  layers.Dropout(0.2),   layers.Conv2D(64, (3, 3), activation='relu', padding='same'),  layers.MaxPooling2D((2, 2)),  layers.Dropout(0.2),    layers.Flatten(),  layers.Dense(64, activation='relu'),  layers.Dropout(0.2),  layers.Dense(10, activation='softmax') ]) |

Процесс обучения данных моделей приведен ниже.

|  |
| --- |
|  |
| Рис. 2. Процесс обучения модели MLP |

|  |
| --- |
|  |
| Рис. 3. Процесс обучения модели CNN |

Исходя из этих графиков хорошо видно, что оптимальное количество эпох для обучения данных моделей – 30 эпох, т.к. подходя к этому числу эпох, точность работы нейросетей перестает активно расти и дальнейшее обучение уже не имеет большого смысла.

Модель нейронной сети Хемминга была реализована вручную и без использования готовых библиотек. Суть работы данной модели заключается в том, что при обучении происходит вычисление средних значений пикселей изображений каждого класса, а при классификации происходит сравнение полученного на вход изображения с найденными прототипами. Наиболее вероятным считается тот класс, который показал минимальную сумму разностей пикселей изображения. Реализация данной модели приведена ниже.

|  |
| --- |
| class Hamming:  def \_\_init\_\_(self):  self.load\_and\_preprocess\_mnist()   images = []  for i in range(10):  images.append(self.X\_train[self.y\_train == i].mean(axis=0))  self.prototypes = np.array(images)   def classify(self, input\_vector):  distances = np.sum(np.abs(self.prototypes - input\_vector), axis=1)  return np.argmin(distances)   def load\_and\_preprocess\_mnist(self):  (x\_train, y\_train), (x\_test, y\_test) = datasets.mnist.load\_data()   x\_train = x\_train.reshape(-1, 28 \* 28)  x\_test = x\_test.reshape(-1, 28 \* 28)   self.X\_train = x\_train  self.X\_test = x\_test  self.y\_train = y\_train  self.y\_test = y\_test |

Было проведено обучение этих моделей на датасете рукописных цифр MNIST. Модели MLP и CNN прошли 30 эпох обучения. Полученные точности обученных моделей:

* MLP: 0.9745
* Сеть Хемминга: 0.6685
* CNN: 0.9906

Ниже приведена столбчатая диаграмма точности моделей:

|  |
| --- |
|  |
| Рис. 4. Столбчатая диаграмма точности моделей |

После обучения нейронных сетей также было приведено сравнение их работы между собой на одних и тех же случайных изображениях из тестового датасета. Сравнение проводилось на примере 10 изображений.

|  |
| --- |
|  |
| Рис. 5. Сравнение работы обученных нейронных сетей |

Исходя из этого сравнения хорошо видно, соответствие полученных точностей нейросетей с результатами их работы на практике. Сети MLP и CNN показали безошибочную работу на примере 10 случайных изображений. Сеть Хемминга же показала примерно 70% точности, что хорошо соотносится с полученными 66% процентами.

Код программы приведен в Приложении 1.

Вывод

В ходе работы были составлены и обучены три модели нейронной сети: MLP, Хемминга и CNN. Исходя из анализа эффективности моделей можно выделить то, что сеть Хемминга хуже всего подошла для решения данной задачи. Лучше всех решила задачу сеть CNN. MLP тоже очень успешно справилась с данной задачей и лишь незначительно отстает от CNN, но имеет намного более быструю скорость обучения, что позволяет считать её наиболее эффективной по соотношению точности ко времени обучения.

Приложение 1 Код программы

import os  
from random import randint  
  
import keras  
import matplotlib.pyplot as plt  
import numpy as np  
from keras import datasets, layers, models, optimizers, utils, Model  
from sklearn.metrics import accuracy\_score  
  
  
def get\_trained\_mlp\_model():  
 model = models.Sequential([  
 layers.Flatten(input\_shape=(28, 28, 1)),  
 layers.Dense(64, activation='relu'),  
 layers.Dropout(0.2),  
 layers.Dense(10, activation='softmax')  
 ])  
  
 model.compile(  
 optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy']  
 )  
  
 model.fit(  
 x\_train,  
 y\_train,  
 epochs=30,  
 validation\_data=(x\_test, y\_test)  
 )  
  
 return model  
  
  
class Hamming:  
 def \_\_init\_\_(self):  
 self.load\_and\_preprocess\_mnist()  
  
 images = []  
 for i in range(10):  
 images.append(self.X\_train[self.y\_train == i].mean(axis=0))  
 self.prototypes = np.array(images)  
  
 def classify(self, input\_vector):  
 distances = np.sum(np.abs(self.prototypes - input\_vector), axis=1)  
 return np.argmin(distances)  
  
 def load\_and\_preprocess\_mnist(self):  
 (x\_train, y\_train), (x\_test, y\_test) = datasets.mnist.load\_data()  
  
 x\_train = x\_train.reshape(-1, 28 \* 28)  
 x\_test = x\_test.reshape(-1, 28 \* 28)  
  
 self.X\_train = x\_train  
 self.X\_test = x\_test  
 self.y\_train = y\_train  
 self.y\_test = y\_test  
  
  
def get\_trained\_cnn\_model():  
 model = models.Sequential([  
 layers.Conv2D(32, (3, 3), activation='relu', padding='same', input\_shape=(28, 28, 1)),  
 layers.MaxPooling2D((2, 2)),  
 layers.Dropout(0.2),  
  
 layers.Conv2D(64, (3, 3), activation='relu', padding='same'),  
 layers.MaxPooling2D((2, 2)),  
 layers.Dropout(0.2),  
  
 # layers.Conv2D(128, (3, 3), activation='relu', padding='same'),  
 # layers.MaxPooling2D((2, 2)),  
 # layers.Dropout(0.2),  
  
 layers.Flatten(),  
 layers.Dense(64, activation='relu'),  
 layers.Dropout(0.2),  
 layers.Dense(10, activation='softmax')  
 ])  
  
 opt = optimizers.SGD(learning\_rate=0.001, momentum=0.9)  
 model.compile(  
 optimizer=opt,  
 loss='categorical\_crossentropy',  
 metrics=['accuracy']  
 )  
  
 model.fit(  
 x\_train,  
 y\_train,  
 epochs=30,  
 validation\_data=(x\_test, y\_test),  
 shuffle=True  
 )  
  
 return model  
  
  
def train\_or\_load\_keras\_model(train\_model\_func, trained\_model\_path: str, force\_train: bool = False):  
 if not force\_train and os.path.exists(trained\_model\_path):  
 model = keras.models.load\_model(trained\_model\_path)  
 else:  
 model = train\_model\_func()  
  
 model.save(trained\_model\_path)  
  
 model.summary()  
 return model  
  
  
def show\_random\_images(count=10):  
 fig, ax = plt.subplots(2, count, figsize=(10, 5))  
  
 train\_axes = ax[0].flatten()  
 test\_axes = ax[1].flatten()  
 for i in range(count):  
 train\_axes[i].axis('off')  
 test\_axes[i].axis('off')  
  
 train\_image\_index = randint(0, len(x\_train))  
 test\_image\_index = randint(0, len(x\_test))  
 train\_axes[i].imshow(x\_train[train\_image\_index])  
 test\_axes[i].imshow(x\_test[test\_image\_index])  
  
 train\_axes[i].set\_title(np.argmax(y\_train[train\_image\_index]))  
 test\_axes[i].set\_title(np.argmax(y\_test[test\_image\_index]))  
  
 train\_y = train\_axes[0].get\_position().ymax  
 test\_y = test\_axes[0].get\_position().ymax  
  
 fig.text(0.5, train\_y + 0.1, "Образцы из тренировочного набора", ha='center', fontsize=28)  
 fig.text(0.5, test\_y + 0.1, "Образцы из тестового набора", ha='center', fontsize=28)  
  
 plt.show()  
  
  
def show\_model\_training\_metrics(history: dict, label):  
 try:  
 loss = history['loss']  
 val\_loss = history['val\_loss']  
 accuracy = history['accuracy']  
 val\_accuracy = history['val\_accuracy']  
 except Exception:  
 print(f'Не удалось извлечь историю обучения модели {label} для построения графиков.')  
 return  
  
 px = 1 / plt.rcParams['figure.dpi']  
 fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(1800\*px, 500\*px))  
  
 fig.text(0.5, 0.9, label, ha='center', fontsize=28)  
  
 ax1.plot(loss, color='green')  
 ax1.plot(val\_loss, color='red')  
 ax1.legend(['loss', 'validate loss'])  
 ax1.grid(True)  
  
 ax2.plot(accuracy, color='green')  
 \_ = ax2.plot(val\_accuracy, color="red")  
 ax2.legend(['accuracy', 'validate accuracy'])  
  
 plt.show()  
  
  
def show\_predictions(mlp\_model: Model, hamming\_model: Hamming, cnn\_model: Model, count=10):  
 fig, ax = plt.subplots(3, count, figsize=(10, 5))  
  
 mlp\_axes = ax[0].flatten()  
 hamming\_axes = ax[1].flatten()  
 cnn\_axes = ax[2].flatten()  
 for i in range(count):  
 mlp\_axes[i].axis('off')  
 hamming\_axes[i].axis('off')  
 cnn\_axes[i].axis('off')  
  
 test\_image\_index = randint(0, len(x\_test))  
 test\_image = x\_test[test\_image\_index]  
 test\_answer = np.argmax(y\_test[test\_image\_index])  
 hamming\_test\_image = hamming\_model.X\_test[test\_image\_index]  
  
 mlp\_axes[i].imshow(test\_image)  
 hamming\_axes[i].imshow(test\_image)  
 cnn\_axes[i].imshow(test\_image)  
  
 mlp\_prediction = np.argmax(mlp\_model.predict(np.expand\_dims(test\_image, 0)))  
 hamming\_prediction = hamming\_model.classify(hamming\_test\_image)  
 cnn\_prediction = np.argmax(cnn\_model.predict(np.expand\_dims(test\_image, 0)))  
  
 mlp\_axes[i].set\_title(f'{mlp\_prediction}{"=" if mlp\_prediction == test\_answer else "<>"}{test\_answer}', color="g" if mlp\_prediction == test\_answer else "r")  
 hamming\_axes[i].set\_title(f'{hamming\_prediction}{"=" if hamming\_prediction == test\_answer else "<>"}{test\_answer}', color="g" if hamming\_prediction == test\_answer else "r")  
 cnn\_axes[i].set\_title(f'{cnn\_prediction}{"=" if cnn\_prediction == test\_answer else "<>"}{test\_answer}', color="g" if cnn\_prediction == test\_answer else "r")  
  
 fig.text(mlp\_axes[0].get\_position().xmin, mlp\_axes[0].get\_position().ymax + 0.07, "MLP", ha='left', fontsize=20)  
 fig.text(hamming\_axes[0].get\_position().xmin, hamming\_axes[0].get\_position().ymax + 0.07, "Сеть Хемминга", ha='left', fontsize=20)  
 fig.text(cnn\_axes[0].get\_position().xmin, cnn\_axes[0].get\_position().ymax + 0.07, "CNN", ha='left', fontsize=20)  
  
 fig.text(0.5, 0.05, "Результаты работы нейросетей", ha='center', fontsize=24)  
  
 plt.show()  
  
  
def show\_accuracies\_comparison(accuracies: dict[str, str]):  
 plt.bar(accuracies.keys(), accuracies.values())  
 plt.xlabel('Модель нейронной сети')  
 plt.ylabel('Точность')  
 plt.title('Сравнение эффективности \nмоделей классификации рукописных цифр')  
 plt.show()  
  
  
def init\_keras\_dataset():  
 global x\_train, y\_train, x\_test, y\_test  
 (x\_train, y\_train), (x\_test, y\_test) = datasets.mnist.load\_data()  
  
 x\_train = np.expand\_dims(x\_train, -1)  
 x\_test = np.expand\_dims(x\_test, -1)  
  
 y\_train = utils.to\_categorical(y\_train, 10)  
 y\_test = utils.to\_categorical(y\_test, 10)  
  
  
TRAINED\_MODELS\_DIR = 'trained\_models'  
  
MLP\_TRAINED\_MODEL\_PATH = f'{TRAINED\_MODELS\_DIR}/mnist\_mlp.keras'  
CNN\_TRAINED\_MODEL\_PATH = f'{TRAINED\_MODELS\_DIR}/mnist\_cnn.keras'  
  
x\_train = np.array([])  
y\_train = np.array([])  
x\_test = np.array([])  
y\_test = np.array([])  
  
FORCE\_RETRAIN = True  
  
  
def main():  
 init\_keras\_dataset()  
  
 show\_random\_images()  
  
 # Модели  
 mlp\_model = train\_or\_load\_keras\_model(get\_trained\_mlp\_model, MLP\_TRAINED\_MODEL\_PATH, FORCE\_RETRAIN)  
 hamming\_model = Hamming()  
 cnn\_model = train\_or\_load\_keras\_model(get\_trained\_cnn\_model, CNN\_TRAINED\_MODEL\_PATH, FORCE\_RETRAIN)  
  
 # Графики процесса обучения моделей  
 if FORCE\_RETRAIN:  
 show\_model\_training\_metrics(mlp\_model.history.history, label="MLP")  
 show\_model\_training\_metrics(cnn\_model.history.history, label="CNN")  
  
 # Точности  
 mlp\_accuracy = mlp\_model.evaluate(x\_test, y\_test)[1]  
 hamming\_accuracy = accuracy\_score(hamming\_model.y\_test, [hamming\_model.classify(x) for x in hamming\_model.X\_test])  
 cnn\_accuracy = cnn\_model.evaluate(x\_test, y\_test)[1]  
  
 print('\nТочности работы нейросетей на тестовых данных:')  
 print(f"MLP: {mlp\_accuracy:.4f}")  
 print(f"Сеть Хемминга: {hamming\_accuracy:.4f}")  
 print(f"CNN: {cnn\_accuracy:.4f}")  
 print()  
  
 # Графики  
 ## Эффективность  
 labels = ['MLP', 'Сеть Хемминга', 'CNN']  
 accuracies = [mlp\_accuracy, hamming\_accuracy, cnn\_accuracy]  
 show\_accuracies\_comparison(dict(zip(labels, accuracies)))  
  
 ## Примеры предсказывания нейросетей  
 show\_predictions(mlp\_model=mlp\_model, hamming\_model=hamming\_model, cnn\_model=cnn\_model)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()