УДК 004.8

Обучение нейронный сети сложение два переменных числа

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования модели нейронный сети для обучения простой операции сложения два числа. В работе использовалась библиотека Torch, и модель MLP (многослойный перцептрон), набор данных представлялся собой таблица сложения числа. В результате работы, модель обучилась складывать два числа, а также была получена оценка точность предсказания модели.
Ключевые слова: MLP, таблица сложения, линейная нейронная сеть, Torch.

Training a neural network to add two variable numbers

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of using a neural network model for the simple operation of adding two numbers. The Torch library was used in the work, and the MLP model (multilayer perceptron), the data set was a number addition table. As a result of the work, the model learned to add two numbers, and an estimate of the model's prediction accuracy was also obtained.
Keywords: MLP, addition table, linear neural network, Torch.

1 Введение

1.1. Актуальность исследования

Актуальность исследования заключается в том, что арифметические операции необходимы для разработки основ машинного обучения. Сложение и вычитание значений — это основная математическая операция, которая лежит в основе различных статистических операций и алгоритмов. Кроме того, таблица сложения является полезным инструментом для демонстрации концепции линейной регрессии в машинном обучении.

1.2. Цель исследования

Целью работы является создания и обучение модели для сложения двух чисел.

1.3. Обзор исследований

Л. Париси, проводит анализ производительность двух алгоритмов машинного обучения, SVM и MLP, в библиотеке scikit-learn. Автор

исследователь показал, что новая реализация функции M-assinh значительно повышает эффективность и надежность обоих алгоритмов. Полученные данные свидетельствуют о том, что улучшенная функция может стать полезным инструментом для исследователей и практиков, работающих в области машинного обучения и анализа данных. Однако в исследовании отсутствуют подробные сведения о причинах повышения производительности и его потенциальных ограничениях. В целом, исследование вносит ценный вклад в области машинного обучения и науки о данных [1].

Н. А. Хамид, Н. Н. А. Сяриф проводит анализ три различных алгоритма машинного обучения: SVM, Knn и нейронная сеть на точность для распознавания рукописного текста. Авторы работ проводит сравнительный анализ производительность этих алгоритмов и подробно анализируют их сильные и слабые стороны. Результаты показывают, что нейронная сеть превосходит два других алгоритма с точки зрения точности, в то время как SVM и Knn демонстрируют более быстрое время обработки. В целом, исследование дает ценную информацию о применимости различных методов машинного обучения для распознавания рукописного текста [2].

А. Е. И. Брощнлее и др. использовали различные алгоритмы машинного обучения для разных наборов данных, чтобы измерить энергопотребление и точность каждого из них. Они обнаружили, что действительно существует компромисс между точностью и энергопотреблением, и что некоторые алгоритмы более энергоэффективны, чем другие. В заключении исследования подчеркивается важность учета энергопотребления при разработке алгоритмов машинного обучения [3].

Ж. Хонг и др. рассматривает разработку и оценке комбинированных моделей машинного обучения для прогнозирования притока плотины. Используя данные в режиме реального времени, полученные от плотины в Южной Корее, авторы исследователи используют несколько алгоритмов машинного обучения для разработки гибридной модели, которая может точно прогнозировать приток воды к плотине. В работе представлено подробное описание процесса моделирования, включая предварительную обработку данных, выбор признаков и оценку модели. Авторы исследователи сообщают, что гибридная модель демонстрирует более высокую точность предсказания, чем отдельные модели и традиционные статистические методы. В целом, работа демонстрирует потенциал объединения алгоритмов машинного обучения для повышения точности прогнозирования притока плотины, что может помочь в принятии обоснованных решений по эксплуатации плотины и управлению [4].

А. Истхиая, Н. А. Саиф, описывает этапы предварительной обработки и методы извлечения признаков, используемые для подачи данных в нейронную сеть, и сообщают о точности распознавания символов более 97%. Они также провели сравнительный анализ с другими методами OCR, используемыми для распознавания символов Bangla, и пришли к выводу, что предложенный ими метод нейронной сети превосходит эти методы с точки зрения точности. Исследование вносит ценный вклад в область оптического распознавания

текста на бенгальском языке и может иметь практическое применение в оцифровке документов и языковой обработке [5].

Ф. Стиннер и др. предлагает подход машинного обучения к структурированию идентификаторов точек данных технического оборудования здания в контексте Айкидо, программной платформы для автоматизации и управления зданием. Они применяют различные алгоритмы машинного обучения для определения и классификации точек данных и оценивают их производительность, используя реальные данные из разных зданий. Результаты показывают, что предложенный подход эффективен для точного распознавания и классификации точек данных и может повысить эффективность и точность систем автоматизации зданий. В целом исследование демонстрирует потенциал машинного обучения для облегчения управления сложными техническими системами в зданиях [6].

2. Рабочий процесс

2.1. Набор данных

Набор данных представлен таблица сложений, содержит 10 201 записи.

Таблица 1. Таблица сложений.

a	b	c
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
...		
100	96	196
100	97	197
100	98	198
100	99	199
100	100	200

В наборе данных a и b исходные данные, c — правильный ответ.

Листинг 2.1. Генерация набор данных.

```
1  df = pd.DataFrame(list(itertools.product(range(0, 101), repeat=2)), columns=["a", "b"])
2  df["c"] = df["a"] + df["b"]
3  df.to_excel("data_sum.xlsx", index=None)
```

Строка 1 Создания набор данных

Строка 2. Создания «правильных ответов» столбцов.

Строка 3. Сохранение набор данных.

Листинг 2.2. Генерация набор данных.

```
1  X_train, X_test, y_train, y_test = train_test_split(df[["a", "b"]], df["c"],
2                                                      train_size=0.6,
3                                                      random_state=42)
4  n_range = [df.min().min(), df.max().max()]
5  def norm(x, n_range):
6          return (x - n_range[0]) / (n_range[1] - n_range[0])
7  def norm_reset(x, n_range):
8          return x * (n_range[1] - n_range[0]) + n_range[0]
9  tensor_x = torch.tensor(X_train.values, dtype=torch.float, device=device)
10 tensor_y = torch.tensor(y_train.values, dtype=torch.float, device=device)
11 tensor_x = norm(tensor_x, n_range)
12 tensor_y = norm(tensor_y, n_range).unsqueeze(1)
```

Строка 1 — 3. разделение на тренировочные и тестовые данные, на 40% тестовые данные.

Строка 4 — 8. подготовка нормировка данных.

Строка 9 — 12. нормировка и преобразования в тензор данные.

2.2. Модель

Листинг 2.3. Структура модель MLP.

```
MLP(
  (Dense_in): Linear(in_features=2, out_features=100, bias=True)
  (Dense_0): Linear(in_features=100, out_features=100, bias=True)
  (Dense_1): Linear(in_features=100, out_features=80, bias=True)
  (Dense_out): Linear(in_features=80, out_features=1, bias=True)
)
```

Модель принимает два значений, а на выходе один значение.

Модель содержит 2 скрытых слоев и 2 слоев вход и выход.

Листинг 2.4. Параметры для обучения модели

```
1  class MLP(nn.Module):
2          def __init__(self, in_dim, hidden_dim, out_dim):
3                  super(MLP, self).__init__()
4                  self.in_dim = in_dim
5                  self.hidden_dim = hidden_dim
6                  self.out_dim = out_dim
7                  self.Dense_in = nn.Linear(self.in_dim, self.hidden_dim[0])
8                  self.Dense_0 = nn.Linear(self.hidden_dim[0], self.hidden_dim[1])
9                  self.Dense_1 = nn.Linear(self.hidden_dim[1], self.hidden_dim[2])
10                 self.Dense_out = nn.Linear(self.hidden_dim[2], self.out_dim)
11         def forward(self, x):
12                 x = self.Dense_in(x)
13                 x = self.Dense_0(x)
14                 x = self.Dense_1(x)
15                 x = torch.sigmoid(self.Dense_out(x))
16                 return x
17 model = MLP(X_train.shape[1], [100, 100, 80], 1).to(device)
18 optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
19 m_mae = nn.L1Loss()
```

Строка 1 - 16. Класс MLP модель.

Строка 17. Модель.
Строка 18. Оптимизатор Adam, скорость обучение 0,0001.
Строка 19. Метрика L1.

2.3. Обучение

Рисунок 1. Обучение модели

Написали программу на языке Python и использовали библиотеку Torch. Чтобы обучить модель задали эпохи 3000 с скоростью обучения 0.0001, чтобы оценить модель вовремя обучения использовали метрику L1Loss. Размер пакета мы выбрали 512, чтобы программа обучения выполнялась быстро.

Листинг 2.5. Функция для обучение модель

```
def fit(model, x_data, y_data, epochs = 20, batch_size = 128):
        start_time = time.time()
        time_train = time.time()
        sl_ind = None
        ind = None
        i_max = None
        for epoch in range(epochs):
                ind = torch.randperm(len(x_data))
                i_max = int(np.ceil(ind.shape[0] / batch_size))
                for i in range(0, i_max):
                        time_train = time.time()
                        sl_ind = ind[i * batch_size: (i + 1) * batch_size]
                        model.train()
                        batch_x, batch_y = shuffle(x_data[sl_ind], y_data[sl_ind])
                        #print("batch_x, batch_y: ", batch_x.shape, " ; ", batch_y.shape)
                        optimizer.zero_grad()

                        y_pred = model(batch_x)
                        loss = m_mae(y_pred, batch_y)
                        loss.backward()
                        optimizer.step()

                        metric["time"].append(time.time() - time_train)
                        metric["loss"].append(loss.item())
                if epoch % 10 == 9:
                        elapsed_time = time.time() - start_time
                        print('\rEpoch:{} Loss:{:.4f} Time: {:.2f}s'.format(
                                epoch + 1, loss.item(),
                                elapsed_time)
                        )
```

```
31                          start_time = time.time()
32          print()
```

Строка 2 - 6, инициализация перемены.

Строка 7, цикл обучения по эпохам.

Строка 8 — 9. генерируем случайные выбранные пакеты.

Строка 11 — 24. обучаем модель и записываем метрику, время и функция потерь.

Строка 25 — 31. выводим в окно информация метрика и эпоха, время прохождения шага обучения.

2.4. Оценка модели

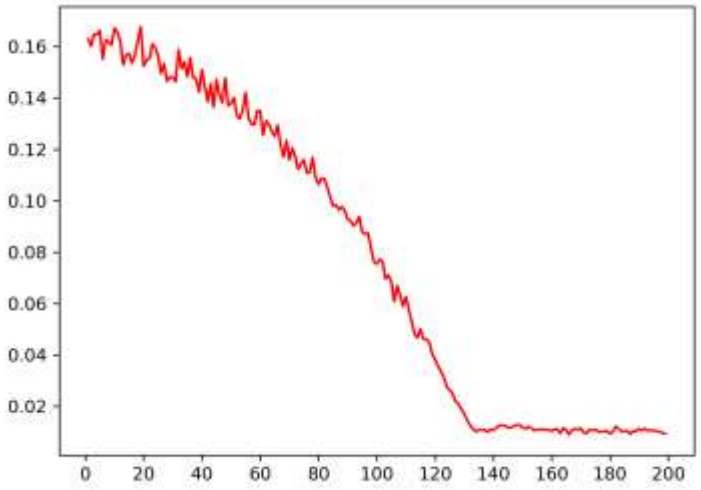

Рисунок 2. Функция потерь (метрика L1Loss)

Графике представлен по осью X — цикл (шаг) обучение условно выбрано 0 — 200, а по осью Y — метрика функция потерь. Функция потерь уменьшилась до примерно 110 цикл обучения. Средняя функция потерь значение начиная 110 цикла до 36000 — 0,01040. Минимальная значение функция потерь который удалось достигнуть - 0,0078884.

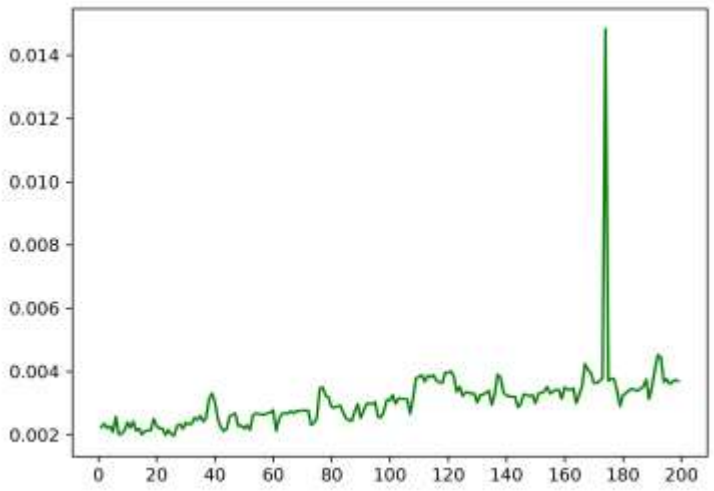

Рисунок 3. Время выполнения обучения.

На графике представлено время выполнения обучения считается каждый шаг с задержкой выполнения в секундах по осью Y.

Время выполнение обучение было 150,884 секунда (2 минута, 30,884 секунда).

Точность (accuracy) предсказания модели на обучающий выборках 21,601%. А на тестовых выборках — 21,882%.

3 Выводы

В данной статьи была использована модель MLP (многослойный перцептрон) для обучения возможности складывать числа, набор данных представлял собой два исходные числа, а с целевой — результат сложения числа, в результате работы получили модель способный складывать числа, однако точность (accuracy) оценки модели на тестовых образцах получили 21,882%, для функция потерь минимальный значение удалось достигнуть 0,0078884.

Библиографический список

1. Parisi L. m-arcsinh: An Efficient and Reliable Function for SVM and MLP in scikit-learn //arXiv preprint arXiv:2009.07530. 2020.
2. Hamid N. A., Sjarif N. N. A. Handwritten recognition using SVM, KNN and neural network //arXiv preprint arXiv:1702.00723. 2017.
3. Brownlee A. E. I. et al. Exploring the accuracy–energy trade-off in machine learning //2021 IEEE/ACM International Workshop on Genetic Improvement (GI). IEEE, 2021. С. 11-18.
4. Hong J. et al. Development and evaluation of the combined machine learning models for the prediction of dam inflow //Water. 2020. Т. 12. №. 10. С. 2927.
5. Isthiaq A., Saif N. A. OCR for printed bangla characters using neural network //International Journal of Modern Education and Computer Science. 2020. Т. 12. №. 2. С. 19.
6. Stinner F. et al. Aikido: Structuring data point identifiers of technical building equipment by machine learning //Journal of Physics: Conference Series. IOP Publishing, 2019. Т. 1343. №. 1. С. 012039.

4. Приложения

Листинг 4.1. Исходный код программы

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import pandas as pd
import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import time
import numpy as np
import itertools
from collections import deque
import matplotlib.pyplot as plt

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
df = pd.DataFrame(list(itertools.product(range(0, 101), repeat=2)), columns=["a", "b"])
df["c"] = df["a"] + df["b"]

df.to_excel("data_sum.xlsx", index=None)
X_train, X_test, y_train, y_test = train_test_split(df[["a", "b"]], df["c"],
                                                    train_size=0.6,
                                                    random_state=42)
n_range = [df.min().min(), df.max().max()]

def norm(x, n_range):
    return (x - n_range[0]) / (n_range[1] - n_range[0])
def norm_reset(x, n_range):
    return x * (n_range[1] - n_range[0]) + n_range[0]

tensor_x = torch.tensor(X_train.values, dtype=torch.float, device=device)
tensor_y = torch.tensor(y_train.values, dtype=torch.float, device=device)

tensor_x = norm(tensor_x, n_range)
tensor_y = norm(tensor_y, n_range).unsqueeze(1)

class MLP(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(MLP, self).__init__()
        self.in_dim = in_dim
        self.hidden_dim = hidden_dim
        self.out_dim = out_dim
        self.Dense_in = nn.Linear(self.in_dim, self.hidden_dim[0])
        self.Dense_0 = nn.Linear(self.hidden_dim[0], self.hidden_dim[1])
        self.Dense_1 = nn.Linear(self.hidden_dim[1], self.hidden_dim[2])
        self.Dense_out = nn.Linear(self.hidden_dim[2], self.out_dim)
    def forward(self, x):
        x = self.Dense_in(x)
        x = self.Dense_0(x)
        x = self.Dense_1(x)
        x = torch.sigmoid(self.Dense_out(x))
        return x

model = MLP(X_train.shape[1], [100, 100, 80], 1).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
m_mae = nn.L1Loss()
metric = {
    "time": deque(),
    "loss": deque()
}

def fit(model, x_data, y_data, epochs = 20, batch_size = 128):
    start_time = time.time()
    time_train = time.time()
    sl_ind = None
    ind = None
    i_max = None
    for epoch in range(epochs):
        ind = torch.randperm(len(x_data))
        i_max = int(np.ceil(ind.shape[0] / batch_size))
        for i in range(0, i_max):
            time_train = time.time()
            sl_ind = ind[i * batch_size: (i + 1) * batch_size]
```

```
                                        model.train()
                                        batch_x, batch_y = shuffle(x_data[sl_ind], y_data[sl_ind])
                                        #print("batch_x, batch_y: ", batch_x.shape, " ; ", batch_y.shape)
                                        optimizer.zero_grad()

                                        y_pred = model(batch_x)
                                        loss = m_mae(y_pred, batch_y)
                                        loss.backward()
                                        optimizer.step()

                                        metric["time"].append(time.time() - time_train)
                                        metric["loss"].append(loss.item())
                                if epoch % 10 == 9:
                                        elapsed_time = time.time() - start_time
                                        print('\rEpoch:{} Loss:{:.4f} Time: {:.2f}s'.format(
                                                epoch + 1, loss.item(),
                                                elapsed_time)
                                        )
                                        start_time = time.time()
        print()
def predict(model, x):
        global n_range
        model.eval()
        x = torch.tensor(x, dtype=torch.float, device=device)
        x = norm(x, n_range)
        return norm_reset(model(x), n_range).round().cpu().detach().numpy()

epochs = 3000
fit(model, tensor_x, tensor_y, epochs, batch_size=512)
df_metric = pd.DataFrame(metric)
df_metric.to_csv("metric.csv", index=None)
df_metric_sel = df_metric[1:200]

#-------------------------------------------------------------
#x_step = 3600
x_step = 20

fig, ax = plt.subplots()
ax.plot(df_metric_sel.index, df_metric_sel["loss"], label ='loss', color="red")
ax.set_xticks(np.arange(0, max(df_metric_sel.index)+2, x_step))
fig.savefig("./{0}.png".format("loss"), dpi = 300)

fig, ax = plt.subplots()
ax.plot(df_metric_sel.index, df_metric_sel["time"], label ='time', color="green")
ax.set_xticks(np.arange(0, max(df_metric_sel.index)+2, x_step))
fig.savefig("./{0}.png".format("time"), dpi = 300)


tensor_x_test = torch.tensor(X_test.values, dtype=torch.float, device=device)
tensor_y_test = torch.tensor(y_test.values, dtype=torch.float, device=device)

tensor_x_test = norm(tensor_x_test, n_range)
tensor_y_test = norm(tensor_y_test, n_range).unsqueeze(1)

def predict_test(model, x_data):
        output = model(x_data)
        return output

# Train
y_pred = norm_reset(predict_test(model, tensor_x).squeeze(1), n_range).round().cpu().detach().numpy()
accuracy = 100 * (y_pred == y_train).sum() / len(tensor_x)
accuracy

# Test
y_pred_test = norm_reset(predict_test(model, tensor_x_test).squeeze(1), n_range).round().cpu().detach().numpy()
accuracy = 100 * (y_pred_test == y_test).sum() / len(tensor_x_test)
accuracy
```