Въведение в Машинното обучение
Лабораторно упражнение №9

TensorFlow

1. Цел на упражнението.

Целта на лабораторното упражнение е студентите да усвоят темата невронна мрежа, като се използва платформата TensorFlow.

2. TensorFlow

TensorFlow е платформа за машинно обучение с отворен код, като я прави достъпна за всеки разработчик и хора от научната общност, които да подобрят изходния код.

TensorFlow произлиза от необходимостта на Google да инструктира компютърната система да имитира как работи човешкият мозък в обучението и разсъжденията. Системата, известна като невронни мрежи, трябва да може да изпълнява многомерни масиви от данни, наречени „тензори“. Крайната цел е да се обучат невронните мрежи за откриване и дешифриране на модели и корелации.

TensorFlow първоначално е разработен от изследователи и инженери, работещи в екипа на Google Brain в рамките на изследователската организация на Machine Intelligence за целите на провеждането на машинно обучение и дълбоки изследвания на невронни мрежи, но системата е достатъчно обща, за да бъде приложима в широк спектър от други области. Системният код е написан на C ++ и Python и се разпространява под лиценза Apache.

TensorFlow има много функционалности, включително вградени модели за машинно обучение, като линейна регресия, класификация, рекурентни невронни мрежи и други. Платформата позволява също да се създават и обучават собствени модели с помощта на функциите за оптимизация, графично изображение и др.

TensorFlow е наличен като библиотека на езиците Python, C ++ и Java и е наличен за Windows, Linux и macOS. TensorFlow е използван в много различни области, включително визуално разпознаване, естествен език обработка, роботика, игри и много други.

Гъвкавата архитектура позволява да се внедряват изчисления на един или повече процесори или графични процесори в настолен компютър, сървър или мобилно устройство с един API. TensorFlow може да работи на голямо разнообразие от процесори и графични процесори, но работи особено добре със собствените Tensor Processing Units (TPU) на Google. Разработчиците могат също да използват мощта на Google Cloud Platform, като възлагат операции на машинно обучение на сървърите на Google.

Използва се от големи компании по целия свят, включително Airbnb, Ebay, Dropbox, Snapchat, Twitter, Uber, SAP, Qualcomm, IBM, Intel и, разбира се, Google.

Основи на TensorFlow - https://www.tensorflow.org

Уроци за начинаещи и експерти - https://www.tensorflow.org/tutorials

Уроци за начинаещи - https://www.tensorflow.org/tutorials/quickstart/beginner

Уроците за TensorFlow са написани като тетрадки Jupyter и се изпълняват директно в **Google Colab** - хоствана среда за бележници, която не изисква настройка.

3. Google Colab

Google Colab е безплатна онлайн среда за изпълнение на Python код и машинно обучение, която се предоставя от Google. Това е част от Google Drive и използва Jupyter Notebook като интерфейс. Colab ви позволява да пишете, изпълнявате и съхранявате Python код, включително библиотеки за машинно обучение като TensorFlow, Keras и PyTorch, на облачни компютри на Google.

За да използвате Google Colab, трябва да имате Google акаунт. След като сте влезли в акаунта си, можете да отворите Google Colab от менюто Google Drive, като щракнете върху "Ново" и след това изберете "Повече" и "Google Colaboratory". Това ще отвори нов прозорец в браузъра с Jupyter Notebook интерфейс.

В Google Colab можете да създавате нови бележници и да пишете и изпълнявате Python код, като използвате клетки. Можете да добавите клетки за код, за текстово форматиране или за маркиране на бележника. Кодът се изпълнява на облачен компютър, а изходът се показва в самия бележник. Можете да съхранявате бележниците си в Google Drive или GitHub.

За да достъпите Google Colab, трябва да посетите адреса на уеб сайта: https://colab.research.google.com/. Там можете да влезете със своя Google акаунт и да започнете да използвате платформата.

4. Допълнителни пояснения към кода на задача 1

а) за ред 7

7) iris = load_iris()

"load_iris" е функция в библиотеката scikit-learn, която зарежда набор от данни за цветята iris. Този набор от данни е често използван за тестване на алгоритми за класификация и регресия.

Наборът от данни включва общо 150 примера на различни видове цветя iris, като за всеки е записана информация за 4 характеристики на цветята - дължина и ширина на чашката и листата, измерени в сантиметри. За всеки пример е зададен и клас, към който принадлежи цветето - setosa, versicolor или virginica.

"load_iris" функцията връща обект с атрибути data и target, където data е матрица от характеристиките на цветята (размерност 150x4) и target е масив от класовете на цветята (размерност 150x1).

б) за ред 8

8) X, y = iris.data, iris.target

На ред 8 се използват атрибутите data и target на обекта iris, за да се зададат променливите X и y. X е матрица от характеристиките на цветята, а y е масив от класовете на цветята. Така X и y ще бъдат използвани за обучение и тестване на невронната мрежа.

в) за редове 12, 13 и 14

12) lb = LabelBinarizer()

13) y_train = lb.fit_transform(y_train)

14) y_test = lb.transform(y_test)

Тези редове код служат за преобразуване на целевите променливи (маркировките) на класификационната задача в множество от еднокомпонентни маркери. Това е необходимо, за да може невронната мрежа да работи с тези маркери.

На ред 12 се създава обект от клас LabelBinarizer(), който ще се използва за преобразуване на маркировките.

На ред 13 се извиква методът fit_transform(y_train) на обекта lb, като му се подава масивът y_train, който съдържа маркировките за обучение. Този метод преобразува маркировките в множество от еднокомпонентни маркери, които се записват в y_train.

На ред 14 се извиква методът transform(y_test) на обекта lb, като му се подава масивът y_test, който съдържа маркировките за тест. Този метод преобразува маркировките в множество от еднокомпонентни маркери, които се записват в y_test.

Така сега маркировките y_train и y_test са преобразувани в множество от еднокомпонентни маркери, които ще бъдат използвани за обучение и тестване на невронната мрежа.

г) от ред 16 до ред 20

16) model = tf.keras.Sequential([

17) tf.keras.layers.Dense(64, activation='relu', input_shape=(4,)),

18) tf.keras.layers.Dense(64, activation='relu'),

19) tf.keras.layers.Dense(3, activation='softmax')

20)])

Тези редове създават невронна мрежа, която се състои от три слоя Dense. Първите два слоя имат по 64 неврона с функция на активация 'relu', а последният слой има три неврона с функция на активация 'softmax'. Входните данни имат размерност (4,), която е посочена чрез параметъра input_shape на първия слой. Тази мрежа е предназначена за класификация на цветовете на ириси в три класа.

Създава се невронна мрежа с помощта на Keras API от TensorFlow. Моделът е тип Sequential, което означава, че невроните са подредени последователно във вход-изход посока.

tf.keras.layers.Dense е слой от неврони, който има пълно свързване между предишния и следващия слой. Това означава, че всеки неврон в предишния слой е свързан с всеки неврон в следващия слой. В първите два слоя има по 64 неврона, които използват

функцията за активация "relu" (Rectified Linear Unit). Функцията "relu" има нелинейна форма, която помага за моделиране на сложни връзки в данните.

В последния слой има 3 неврона, като функцията за активация е "softmax". Функцията "softmax" се използва за мултикласова класификация и извежда вероятностно разпределение за всяка класификационна категория. Класификационната категория, която има най-висока вероятност, се избира като предсказан клас.

д) от ред 22 до ред 24

22) model.compile(optimizer='adam',

23) loss='categorical_crossentropy',

24) metrics=['accuracy'])

Редовете 22-24 се отнасят за компилацията на модела и определят как ще бъде обучаван моделът.

optimizer определя оптимизационния алгоритъм, който ще бъде използван по време на обучението. В този случай се използва оптимизаторът 'adam', който е адаптивен метод за стохастична оптимизация (Стохастичната оптимизация е вид оптимизационен алгоритъм, който се използва за намиране на глобален минимум или максимум на функция.)

loss определя функцията за грешка, която ще се използва по време на обучението. В този случай се използва функцията 'categorical_crossentropy', която се използва за многокласова класификация, когато класовете са единствено взаимно изключващи се.

metrics определя метриката, която ще се използва за оценка на производителността на модела. В този случай метриката е 'accuracy', която показва процента на правилните класификации.

е) за ред 26

26) model.fit(X_train, y_train, epochs=50, batch_size=8, validation_data=(X_test, y_test))

Този ред от кода служи за обучение на модела с помощта на тренировъчните данни (X_train и y_train). В него се извиква методът fit() на обекта model, като му се подават следните параметри:

X_train: Тренировъчните данни за признаците.

y_train: Тренировъчните метки (целевите прогнози).

epochs: Брой на епохите (преоходи) на тренировката. Всяка епоха се състои от преход върху всички тренировъчни данни.

batch_size: Размерът на пакетите, в които да се изпълняват изчисленията по време на тренировката. По подразбиране batch_size е 32.

validation_data: Данни, които се използват за валидация на модела по време на тренировката. В този случай те са тестовите данни X_test и y_test. Това позволява да се

следи за претоварване (overfitting) на модела по време на тренировката и да се избягват неговите ефекти.

При изпълнение на този ред от кода моделът се тренира върху тренировъчните данни, като се извършват няколко епохи (50 в този случай), при което се използват пакети от 8 примера. Валидацията на модела се извършва върху тестовите данни X_test и y_test, които са били разделени предварително от данните за тренировка.

Епоха представлява една пълно обхождане на целия тренировъчен набор от данни от модела. Тоест, когато извършваме обучение с 100 епохи, това означава, че моделът ще бъде обучаван 100 пъти с всички данни в тренировъчния набор.

Размерът на batch-а определя колко примера от тренировъчния набор се използват за една итерация на обучението. Тоест, когато batch_size е 2, моделът ще обработва и ще обновява теглата си след всяка двойка примери от тренировъчния набор.

Изборът на оптимални стойности за епохите и размера на batch-а може да подобри производителността на модела, като постигне по-добри резултати в по-кратък период от време. Оптималните стойности могат да се определят чрез експериментиране с различни комбинации от параметри.

ж) за ред 28

28) test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)

Ред 28 използва метода evaluate() на модела за оценка на производителността на модела върху тестовите данни X_test и y_test. Методът връща загубата (loss) и точността (accuracy) на модела върху тестовите данни.

Аргументът verbose указва дали да се извежда информация по време на оценката. Ако verbose=0, няма да се извежда никаква информация. Ако verbose=1, ще се изведе прогресбар с информация за броя обработени примери и загубата и точността на модела. Ако verbose=2 (както в случая), ще се изведат само стойностите на загубата и точността.

Задача 1. Класификация на цветя от набора от данни Iris

Стартирайте https://colab.research.google.com/

С помощта на невронна мрежа, използвайки TensorFlow и Scikit-learn библиотеките в Python да се класифицират цветя от набора от данни Iris. Да се изчисли точността на модела и се класифицират нови данни за цветя. В програмата на задачата липсват части от кода, които трябва да въведете. Това са редове: 10, 41, 42 и 43 (1 точка).

1) *import tensorflow as tf*
2) *import numpy as np*
3) *from sklearn.datasets import load_iris*
4) *from sklearn.model_selection import train_test_split*
5) *from sklearn.preprocessing import LabelBinarizer*

6) *# Зареждане на данните*

```
iris = load_iris()
X, y = iris.data, iris.target

# Разделяне на данните за обучение и тест

# Преобразуване на маркировките към множество от еднокомпонентни маркери
lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_test = lb.transform(y_test)

# Създаване на модел
model = tf.keras.Sequential([
tf.keras.layers.Dense(64, activation='relu', input_shape=(4,)),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(3, activation='softmax')
])

# Компилиране на модела
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

# Обучение на модела
model.fit(X_train, y_train, epochs=50, batch_size=8, validation_data=(X_test, y_test))

# Оценка на модела
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print('\nТочност на модела:', test_acc)

# Генериране на произволни данни
new_data = np.array([[6.0, 3.1, 4.0, 1.3],
[5.1, 3.5, 1.4, 0.2],
[5.7, 2.9, 4.2, 1.3]])

# Класификация на новите данни
predictions = model.predict(new_data)
predicted_classes = np.argmax(predictions, axis=1)

# Преобразуване на класовете от числа към текст
class_names = ['setosa', 'versicolor', 'virginica']
predicted_classes_text = [class_names[i] for i in predicted_classes]

# Извеждане на резултатите
```

41)
42)
43)

Резултати:

Точността на модела: 0.9666666388511658

Нови данни:

[[6. 3.1 4. 1.3]

[5.1 3.5 1.4 0.2]

[5.7 2.9 4.2 1.3]]

Класове на новите данни (като числа):

[1 0 1]

Класове на новите данни (като текст):

['versicolor', 'setosa', 'versicolor']

5. Допълнителни пояснения към кода на задача 2

а) за ред 5

5) from sklearn.preprocessing import StandardScaler

StandardScaler е клас в модула sklearn.preprocessing, който се използва за стандартизиране на данните. Той работи като преобразува данните, като ги изваждате средното им и ги разделяте по стандартното им отклонение, така че новите данни да имат средно значение 0 и стандартно отклонение 1. Това е полезно при машинното обучение, защото може да помогне за подобряване на производителността и точността на моделите.

б) за ред 7

7) data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv", sep=";")

Тези данни представляват информация за свойствата на бели вина и тяхното качество. Те са извлечени от базата данни на UCI Machine Learning Repository. Файлът, който се зарежда, е във формат CSV и съдържа 11 колони с различни характеристики на виното и оценки за качеството му в интервала от 0 до 10.

в) за ред 11 и 12

11) X_train = train_data.iloc[:, :-1].values

12) y_train = train_data.iloc[:, -1].values

В тези редове се извличат данните, които ще се използват за обучение на модела.

X_train е множеството от примери за обучение, като те се извличат от train_data - това е пълното множество от данни за обучение. В този случай, X_train се състои от всички колони на train_data освен последната колона, защото това е колоната с целевата променлива, която трябва да се предскаже.

y_train е множеството от целеви променливи за обучението. Те се извличат от последната колона на train_data, защото тази колона съдържа истинските стойности на целевата променлива за всеки от обучаващите примери.

г) за ред 13 и 14

13) X_test = test_data.iloc[:, :-1].values

14) y_test = test_data.iloc[:, -1].values

Тези два реда отделят тестовите данни от използваните за обучение на модела. Първият ред (X_test = test_data.iloc[:, :-1].values) извлича всички колони от тестовия датасет, без последната, която съдържа целевата променлива. Той връща само стойностите на атрибутите на тестовия датасет като NumPy масив. Вторият ред (y_test = test_data.iloc[:, -1].values) извлича само целевата променлива от тестовия датасет и връща стойностите й като NumPy масив.

д) за ред 16, 17 и 18

16) scaler = StandardScaler()

17) X_train_scaled = scaler.fit_transform(X_train)

18) X_test_scaled = scaler.transform(X_test)

Тези редове служат за стандартизиране на данните в X_train и X_test, като им се прилага методът на StandardScaler. Този метод извършва скалиране на всяка функция (feature) в набора от данни, като изчислява средната стойност и стандартното отклонение на всяка функция и след това ги трансформира, за да имат средна стойност 0 и стандартно отклонение 1. Това помага да се избегнат възможни проблеми с небалансирани мащаби в данните и да се подобри производителността на модела.

е) от ред 20 до ред 25

20) model = tf.keras.models.Sequential([

21) tf.keras.layers.Dense(64, input_dim=11, activation='relu'),

22) tf.keras.layers.Dense(32, activation='relu'),

23) tf.keras.layers.Dense(16, activation='relu'),

24) tf.keras.layers.Dense(1)

25)])

Тези редове служат за създаване на невронна мрежа с помощта на Keras API на TensorFlow. Конкретно, те определят архитектурата на модела, като определят броя на скритите слоеве и броя на невроните в тези слоеве.

В този конкретен модел имаме 4 слоя - първият е входен слой, който получава входния набор от данни с размерност 11 (броя на характеристиките), следван от три скрити слоя с 64, 32 и 16 неврона съответно, като всякакъв слой освен последния има ReLU активация.

ReLU (Rectified Linear Unit) е функция за активация, която се използва често в невронните мрежи. Функцията приема входяща стойност и я прехвърля към изхода, като приема само положителните стойности, а негативните стойности заменя с 0. Формално, ReLU функцията е дефинирана като:

$f(x) = \max(0, x)$

където x е входящата стойност, а f(x) е изхода. Тази функция има някои полезни свойства като бързина на изчисление и способност да избегне проблема със затихващите градиенти (vanishing gradients problem), който може да възникне при използването на други функции за активация, като например сигмоидната функция.

Последният слой е изходен слой, който има един неврон и не задава активация, защото се търси регресионен модел, който предсказва едно непрекъснато числено значение.

ж) от ред 27 до ред 29

27) model.compile(loss='mean_squared_error',

28) optimizer='adam',

29) metrics=['mse'])

Тези редове са свързани с компилацията на модела.

Параметърът loss определя функцията за загуба (loss function), която моделът ще оптимизира по време на тренирането. В този случай се използва среднoквадратична грешка (mean squared error).

Параметърът optimizer определя алгоритъма за оптимизация, който ще се използва за обучение на модела. В този случай се използва Adam оптимизатор.

metrics указва метриките, които ще бъдат използвани за оценка на производителността на модела. В този случай се използва средно-квадратична грешка (mse).

з) за ред 31

31) model.fit(X_train_scaled, y_train, epochs=100, batch_size=2)

Този ред код представлява обучение на модела на набор от данни, като се извършва зададения брой епохи (epochs) и се използва зададен размер на батча (batch_size) за стохастичната градиентна оптимизация.

Епоха представлява една пълна обходка на целия тренировъчен набор от данни от модела. Тоест, когато извършваме обучение с 100 епохи, това означава, че моделът ще бъде обучаван 100 пъти с всички данни в тренировъчния набор.

Размерът на batch-а определя колко примера от тренировъчния набор се използват за една итерация на обучението. Тоест, когато batch_size е 2, моделът ще обработва и ще обновява теглата си след всяка двойка примери от тренировъчния набор.

Изборът на оптимални стойности за епохите и размера на batch-а може да подобри производителността на модела, като постигне по-добри резултати в по-кратък период от време. Оптималните стойности могат да се определят чрез експериментиране с различни комбинации от параметри.

Конкретно, методът "fit" на модела получава входни параметри X_train_scaled и y_train, които са стандартизирани входни данни и съответните им метки за класификация, и ги използва за обучение на модела за 100 епохи. Всяка епоха включва обработване на целия набор от данни и корекция на параметрите на модела посредством стохастичната градиентна оптимизация.

и) за ред 33 и 34

33) X_test_scaled = scaler.transform(X_test)

34) _, mse = model.evaluate(X_test_scaled, y_test, batch_size=2)

Ред 33 се отнася до стандартизирането на тестовите данни X_test с помощта на обекта на класа StandardScaler, който е бил използван преди за стандартизиране на тренировъчните данни.

Ред 34 използва метода evaluate() на обекта model за да изчисли средноквадратичната грешка (mean squared error - MSE) между предсказаните и реалните стойности на тестовите данни. В метода се подават стандартизираните тестови данни X_test_scaled и реалните стойности на тестовите метки y_test. Опцията batch_size указва размера на пакетите, в които да се извършват изчисленията, за да се оптимизира паметта и да се осигури по-бързо обучение на модела. В този случай размерът на пакета е 2. Резултатът на оценката на MSE се присвоява на променливата mse.

Задача 2. Предсказване на цени на вино

Стартирайте https://colab.research.google.com/

С помощта на невронна мрежа, използвайки TensorFlow и Scikit-learn библиотеките в Python да се предскаже цената на вино на базата нови данни [[7.1, 0.16, 0.28, 14.3, 0.045, 39, 111, 0.99146, 3.19, 0.64, 10.2], [6.9, 0.19, 0.35, 6.4, 0.044, 19, 96, 0.9949, 3.51, 0.43, 9.7]]. Да се изчисли точността на модела и средната класификационна грешка на модела. В програмата на задачата липсват части от кода, които трябва да въведете. Това са редове: 1, 2, 3, 37, 38, 40 и 41 (1 точка).

```
1)
2)
3)
4)  from sklearn.model_selection import train_test_split
5)  from sklearn.preprocessing import StandardScaler


6)  # Зареждане на данните
7)  data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-
    quality/winequality-white.csv", sep=";")


8)  # Разделяне на данните на тренировъчни и тестови
9)   train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)


10)   # Извличане на характеристиките и маркировките
11)   X_train = train_data.iloc[:, :-1].values
12)   y_train = train_data.iloc[:, -1].values


13)   X_test = test_data.iloc[:, :-1].values
14)   y_test = test_data.iloc[:, -1].values


15)   # Мащабиране на данните
16)   scaler = StandardScaler()
17)   X_train_scaled = scaler.fit_transform(X_train)
18)   X_test_scaled = scaler.transform(X_test)


19)   # Създаване на модел на невронната мрежа
20)   model = tf.keras.models.Sequential([
21)   tf.keras.layers.Dense(64, input_dim=11, activation='relu'),
22)   tf.keras.layers.Dense(32, activation='relu'),
23)   tf.keras.layers.Dense(16, activation='relu'),
24)   tf.keras.layers.Dense(1)
25)   ])


26)   # Компилиране на модела
27)   model.compile(loss='mean_squared_error',
```

```
28)  optimizer='adam',
29)  metrics=['mse'])


30)  # Трениране на модела
31)  model.fit(X_train_scaled, y_train, epochs=100, batch_size=2)


32)  # Оценка на точността на модела върху тестовите данни
33)  X_test_scaled = scaler.transform(X_test)
34)  _, mse = model.evaluate(X_test_scaled, y_test, batch_size=2)
35)  print('Средна квадратна грешка на модела: %.2f' % mse)


36)  # Използване на модела, за да предскажем нови данни
37)  new_data =
38)
39)  new_data_scaled = scaler.transform(new_data)
40)  predictions =
41)  print
```

Резултати:

Средна квадратна грешка на модела: 0.56

1/1 [==============================] - 0s 64ms/step

Предсказания: [[6.7766647]

[5.21663]]

Задача 3. Прогнозиране на цена на имот

Стартирайте https://colab.research.google.com/

В таблица са дадени цените на вилни имоти на базата на следните данни: квадратура, брой спални, брой бани, басейн и гараж. Да се определи прогнозната цена на вилен имот при квадратура 180 кв. м., брой спални 3, брой бани 2, без басейн и 2 гаража посредством невронна мрежа използвайки TensorFlow библиотеката. В програмата на задачата липсват части от кода, които трябва да въведете. Това са редове: 1, 2, 3, от 5 до 10, 12, от 14 до 21, 25, 28 и 30 (2 точки).

квадратура	брой спални	брой бани	басейн	гаража	цена
140	3	1	0	1	242000

160	3	2	0	2	277000
170	3	2	0	2	329000
187	3	2	0	2	369000
110	2	1	0	1	190000
155	4	2	0	2	250000
235	3	4	1	2	529900
245	4	4	1	2	599000
142	3	2	1	1	255000
170	3	3	0	2	242900

1)
2)
3)

4) # Създаване на модел на невронната мрежа
5)
6)
7)
8)
9)
10)

11) # Компилиране на модела
12)

13) # Задаване на обучаващите данни
14)
15)
16)
17)
18)
19)
20)
21)
22) X_train = df[['квадратура', 'брой спални', 'брой бани', 'басейн', 'гараж']].to_numpy()
23) y_train = df['цена'].to_numpy()

```
24)     # Трениране на модела – 1000 епохи, без да се извежда информация по
        време на оценката
25)

26)     # Предсказване на цена за нов имот
27)     new_data = np.array([[180, 3, 2, 0, 2]])
28)     predictions =

29)     # Извеждане на резултата
30)
```

Резултати:

Цената на новия имот е: 354205.03 долара като данните за имота са [180 3 2 0 2]