П1 - *Розробка моделей глибокого навчання з використанням Keras.*

Глибоке навчання стає все більш популярною підмножиною машинного навчання. Моделі глибокого навчання будуються з використанням нейронних мереж. Нейронна мережа приймає входи, які потім обробляються в прихованих шарах з використанням ваг, які коригуються під час навчання. Потім модель виконує прогнозування. Ваги коректуються для пошуку закономірностей, щоб зробити більш точний прогноз. Користувачеві не потрібно вказувати, які патерни шукати - нейронна мережа вчиться сама.

Keras - зручна нейромережева бібліотека, написана на мові Python. У цій лабораторній роботі ми спробуємо побудувати дві моделі глибокого навчання з використанням Keras: одна - для регресії, інша - для класифікації. Ми побудуємо регресійну модель для передбачення зарплати співробітника на годину, а також класифікаційну модель для передбачення, чи є у пацієнта цукровий діабет чи ні.

Примітка: набори даних, які ми будемо використовувати, відносно чисті, тому ми не будемо виконувати ніякої попередньої обробки даних для того, щоб наші дані були готові до моделювання. Набори даних, які ви будете використовувати в майбутніх проектах, можуть бути не такими чистими - наприклад, у них можуть бути відсутні значення - тому вам може знадобитися використовувати методи попередньої обробки даних, щоб змінити набори даних для отримання більш точних результатів.

Інструментарій

OS Windows, Linux, Macos

Anaconda https://www.anaconda.com/

Tensorflow https://www.tensorflow.org/install/pip

Keras https://pypi.org/project/Keras/

Spyder 4 https://docs.spyder-ide.org/current/installation.html

Pycharm https://www.jetbrains.com/

Install PyCharm and Anaconda (Windows /Mac/Ubuntu)
https://medium.com/@GalarnykMichael/setting-up-pycharm-with-anaconda-plus-installing-packages-windows-mac-db2b158bd8c

Регресійна модель

Зчитування даних

Для нашої регресійній моделі глибокого навчання першим кроком є зчитування даних, які ми будемо використовувати в якості вхідних. Для цього прикладу ми використовуємо набір даних "hourly wages". Для початку ми будемо використовувати Pandas для читання в даних.

"df" означає "датафрейм". Pandas зчитає csv файл як dataframe. Функція 'head()' покаже перші 5 рядків датафрейма, щоб ви могли перевірити, що дані були прочитані коректно, і могли спочатку поглянути на те, як дані структуровані.

```
Import pandas as pd


#read in data using pandas
train_df = pd.read_csv('./hourly_wages_data.csv')


#check data has been read in properly
train_df.head()
```

	wage_per_hour	union	education_yrs	experience_yrs	age	female	marr	south	manufacturing	construction
0	5.10	0	8	21	35	1	1	0	1	0
1	4.95	0	9	42	57	1	1	0	1	0
2	6.67	0	12	1	19	0	0	0	1	0
3	4.00	0	12	4	22	0	0	0	0	0
4	7.50	0	12	17	35	0	1	0	0	0

Розбиття набору даних на входи і виходи

Далі нам потрібно розділити наш набір даних на inputs (X) і нашу target (y). Нашими вхідними даними будуть всі стовпці, окрім 'pay_per_hour', тому що 'pay_per_hour' - це те, що ми будемо намагатися спрогнозувати. Тому 'wage_per_hour' буде нашим виходом.

Ми будемо використовувати функцію pandas 'drop', щоб видалити стовпець 'wage_per_hour' з нашого датафрейма і зберегти його в змінної 'train_X'. Це буде нашими входами.

```
# create a dataframe with all training data except the target column
train_X = train_df.drop(columns=['wage_per_hour'])

#check that the target variable has been removed
train_X.head()
```

	union	education_yrs	experience_yrs	age	female	marr	south	manufacturing	construction
0	0	8	21	35	1	1	0	1	0
1	0	9	42	57	1	1	0	1	0
2	0	12	1	19	0	0	0	1	0
3	0	12	4	22	0	0	0	0	0
4	0	12	17	35	0	1	0	0	0

Ми вставимо стовпець 'wage_per_hour' в нашу цільову змінну (y).

	wage_per_hour
0	5.10
1	4.95
2	6.67
3	4.00
4	7.50

Створення моделі

Далі, ми повинні побудувати модель. Ось код:

```
from keras.models import Sequential
from keras.layers import Dense

#create model
model = Sequential()
```

```
#get number of columns in training data
n_cols = train_X.shape[1]


#add model layers
model.add(Dense(10, activation='relu',
input_shape=(n_cols,)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1))
```

Тип моделі, яку ми будемо використовувати - Sequential. Sequential - це найпростіший спосіб побудувати модель в Keras. Він дозволяє побудувати модель шар за шаром. Кожен шар має ваги, які відповідають наступним за ним шару.

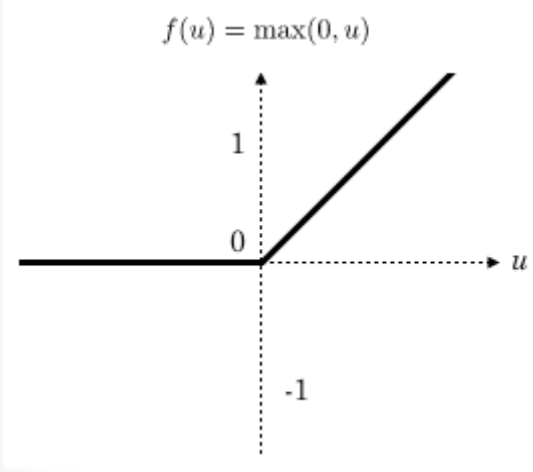
Ми використовуємо функцію 'add ()' для додавання шарів до нашої моделі. Ми додамо два шари і вихідний шар.

Dense ()- це тип шару. Dense - стандартний тип шару, який працює в більшості випадків. У щільному шарі все вузли попереднього шару з'єднуються з вузлами поточного шару.

У нас є 10 вузлів в кожному з вхідних шарів. Це число також може бути в сотнях або тисячах. Збільшення кількості вузлів в кожному шарі збільшує ємність моделі.

“Activation"- це функція активації для шару. Функція активації дозволяє моделям враховувати нелінійні зв'язки. Наприклад, якщо ви передбачаєте цукровий діабет у пацієнтів, вік від 10 до 11 років відрізняється від віку від 60-61 року.

Функція активації, яку ми будемо використовувати, - це ReLU або Rectified Linear Activation. Незважаючи на те, що це дві лінійні частини, було доведено, що вона добре працює в нейронних мережах.



ReLU Activation Function

Більше про активаційні функції можна подивитися тут https://keras.io/api/layers/activation_layers/

Першому шару потрібен input shape. Input shape визначає кількість рядків і стовпців на вході. Кількість стовпців у вхідній формі зберігається в 'n_cols'. Після коми немає нічого, що вказувало б на те, що може бути будь-яку кількість рядків.

Останній шар - вихідний шар. У нього є тільки один вузол, який призначений для нашого прогнозування.

Компіляція моделі

Далі, нам потрібно зібрати нашу модель. Компіляція моделі приймає два параметри: оптимізатор і функцію втрат.

Існуючі оптимізатори можна знайти тут https://keras.io/api/optimizers/

а функції втрат https://keras.io/api/losses/ .

Оптимізатор контролює швидкість навчання. Як оптимізатор (алгоритм навчання) ми будемо використовувати 'adam'. Adam, як правило, є хорошим оптимізатором для багатьох випадків. Оптимізатор Adam регулює швидкість навчання протягом всього тренінгу.

Швидкість навчання визначає, як швидко розраховується оптимальна вага моделі. Менша швидкість навчання може привести до більш точного зважуванню (до певного моменту), але час, необхідний для розрахунку ваг, буде більше.

Для функції втрат ми будемо використовувати 'mean_squared_error'. Вона обчислюється шляхом взяття середньої квадратної різниці між прогнозованим і фактичним значеннями. Це популярна функція втрат для регресійних задач. Чим ближче до 0, тим краще виконується модель.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

де N – довжина вибірки, f_i - значення яке повертає модель, y_i - актуальне значення в точці *i*

```
#compile model using mse as a measure of model
performance

model.compile(optimizer='adam',
loss='mean_squared_error')
```

Навчання моделі

Тепер ми будемо тренувати нашу модель. Для навчання ми будемо використовувати функцію 'fit ()' на нашій моделі з наступними п'ятьма параметрами: тренувальні дані (train_X), цільові дані (train_y), валідаційні спліт, кількість епох і зворотних викликів.

Поділ валідації випадковим чином розділить дані для використання в тренінгу і тестування. Під час навчання ми зможемо побачити втрати при валідації, які дають середню квадратну помилку нашої моделі на валідаційні множині. Ми встановимо поділ валідації на 0,2, що означає, що 20% систему адаптації, які ми надаємо в моделі, будуть зарезервовані для тестування продуктивності моделі.

Кількість епох - це кількість разів, коли модель буде проходити через дані. Чим більше епох ми запустимо, тим більше модель буде поліпшуватися, до певної точки. Після цієї точки, модель перестане поліпшуватися протягом кожної епохи. Крім того, чим більше епох, тим довше модель буде працювати. Для контролю за цим ми будемо використовувати "early stopping".

Рання зупинка зупинить модель від тренувань до досягнення кількості епох, якщо модель перестане поліпшуватися. Ми встановимо наш монітор ранньої зупинки на 3. Це означає, що після 3 епох поспіль, в яких модель не поліпшується, тренування припиняться. Іноді втрата валідації може перестати поліпшуватися, а потім покращитися в наступну епоху, але після

3 епох, в яких втрата валідації не покращується, вона зазвичай не покращиться знову.

```
from keras.callbacks import EarlyStopping

#set early stopping monitor so the model stops training
when # it won't improve anymore


early_stopping_monitor = EarlyStopping(patience=3)


#train model

model.fit(train_X, train_y, validation_split=0.2,
epochs=30, callbacks=[early_stopping_monitor])
```

```
Train on 427 samples, validate on 107 samples
Epoch 1/30
427/427 [==============================] - 3s 7ms/step - loss: 200.7186 - val_loss: 243.7071
Epoch 2/30
427/427 [==============================] - 0s 89us/step - loss: 130.2738 - val_loss: 172.7188
Epoch 3/30
427/427 [==============================] - 0s 97us/step - loss: 86.9109 - val_loss: 123.9527
Epoch 4/30
427/427 [==============================] - 0s 97us/step - loss: 58.5707 - val_loss: 91.3879
Epoch 5/30
427/427 [==============================] - 0s 98us/step - loss: 41.0577 - val_loss: 68.7715
Epoch 6/30
427/427 [==============================] - 0s 97us/step - loss: 30.9307 - val_loss: 53.7192
Epoch 7/30
427/427 [==============================] - 0s 98us/step - loss: 25.0882 - val_loss: 44.5493
Epoch 8/30
427/427 [==============================] - 0s 97us/step - loss: 22.1689 - val_loss: 38.9246
Epoch 9/30
427/427 [==============================] - 0s 98us/step - loss: 20.8058 - val_loss: 35.9207
Epoch 10/30
427/427 [==============================] - 0s 99us/step - loss: 20.3621 - val_loss: 34.2123
Epoch 11/30
427/427 [==============================] - 0s 101us/step - loss: 20.1909 - val_loss: 33.4018
Epoch 12/30
427/427 [==============================] - 0s 94us/step - loss: 20.1781 - val_loss: 32.6290
Epoch 13/30
427/427 [==============================] - 0s 98us/step - loss: 20.1422 - val_loss: 32.8353
Epoch 14/30
427/427 [==============================] - 0s 91us/step - loss: 20.1181 - val_loss: 32.6664
Epoch 15/30
427/427 [==============================] - 0s 98us/step - loss: 20.0899 - val loss: 33.2220
```

Прогнозування нових даних

Якщо ви хочете використовувати цю модель для прогнозування на нових даних, то ми будемо використовувати функцію 'predict ()', передаючи в наших нових даних. Виходом будуть передбачення 'pay_per_hour'.

```
#example on how to use our newly trained model on how to
make #predictions on unseen data (we will pretend our new
data is #saved in a dataframe called 'test_X').
```

```
test_y_predictions = model.predict(test_X)
```

Вітаю! Ви побудували модель глибокого навчання в Keras! Поки вона не дуже точна, але це може бути покращено за рахунок використання більшої кількості тренувальних даних і "model capacity".

Потужність моделі (model capacity)

У міру збільшення кількості вузлів і шарів в моделі збільшується ємність моделі. Збільшення ємності моделі може привести до більш точної моделі, аж до певного моменту, коли модель перестане поліпшуватися. Як правило, чим більше даних з навчання, тим більше повинна бути модель. Ми використовуємо тільки невелику кількість даних, тому наша модель досить мала. Чим більше модель, тим більше обчислювальних можливостей вона вимагає і навчання займе більше часу.

Давайте створимо нову модель, використовуючи ті ж навчальні дані, що і наша попередня модель. На цей раз ми додамо шар і збільшимо вузли в кожному шарі до 200. Ми будемо навчати модель, щоб переконатися, що збільшення потужності моделі поліпшить наш результат валідації.

```
#training a new model on the same data to show the effect
of #increasing model capacity
```

```
#create model
model_mc = Sequential()
```

```
#add model layers
model_mc.add(Dense(200, activation='relu',
input_shape=(n_cols,)))
model_mc.add(Dense(200, activation='relu'))
model_mc.add(Dense(200, activation='relu'))
```

```
model_mc.add(Dense(1))


#compile model using mse as a measure of model
performance

model_mc.compile(optimizer='adam',
loss='mean_squared_error')


#train model

model_mc.fit(train_X, train_y, validation_split=0.2,
epochs=30, callbacks=[early_stopping_monitor])
```

```
Train on 427 samples, validate on 107 samples
Epoch 1/30
427/427 [==============================] - 3s 7ms/step - loss: 39.2107 - val_loss: 40.3353
Epoch 2/30
427/427 [==============================] - 0s 226us/step - loss: 22.6596 - val_loss: 43.5961
Epoch 3/30
427/427 [==============================] - 0s 230us/step - loss: 21.1501 - val_loss: 35.9529
Epoch 4/30
427/427 [==============================] - 0s 222us/step - loss: 20.0189 - val_loss: 34.9528
Epoch 5/30
427/427 [==============================] - 0s 220us/step - loss: 19.5254 - val_loss: 33.4263
Epoch 6/30
427/427 [==============================] - 0s 221us/step - loss: 19.1911 - val_loss: 29.8092
Epoch 7/30
427/427 [==============================] - 0s 218us/step - loss: 18.8980 - val_loss: 28.0586
Epoch 8/30
427/427 [==============================] - 0s 210us/step - loss: 21.4672 - val_loss: 28.4561
Epoch 9/30
427/427 [==============================] - 0s 188us/step - loss: 22.2891 - val_loss: 28.5794
Epoch 10/30
427/427 [==============================] - 0s 189us/step - loss: 19.4059 - val_loss: 32.4649
```

Ми бачимо, що збільшивши потужність нашої моделі, ми поліпшили нашу валідаційну оцінку з 32.63 в старій моделі до 28.06 в новій.

Класифікаційна модель

Тепер давайте перейдемо до побудови нашої моделі для класифікації. Оскільки багато кроків будуть повторюватися з попередньою моделлю, опис буде тільки нової концепції.

Цією моделлю ми будемо передбачати, чи є у пацієнтів цукровий діабет чи ні.

```
#read in training data

train_df_2 =
pd.read_csv('documents/data/diabetes_data.csv')


#view data structure

train_df_2.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
#create a dataframe with all training data except the
target #column

train_X_2 = df_2.drop(columns=['diabetes'])


#check that the target variable has been removed

train_X_2.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

При поділі цільового стовпчика необхідно викликати функцію 'to_categorical ()', щоб стовпець був "one-hot encoded". В даний час пацієнт без цукрового діабету представлений в стовпці "0", а пацієнт з цукровим діабетом - в стовпці "1". При однорозрядному кодуванні, ціле число буде видалено, і для кожної категорії буде введена двійкова змінна. У нашому випадку у нас

є дві категорії: немає цукрового діабету і цукровий діабет. Пацієнт без цукрового діабету буде представлений [1 0], а пацієнт з цукровим діабетом - [0 1].

```
from keras.utils import to_categorical

#one-hot encode target column
train_y_2 = to_categorical(df_2.diabetes)

#vcheck that target column has been converted
train_y_2[0:5]
```

```
array([[ 0.,  1.],
       [ 1.,  0.],
       [ 0.,  1.],
       [ 1.,  0.],
       [ 0.,  1.]], dtype=float32)
```

```
#create model
model_2 = Sequential()

#get number of columns in training data
n_cols_2 = train_X_2.shape[1]

#add layers to model
model_2.add(Dense(250, activation='relu',
input_shape=(n_cols_2,)))
model_2.add(Dense(250, activation='relu'))
model_2.add(Dense(250, activation='relu'))
model_2.add(Dense(2, activation='softmax'))
```

Останній шар нашої моделі має 2 вузла - по одному для кожного варіанту: у пацієнта цукровий діабет чи ні.

Активація - "softmax". Softmax робить вихідну суму рівній 1, тому вихід може бути інтерпретований як ймовірність. Потім модель зробить свій прогноз, виходячи з того, яка опція має більш високу ймовірність.

```
#compile model using accuracy to measure model performance

model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Для функції втрат ми будемо використовувати 'categorical_crossentropy'. Це найбільш поширений вибір для класифікації. Більш низька оцінка говорить про те, що модель працює краще.

Щоб ще більше спростити інтерпретацію, ми будемо використовувати метрику "accuracy", щоб побачити оцінку точності на валідаційній множині в кінці кожної епохи.

Більше про метрики можна знайти тут https://keras.io/api/metrics/ .

```
#train model

model_2.fit(X_2, target, epochs=30, validation_split=0.2, callbacks=[early_stopping_monitor])
```

```
Train on 614 samples, validate on 154 samples
Epoch 1/30
614/614 [==============================] - 0s 240us/step - loss: 0.6972 - acc: 0.6612 - val_loss: 0.7372 - val_acc:
0.6299
Epoch 2/30
614/614 [==============================] - 0s 265us/step - loss: 0.6634 - acc: 0.6547 - val_loss: 0.7887 - val_acc:
0.6688
Epoch 3/30
614/614 [==============================] - 0s 308us/step - loss: 0.6772 - acc: 0.6889 - val_loss: 0.6767 - val_acc:
0.6623
Epoch 4/30
614/614 [==============================] - 0s 267us/step - loss: 0.6745 - acc: 0.6938 - val_loss: 0.8132 - val_acc:
0.5455
Epoch 5/30
614/614 [==============================] - 0s 228us/step - loss: 0.6251 - acc: 0.6922 - val_loss: 0.6641 - val_acc:
0.6299
Epoch 6/30
614/614 [==============================] - 0s 235us/step - loss: 0.5660 - acc: 0.7085 - val_loss: 0.6837 - val_acc:
0.5779
Epoch 7/30
614/614 [==============================] - 0s 256us/step - loss: 0.6009 - acc: 0.6922 - val_loss: 0.5935 - val_acc:
0.7013
Epoch 8/30
614/614 [==============================] - 0s 238us/step - loss: 0.5647 - acc: 0.7166 - val_loss: 0.6208 - val_acc:
0.6753
Epoch 9/30
614/614 [==============================] - 0s 228us/step - loss: 0.5514 - acc: 0.7068 - val_loss: 0.6295 - val_acc:
0.6558
Epoch 10/30
614/614 [==============================] - 0s 229us/step - loss: 0.5920 - acc: 0.7068 - val_loss: 0.6410 - val_acc:
0.6558
```

Вітаю! Тепер ви на правильному шляху до створення дивовижних моделей для глибокого вивчення в Keras!

Завдання 1

1. Використовуючи датасет hourly_wages_data.csv створити модель на базі глибокої нейронної мережі для прогнозування даних.

2. Розбити датасет на 3 підвибірки: тренувальний (70%), валідаційний (20%), тестовий (10%), використовучі власну функцію, або функції які реалізовані в pandas чи sklearn.

3. Дослідити як впливає скейлінг і нормалізація даних на результат моделі https://scikit-learn.org/stable/modules/preprocessing.html

4. Імплементувати модель в Keras за допомогою Sequentinal(). Обґрунтувати обрання кількості нейронів та шарів. Провести дослідження, як буде мінятися точність мережі при різних гіперпараметрах.

5. В якості оптимізатора застосувати декілька алгоритмів навчання (https://keras.io/api/optimizers/):

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam

6. Використати різні активаційні функції і виявити вплив виду функції та точність моделі.

7. Побудувати графіки навчання по кожному оптимізатору і порівняти швидкість збіжності алгоритмів.

8. Провести тестування мережі. Визначити чи не було перенавчання мережі.

Завдання 2

1. Використовуючи датасет hourly_wages_data.csv створити модель на базі глибокої нейронної мережі для класифікування даних.

2. Розбити датасет на 3 підвибірки: тренувальний (70%), валідаційний (20%), тестовий (10%), використовучі власну функцію, або функції які реалізовані в pandas чи sklearn.

3. Дослідити як впливає скейлінг і нормалізація даних на результат моделі https://scikit-learn.org/stable/modules/preprocessing.html

4. Імплементувати модель в Keras за допомогою Sequentinal(). Обгрунтувати обрання кількості нейронів та шарів. Провести дослідження, як буде мінятися точність мережі при різних гіперпараметрах.

5. В якості оптимізатора застосувати декілька алгоритмів навчання (https://keras.io/api/optimizers/):

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam

6. Використати різні активаційні функції і виявити вплив виду функції та точність моделі.

7. Побудувати графіки навчання по кожному оптимізатору і порівняти швидкість збіжності алгоритмів.

8. Провести тестування мережі. Визначити чи не було перенавчання мережі.

Завдання 3

1. Обрати дані з https://github.com/plotly/datasets

2. Побудувати модель глибокого навчання для задачі регресії чи задачі класифікації (в залежності який датасет буде обрано).

3. Провести компіляцію, навчання та тестування моделі.

4. Візуалізувати результати.