Laboratorium: Rekurencyjne sieci neuronowe

June 13, 2023

1 Zakres ćwiczeń

Dzisiejsze laboratorium poświęcone jest rekurencyjnym sieciom neuronowym (RNN), które często stosujemy do przetwarzania danych w postaci sekwencji, takich jak szeregi czasowe, tekst czy dźwięk. Ich zaletą jest możliwość przetwarzania sekwencji o dowolnej (i zmiennej) długości.

Przedmiotem ćwiczeń będzie jedno z klasycznych zastosowań RNN – predykcja szeregów czasowych. Skorzystamy w tym celu z danych miejskiego systemu *bike sharing* w Waszyngtonie – Capital Bikeshare. Dane te dostępne są jako zbiór w repozytorium UCI.

2 Ćwiczenia

2.1 Pobieranie danych

Twój skrypt powinien pobierać dane bezpośrednio z repozytorium UCI. Możesz w tym celu skorzystać z pomocniczej funkcji *get_file*:

```
tf.keras.utils.get_file(
    "bike_sharing_dataset.zip",
    "https://archive.ics.uci.edu/static/public/275/bike+sharing+dataset.zip",
    cache_dir=".",
    extract=True
)
```

```
Downloading data from
https://archive.ics.uci.edu/static/public/275/bike+sharing+dataset.zip
   8192/Unknown - 0s 0us/step
```

```
'./datasets/bike_sharing_dataset.zip'
```

Plik zostanie pobrany do katalogu `./datasets`, a następnie rozpakowany. Archiwum zawiera dwa zbiory, `day.csv` (statystyki dzienne) oraz `hour.csv` (statystyki godzinowe). W ćwiczeniach będziemy korzystać z tego drugiego zbioru.

2.2 Przygotowanie danych

Zapoznaj się z opisem zbioru danych na tej stronie, szczególnie z opisem atrybutów i ich wartości.

Wczytaj zbiór danych do DataFrame. Odpowiednie atrybuty funkcji `read_csv` pozwolą na automatyczne połączenie pól reprezentujących datę oraz godzinę, a także na ustawienie powstałego

pola jako indeksu:

```
df = pd.read_csv('datasets/hour.csv',
                 parse_dates={'datetime': ['dteday', 'hr']},
                 date_format='%Y-%m-%d %H',
                 index_col='datetime'
                )
```

Sprawdź zakres znaczników czasowych w zbiorze:

```
print((df.index.min(), df.index.max()))
```

```
(Timestamp('2011-01-01 00:00:00'), Timestamp('2012-12-31 23:00:00'))
```

Sprawdź, czy mamy rekordy dla każdego znacznika czasowego – 2012 był rokiem przestępnym więc spodziewamy się $(365 + 366) * 24 = 17544$ rekordów:

```
(365 + 366) * 24 - len(df)
```

```
165
```

Okazuje się, że w zbiorze brakuje rekordów dla okresów (godzin) podczas których nikt nie korzystał z rowerów. Aby szeregi czasowe były regularne, trzeba je uzupełnić. Przy okazji pozbędziemy się niepotrzebnych kolumn.

Przeprowadź *resampling* zbioru danych do częstotliwości godzinowej i zastosuj do przydatnych kolumn odpowiednie strategie uzupełniania danych, tak aby zachować semantykę atrybutów:

- dla kolumn przechowujących zarejestrowane liczby wypożyczeń (`casual`, `registered`, `cnt`), wypełnij brakujące wiersze zerami,
- dla kolumn przechowujących sensoryczne dane pogodowe (`temp`, `atemp`, `hum`, `windspeed`), zastosuj interpolację,
- dla kolumn kategoryzowanych (`holiday`, `weekday`, `workingday`, `weathersit`), wypełnij brakujące wartości z poprzedniego rekordu.

Jednym ze sposów na osiągnięcie tego celu jest utworzenie obiektu `Resampler` przy pomocy funkcji `resample()`, następnie użycie go z odpowiednimi funkcjami (`ffill()`, `asfreq()`, `interpolate()`), pobranie z powstałych DataFrame odpowiednich kolumn i ponowne połączenie w jeden obiekt (`pd.concat(..., axis=1)`).

Sprawdź, czy DataFrame ma odpowiednią strukturę i czy nie zawiera brakujących wartości (które stanowiłyby problem w procesie uczenia):

```
df.notna().sum()
```

```
casual        17544
registered    17544
cnt           17544
temp          17544
atemp         17544
hum           17544
```

```
windspeed     17544
holiday       17544
weekday       17544
workingday    17544
weathersit    17544
dtype: int64
```

Kolumny z danymi sensorycznymi są już znormalizowane do zakresu $[0, 1]$, podobnie jak kolumny „logiczne” (0/1). Przeprowadźmy odpowiednią normalizację kolumn z liczbą wypożyczeń oraz sytuacji pogodowej.

```
df[['casual', 'registered', 'cnt', 'weathersit']].describe()
```

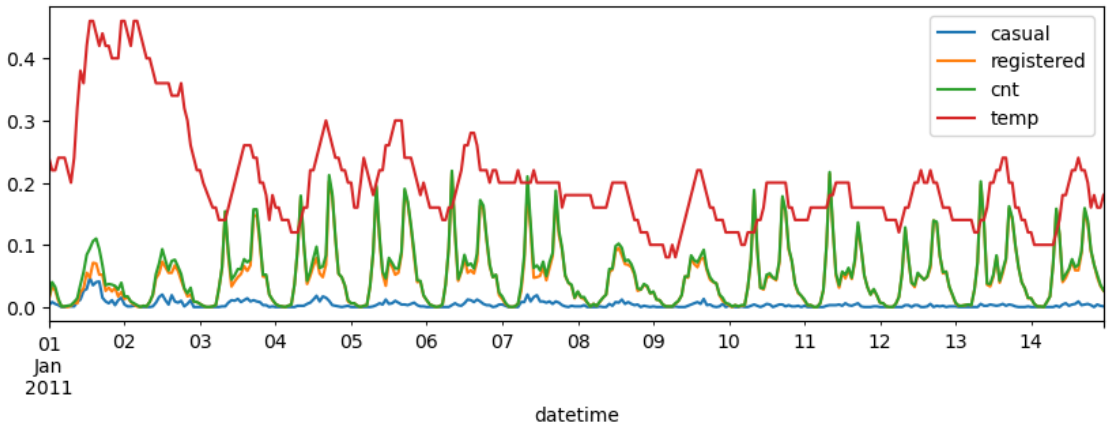
	casual	registered	cnt	weathersit
count	17544.000000	17544.000000	17544.000000	17544.000000
mean	35.340686	152.340515	187.681202	1.434223
std	49.193293	151.373409	181.456478	0.648339
min	0.000000	0.000000	0.000000	1.000000
25%	4.000000	32.000000	38.000000	1.000000
50%	16.000000	114.000000	140.000000	1.000000
75%	48.000000	219.000000	279.000000	2.000000
max	367.000000	886.000000	977.000000	4.000000

```
df.casual /= 1e3
df.registered /= 1e3
df.cnt /= 1e3
df.weathersit /= 4
```

Zawsze dobrym pomysłem jest wizualizacja danych, np.:

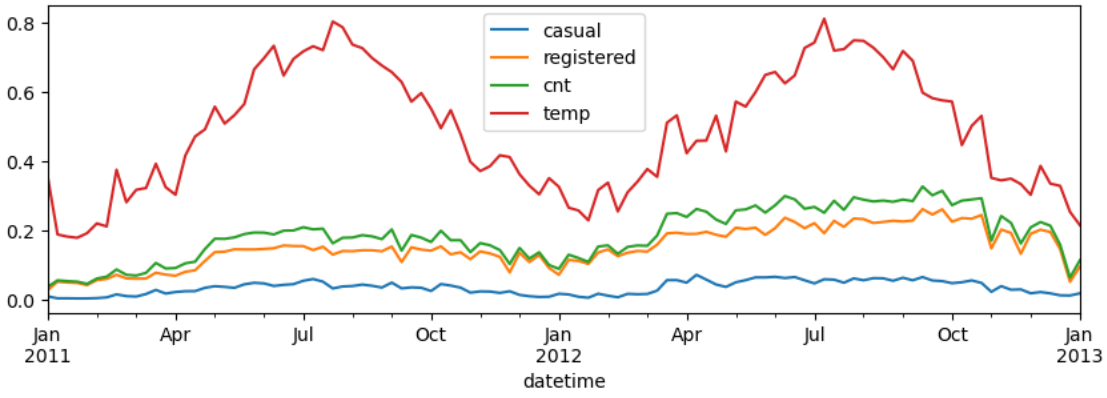
```
df_2weeks = df[:24 * 7 * 2]
df_2weeks[['casual', 'registered', 'cnt', 'temp']].plot(figsize=(10, 3))
```

```
<Axes: xlabel='datetime'>
```



```
df_daily = df.resample('W').mean()
df_daily[['casual', 'registered', 'cnt', 'temp']].plot(figsize=(10, 3))
```

```
<Axes: xlabel='datetime'>
```



2.3 Wskaźniki bazowe

Przy problemach uczenia maszynowego niezwykle ważne są wskaźniki bazowe, które pozwalają na faktyczną ocenę skuteczności modelu.

Określmy „naiwne" wskaźniki dla naszego problemu. Ponieważ dostrzegamy w zbiorze cykle dobowe oraz tygodniowe, obliczmny średni błąd bezwzględny gdyby predykcja polegała po prostu na skopiowaniu poprzedniej doby lub poprzedniego tygodnia. Będziemy dokonywali predykcji sumarycznej liczby wypożyczeń (kolumna `cnt`).

Oblicz te wartości i zapisz jako krotkę `(mae_daily, mae_weekly)` do pliku `mae_baseline.pkl`. Ponieważ wcześniej podzieliliśmy wartości przez stałą 10^3, pamiętaj o ich skalowaniu z powrotem. Błąd powinien wynosić kilkadziesiąt (50–70) wypożyczeń.

2 punkty

2.4 Predykcja przy pomocy sieci gęstej

Pierwszym eksperymentem będzie predykcja przy pomocy sieci gęstej. Przyjmijmy okno (pamięć) o długości 24 rekordów (1 doba).

Pierwszym krokiem jest przygotowanie zbioru danych. Ponieważ zbiór danych obejmuje dwa lata, podzielmy go w stosunku 18 miesięcy / 6 miesięcy na zbiór uczący i walidacyjny.

```
cnt_train = df['cnt']['2011-01-01 00:00':'2012-06-30 23:00']
cnt_valid = df['cnt']['2012-07-01 00:00':]
```

Następnie skorzystamy z funkcji `dataset_from_array` do utworzenia zbiorów danych TensorFlow:

```
seq_len = 1 * 24
train_ds = tf.keras.utils.timeseries_dataset_from_array(
    cnt_train.to_numpy(),
    targets=cnt_train[seq_len:],
    sequence_length=seq_len,
    batch_size=32,
    shuffle=True,
    seed=42
)
valid_ds = tf.keras.utils.timeseries_dataset_from_array(
    cnt_valid.to_numpy(),
    targets=cnt_valid[seq_len:],
    sequence_length=seq_len,
    batch_size=32
)
```

```
Metal device set to: Apple M2
```

Utwórz model zawierający jedną warstwę gęstą z jednym neuronem, o wejściu równym szerokości okna:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_shape=[seq_len])
])
```

Skompiluj model i przeprowadź uczenie, gromadząc metrykę średniego błędu bezwzględnego (`mae`). Dobierz parametry tak, aby wynik uczenia był jak najlepszy.

We wszystkich ćwiczeniach ucz sieć przez 20 epok aby zapewnić rozsądny czas przy ocenianiu modelu.

Podpowiedź: Przy pracy z szeregami czasowymi dobrze sprawdza się optymalizator SGD z pędem (*momentum*) o wartości np. 0.9. Dobierz krok uczenia tak, aby proces był jak najbardziej efektywny przy zadanej liczbie epok. Warto też skorzystać z funkcji straty Hubera, gdyż zapewnia szybką zbieżność, ale nie jest tak wrażliwa na wartości odstające (*outliers*) jak np. MSE.

Zapisz przyuczony model w pliku `model_linear.h5`.

Oblicz uzyskaną wartość MAE dla zbioru walidacyjnego jako krotkę 1-elementową `(mae_linear)` w pliku `mae_linear.pkl`.

4 p.

2.5 Prosta sieć rekurencyjna

Utwórz prostą sieć rekurencyjną, zawierającą jedną warstwę z jednym neuronem:

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(1, input_shape=[None, 1])
])
```

Powtórz procedurę z poprzedniego ćwiczenia. Pamiętaj, że optymalny krok uczenia może być tym razem inny.

Zapisz przyuczony model w pliku `model_rnn1.h5`.

3 p.

Oblicz uzyskaną wartość MAE dla zbioru walidacyjnego jako krotkę 1-elementową `(mae_rnn1)` w pliku `mae_rnn1.pkl`.

1 p.

Wynik nie jest doskonały. Rozbuduj model poprzez zwiększenie liczby neuronów w warstwie rekurencyjnej do 32 oraz dodanie warstwy gęstej z 1 neuronem (jako warstwy wyjściowej).

Powtórz procedurę uczenia dla nowego modelu. Zapisz przyuczony model w pliku `model_rnn32.h5`.

3 p.

Oblicz uzyskaną wartość MAE dla zbioru walidacyjnego jako krotkę 1-elementową `(mae_rnn32)` w pliku `mae_rnn32.pkl`.

1 p.

2.6 Głęboka RNN

Rozbuduj model tak, aby zawierał kilka (np. 3) warstwy rekurencyjne. Pamiętaj o tym, aby wszystkie warstwy RNN oprócz ostatniej przekazywały dalej sekwencje (argument `return_sequences`).

Przeprowadź procedurę uczenia dla nowego modelu (20 epok). Zapisz przyuczony model w pliku `model_rnn_deep.h5`.

3 p.

Oblicz uzyskaną wartość MAE dla zbioru walidacyjnego jako krotkę 1-elementową `(mae_rnn_deep)` w pliku `mae_rnn_deep.pkl`. Pamiętaj o skalowaniu wartości.

1 p.

2.7 Model wielowymiarowy

Dotychczas dokonywaliśmny predykcji liczby wypożyczeń wyłącznie na podstawie poprzednich wartości tego parametru. Ale przecież nasz zbiór danych zawiera też inne parametry, które mogą wpływać na intensywność korzystania z rowerów miejskich.

Przygotuj zbiór danych dla sieci wielowariantowej, który w zbiorze cech, oprócz liczby wypożyczeń, będzie zawierał również:

- sytuację pogodową,
- temperaturę odczuwalną,
- informację, czy dzień jest wolny czy roboczy.

Podziel zbiór tak jak poprzednio (18/6 miesięcy) i przygotuj zbiory danych przy pomocy `timeseries_dataset_from_array`.

Utwórz model zawerający jedną warstwę RNN z 32 neuronami, ale dostosowaną do nowego kształtu danych.

Przeprowadź procedurę uczenia dla nowego modelu przez 20 epok. Zapisz przyuczony model w pliku `model_rnn_mv.h5`.

3 p.

Oblicz uzyskaną wartość MAE dla zbioru walidacyjnego jako krotkę 1-elementową `(mae_rnn_mv)` w pliku `mae_rnn_mv.pkl`. Pamiętaj o przeskalowaniu wartości z powrotem do rzeczywistych jednostek.

1 p.

3 Wyślij rozwiązanie

Skrypt realizujący powyższe punkty zapisz w pliku `lab13/lab13.py` w swoim repozytorium.