Uniwersytet Bielsko-Bialski

Algorytm LSTM dla tekstu

Matematyka Konkretna

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Autor:

Bartosz Bieniek 058085

1. Cel ćwiczenia

Opracować sieć LSTM w celu nauczenia się tekstu z dokładnością 0.1
Wariant 1: "AI research has tried and discarded many different approaches, including simulating the brain, modeling human problem solving, formal logic, large databases of knowledge, and imitating animal behavior"
Zadania umieścić na Github.

2. Przebieg ćwiczenia

1. Import bibliotek
Zaimportowano niezbędne biblioteki do implementacji modelu LSTM w środowisku TensorFlow. Wczytano moduły do tworzenia modelu sekwencyjnego (Sequential) oraz warstw modelu, takich jak Embedding, LSTM i Dense. Dodatkowo zaimportowano funkcję to_categorical do kodowania danych wyjściowych, optymalizator Adam oraz mechanizm wczesnego zatrzymania (EarlyStopping) w celu kontroli procesu treningowego.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

Rys. 1. Import niezbędnych bibliotek

2. Definicja podstawowych parametrów
Zdefiniowano tekst źródłowy, który posłużył jako dane treningowe dla modelu. Na jego podstawie utworzono zestaw unikalnych znaków oraz skonstruowano słowniki mapujące znaki na odpowiadające im indeksy (char2int) oraz indeksy na znaki (int2char). Dodatkowo ustalono długość sekwencji wejściowej, która wynosiła 40 znaków.

```python
# Tekst źródłowy do nauki
input_text = (
    "AI research has tried and discarded many different approaches, including "
    "simulating the brain, modeling human problem solving, formal logic, "
    "large databases of knowledge, and imitating animal behavior"
)

# Przygotowanie mapowania znaków
unique_chars = sorted(set(input_text))
char2int = {ch: idx for idx, ch in enumerate(unique_chars)}
int2char = {idx: ch for ch, idx in char2int.items()}

# Parametry sekwencji
sequence_len = 40
```

Rys. 2. Definicja podstawowych parametrów.

3. Generowanie danych treningowych

Wygenerowano dane treningowe na podstawie tekstu źródłowego, dzieląc go na nakładające się sekwencje o długości 40 znaków. Dla każdej sekwencji wyznaczono kolejny znak jako oczekiwaną odpowiedź modelu. Następnie dokonano konwersji sekwencji znaków na odpowiadające im indeksy całkowite zgodnie z wcześniej zdefiniowanym słownikiem char2int. Dane wejściowe zapisano jako tablicę NumPy, natomiast dane wyjściowe zakodowano w formacie one-hot przy użyciu funkcji to_categorical, dostosowując je do liczby unikalnych znaków w tekście.

```python
# Generowanie danych wejściowych i oczekiwanych wyjść
sequences = []
next_chars = []

for i in range(len(input_text) - sequence_len):
    seq = input_text[i:i + sequence_len]
    next_c = input_text[i + sequence_len]
    sequences.append([char2int[c] for c in seq])
    next_chars.append(char2int[next_c])

X = np.array(sequences)
y = to_categorical(next_chars, num_classes=len(unique_chars))
```

Rys. 3. Generowanie danych treningowych

4. Budowanie sieci neuronowej

Zbudowano model sieci neuronowej w architekturze sekwencyjnej. Na wejściu zastosowano warstwę osadzającą (Embedding), przekształcającą indeksy znaków na wektory o wymiarze 32. Następnie dodano warstwę rekurencyjną typu LSTM z 128 jednostkami ukrytymi, umożliwiającą modelowanie zależności sekwencyjnych w danych tekstowych. Na wyjściu umieszczono warstwę gęstą (Dense) z funkcją aktywacji softmax, generującą rozkład prawdopodobieństwa dla wszystkich możliwych znaków. Model skompilowano, stosując optymalizator Adam z zadanym współczynnikiem uczenia 0.01 oraz funkcję straty categorical_crossentropy, odpowiednią dla problemu wieloklasowej klasyfikacji.

```python
# Budowa sieci neuronowej
model = Sequential([
    Embedding(input_dim=len(unique_chars), output_dim=32),
    LSTM(128),
    Dense(len(unique_chars), activation='softmax')
])

# Kompilacja modelu
model.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy')

# Wczesne zatrzymanie uczenia przy braku poprawy
stop_callback = EarlyStopping(monitor='loss', patience=10, restore_best_weights=True)

# Trening
history = model.fit(X, y, batch_size=32, epochs=20, callbacks=[stop_callback])

# Wynik końcowy
print(f"Final training loss: {history.history['loss'][-1]:.4f}")
```

Rys. 4. Budownie sieci neuronowej dla 20 epok.

```
Epoch 1/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 3.2286
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 3.0532
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.9430
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.8513
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.8036
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.6417
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.4741
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2.1523
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 1.9568
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 1.6594
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 1.4406
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 1.1218
Epoch 13/20
...
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0800
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0581
Final training loss: 0.0561
```

Rys. 5. Wynik wywołania programu - trenowanie 20 epok, obliczanie final_loss.

4. Wyniki

Skuteczność procesu uczenia: Zaobserwowano systematyczny i znaczący spadek wartości funkcji straty już od pierwszej epoki, co świadczyło o prawidłowym działaniu mechanizmu uczenia się modelu. Początkowa wartość błędu wynosiła 3.2286, natomiast końcowa spadła do poziomu 0.0561.

Zdolność modelu do uczenia zależności sekwencyjnych: Utrzymujący się trend malejący funkcji straty w kolejnych epokach wskazywał, że model LSTM skutecznie nauczył się przewidywać kolejne znaki na podstawie zadanych sekwencji znaków wejściowych.