Klasyfikacja sentymentów recenzji filmów IMDB przy użyciu rekurencyjnych sieci neuronowych
Sieci Neuronowe

Karolina Łukasik

16 grudnia 2023

Spis treści

1 Wstęp

W poniższym raporcie zajmiemy się konstrukcją modeli klasyfikujących recenzje filmów na pozytywne bądź negatywne. W tym celu wykorzystamy rekurencyjne sieci neuronowe, a dokładniej Vanilla RNN oraz LSTM.

Zbiór danych pobrany został ze strony Kerasa i nie wymagał już dodatkowego preprocessingu. Sekwencje przycinane były do różnych długości, poniższe eksperymenty zostały przeprowadzone na sekwencjach długości 100.

Implementacja modeli przeprowadzona została z wykorzystaniem Pytorcha, dlatego dane przekształcone zostały do tensorów i pogrupowane w paczki zapomocą DataLoadera. Uczenie przebiegało na podzbiorze zbioru uczącego od 10 do 20% całego podzbioru. Dodatkowo ze zbioru treningowego został wydzielony zbiór walidacyjny, a testowy załadowany bez zmian.

Model LSTM zbudowany został z warstwy embeddingu, jednej lub dwóch warstw LSTM o dowolnym wymiarze, warstwy Dropout oraz warstwy wyjściowej składającej się z jednego neuronu. Analogicznie zbudowany został klasyczny model RNN.

```
RNNModel(
  (embedding): Embedding(20000, 100)
  (rnn): RNN(100, 128, num_layers=2, batch_first=True, dropout=0.1)
  (fc): Linear(in_features=128, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```

Rysunek 1: Model RNN

```
LSTMModel(
  (embedding): Embedding(20000, 100)
  (lstm): LSTM(100, 128, num_layers=2, batch_first=True, dropout=0.1)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=128, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```

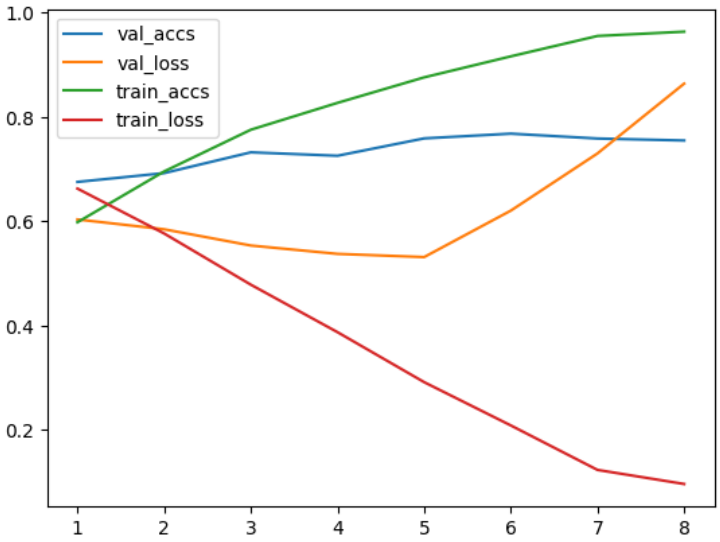
Rysunek 2: Model LSTM

Funkcja trenująca została zaimplementowana z wykorzystaniem optymalizatora Adam z wybraną wartością współczynnika uczenia, binarną cross entropią w roli funkcji straty oraz działa dla dowolnej liczby epoch. Podczas uczenia możemy obserwować postęp za pomocą wyświetlanej liczby batchy, które przeszły już przez sieć, czas uczenia oraz wyniki po każdym epochu. Funkcja zwraca wyniki po każdym epochu, dzięki czemu możemy wygenerować krzywe uczenia.

Poniżej przykład uczenia sieci zawieracącej jedną warstwę LSTM o wymiarze 128:

```
Training:  25%|██        | 2/8 [00:16<01:40, 16.68s/it, Training batch 1/78]
Epoch 1/8 | Train Loss: 0.663 Train Acc: 0.598 | Val Loss: 0.604 Val Acc: 0.676
Training:  50%|████      | 4/8 [00:34<00:42, 10.74s/it, Training batch 1/78]
Epoch 2/8 | Train Loss: 0.577 Train Acc: 0.696 | Val Loss: 0.585 Val Acc: 0.693
Training:  62%|█████     | 5/8 [00:51<00:29,  9.81s/it, Training batch 0/78]
Epoch 3/8 | Train Loss: 0.479 Train Acc: 0.776 | Val Loss: 0.554 Val Acc: 0.732
Training:  88%|███████▏  | 7/8 [01:13<00:12, 12.87s/it, Training batch 1/78]
Epoch 4/8 | Train Loss: 0.388 Train Acc: 0.827 | Val Loss: 0.538 Val Acc: 0.726
Training: 100%|████████| 8/8 [01:33<00:00, 11.59s/it, Training batch 1/78]
Epoch 5/8 | Train Loss: 0.292 Train Acc: 0.876 | Val Loss: 0.532 Val Acc: 0.759
Training: 9it [01:51, 13.20s/it, Training batch 0/78]
Epoch 6/8 | Train Loss: 0.209 Train Acc: 0.916 | Val Loss: 0.621 Val Acc: 0.768
Training: 11it [02:14, 15.78s/it, Training batch 1/78]
Epoch 7/8 | Train Loss: 0.124 Train Acc: 0.955 | Val Loss: 0.730 Val Acc: 0.759
Training: 100%|████████| 8/8 [02:33<00:00, 19.24s/it, Validation Loss: 0.864 | Validation Accuracy: 0.755]
Epoch 8/8 | Train Loss: 0.097 Train Acc: 0.964 | Val Loss: 0.864 Val Acc: 0.755
```

Rysunek 3: Trenowanie modelu



Rysunek 4: Krzywe uczenia po przejściu 8 epoch

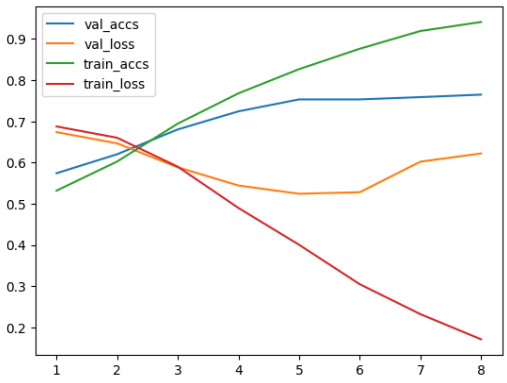
```
1  test_loss, test_accuracy = validate(test_loader)
2  print(f'Test loss: {test_loss} | Test accuracy: {test_accuracy}')
✓ 36.0s
Test loss: 0.5561053054716886 | Test accuracy: 0.7536365089514067
```

Rysunek 5: Wyniki na zbiorze testowym

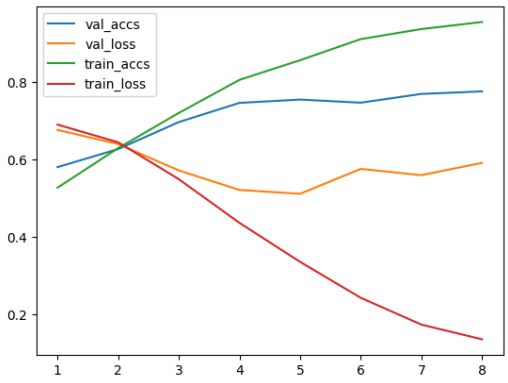
2 Porównanie działania modeli

Poniżej zostaną przeprowadzone ekperymenty na sieciach LSTM oraz RNN. Porównane zostaną różne wymiary sieci tzn. liczba warstw oraz wymiar warstwy.

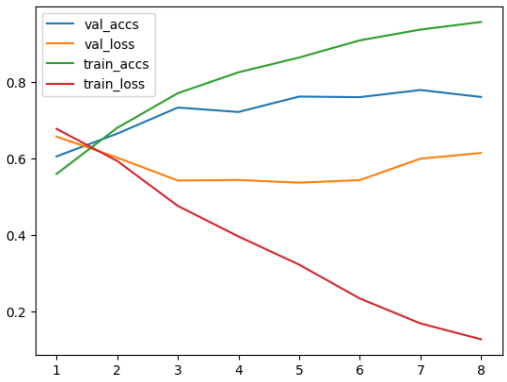
2.1 Sieć LSTM



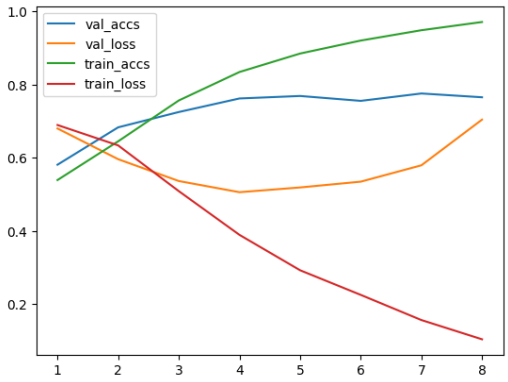
(a) 1 warstwa o wymiarze 32



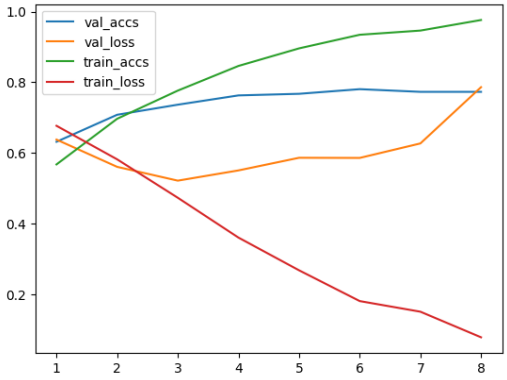
(b) 1 warstwa o wymiarze 64



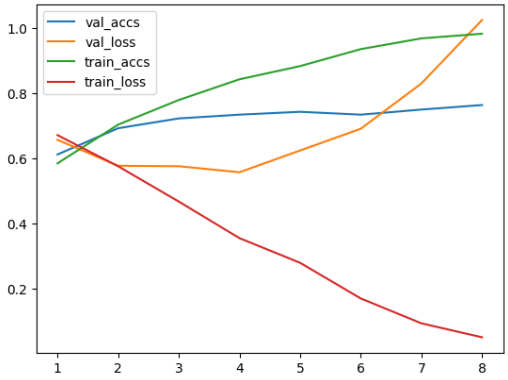
(c) 1 warstwa o wymiarze 128



(d) 2 warstwy o wymiarze 32

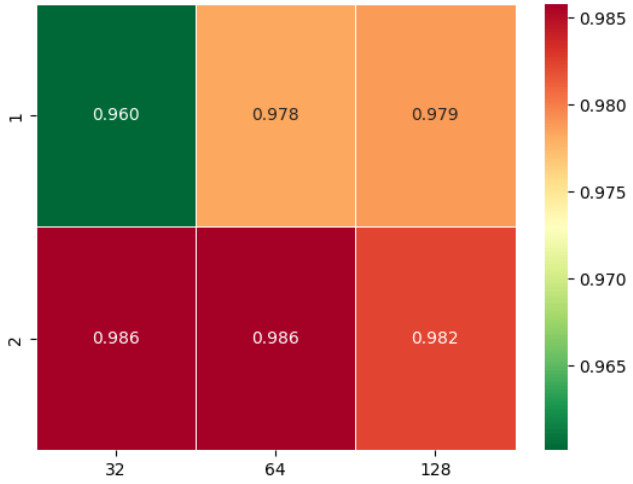


(e) 2 warstwy o wymiarze 64

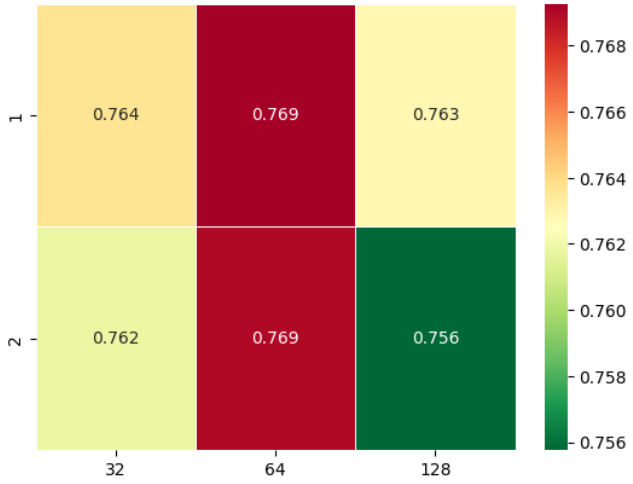


(f) 2 warstwy o wymiarze 128

Rysunek 6: Porównanie krzywych uczenia dla sieci LSTM o różnym wymiarze



(a) Wyniki na zbiorze treningowym

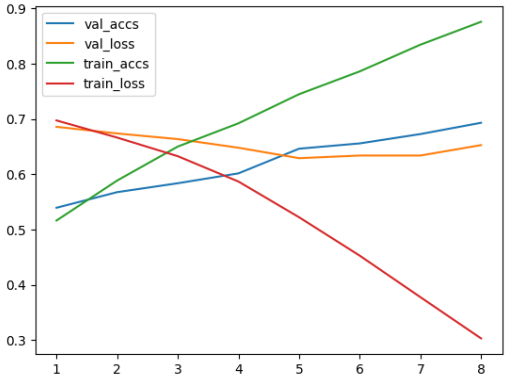


(b) Wyniki na zbiorze testowym

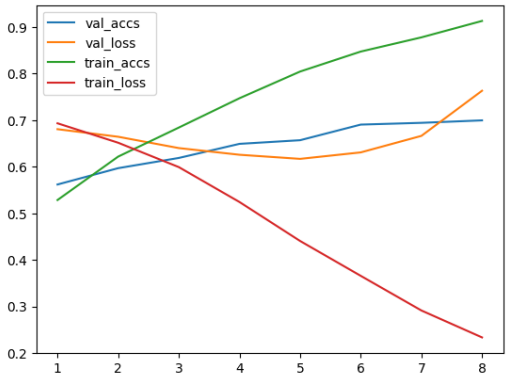
Rysunek 7: Porównanie wyników sieci LSTM o różnym wymiarze

Najlepsze wyniki osiągnęły modele o wymiarze 64 zarówno dla jednej jak i dwóch warstw. Większe modele zaczęły być podatne na przeuczenie, a mniejsze na niedouczenie. Jeśli chodzi o krzywe uczenia to prezentują się one bardzo podobnie oprócz funkcji straty na zbiorze walidacyjnym dla bardziej złożonych modeli, co może świadczyć o podatności modeli na przeuczenie.

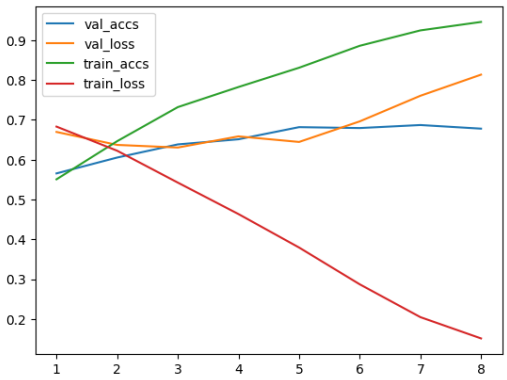
2.2 Sieć Vanilla RNN



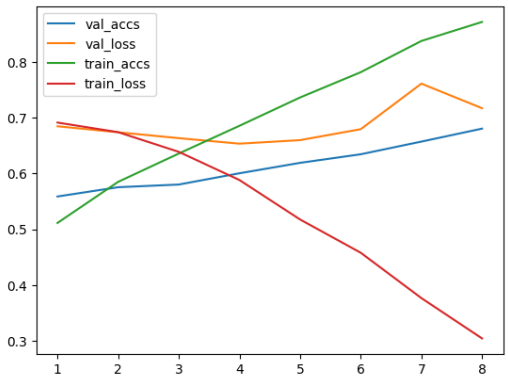
(a) 1 warstwa o wymiarze 32



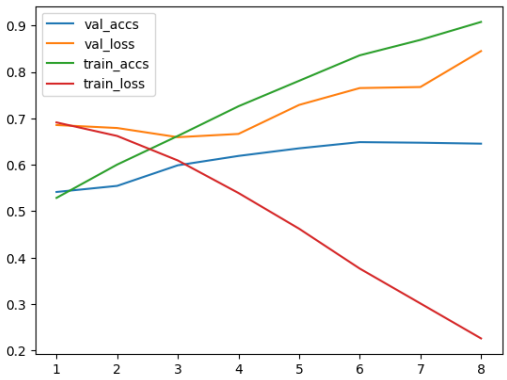
(b) 1 warstwa o wymiarze 64



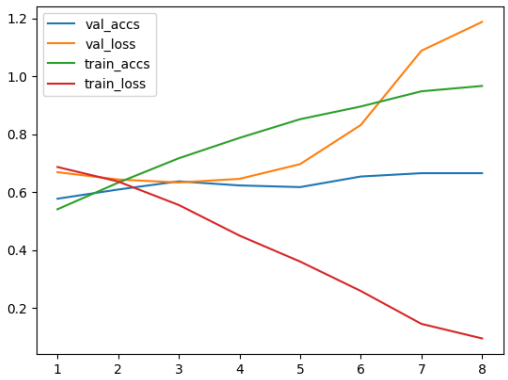
(c) 1 warstwa o wymiarze 128



(d) 2 warstwy o wymiarze 32



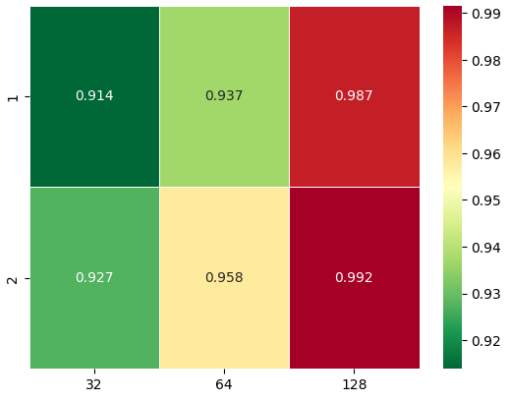
(e) 2 warstwy o wymiarze 64



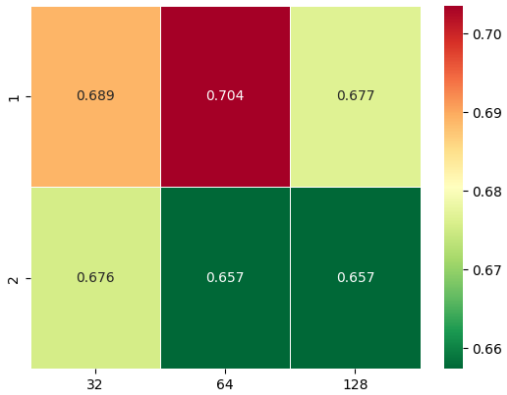
(f) 2 warstwy o wymiarze 128

Rysunek 8: Porównanie krzywych uczenia dla sieci RNN o różnym wymiarze

Sieci uzyskały najlepsze wyniki na zbiorze treningowym, kiedy wymiar warstwy był największy tzn. 128. Nie przekładało się to na wyniki na zbiorze testowym, gdzie najlepsze wyniki osiągnęła sieć jednowymiarowa z dowolnym wymiarem warstwy, szczególnie wymiarem 64. Daje to zupełnie inne rezultaty niż dla sieci LSTM. Wyniki osiągnięte za pomocą RNN były ogólnie lepsze na zbiorze treningowym, ale zdecydowanie gorsze na zbiorze testowym w porównaniu do sieci LSTM.



(a) Wyniki na zbiorze treningowym

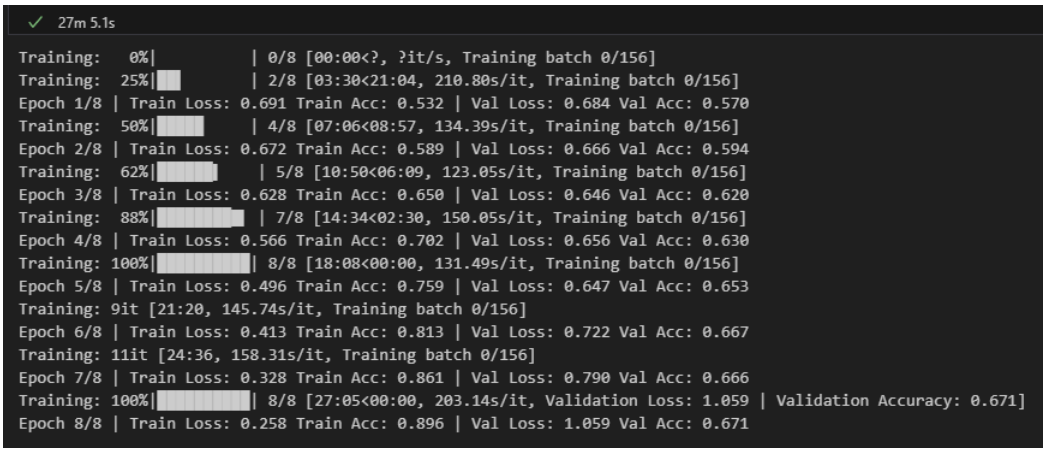


(b) Wyniki na zbiorze testowym

Rysunek 9: Porównanie wyników sieci RNN o różnym wymiarze

2.3 Przycinanie sekwencji

W poprzednich sekcjach trenowaliśmy modele na sekwencjach przyciętych do określonej długości (40/50/100 słów). Wynika to z faktu, że sieci muszą być trenowane na sekwencjach o tej samej długości, a przyrównanie wszystkich do długości maksymalnej obecnej w zbiorze danych wiązałoby się z bardzo długim czasem uczenia. Z drugiej strony eksperymenty na różnych długościach sekwencji pokazują, że czym dłuższa sekwencja tym lepsze wyniki osiągał model. Z tego względu przetestujemy teraz drugie podejście, czyli paddowanie sekwencji w obrębie każdego z batchy osobno. Dla każdej paczki znajdziemy maksymalną długość sekwencji i całą resztę przyrównamy do tej najdłuższej w obrębie tego batcha.



Rysunek 10: Trenowanie sieci na batchach paddowanych do maksymalnej długości sekwencji obecnej w paczce

Trenowanie sieci przebiegało o wiele wolniej niż w przypadku krótszych sekwencji, funkcje straty również spadały wolniej. Przejście 8 epoch zajęło prawie 30 minut, a wyniki wciąż nie były satysfakcjonujące, co świadczy o dużej przewadze strategii przycinania wszystkich sekwencji do tej samej, nie za długiej długości.