Politechnika Wrocławska

Autor: Filip Strózik 260377

Sieci Neuronowe

Sprawozdanie z ćwiczenia 7

Technologia

Python 3.11, Jupyter Notebook, PyTorch, Google Colab środowisko T4

Realizacja ćwiczenia

Cel ćwiczenia

W ćwiczeniu 7 należy porównać typy warstw sieciach rekurencyjnych, wymiar tych wartsw jak i wpływ przycinania sekwencji do niepełnej długości. Ćwiczenie będzie porównywać wartswy RNN (Recurrent Neural Network) z LSTM (Long short-term memory)

Przygotowanie i eksploracja zbioru danych IMDB

Próby pobrania zbioru z biblioteki torchtext nie zakończyły się powodzeniem, zatem postawiono na bardziej tradycyjną metode. Plik *IMDB Dataset.csv* pochodzi ze bioru na platformie Kaggle https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Zbiór posiada 50 000 rekordów opinii filmów ze strony IMDB. Atrybuty to: opinia (review), odczucie (sentiment). Opinia to sekwencja słów, a odczucie wartość, którą można sprowadzić do wartości binarnej: pozytywna, negatywna.

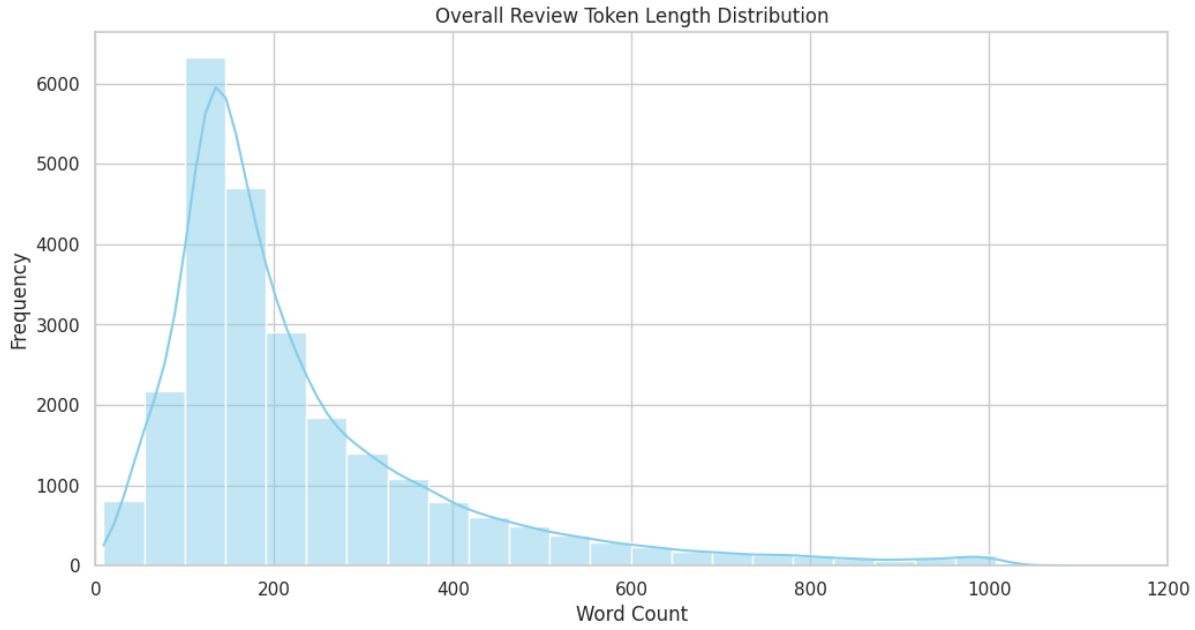
W ramach przeprowadzenia ćwiczenia w skończonym czasie, zdecydowano o użyciu 50% zbioru danych (25 000).

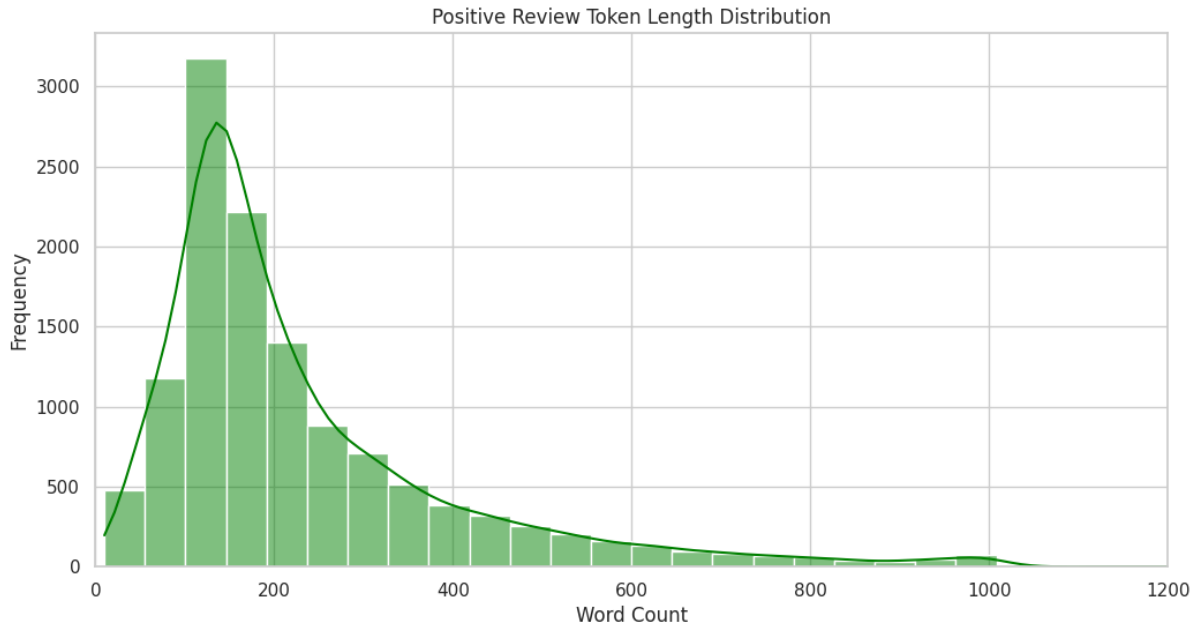
```
imdb_data = pd.read_csv('/content/drive/MyDrive/IMDB_Dataset.csv')

# Take only 50% of the dataset
imdb_data = imdb_data.sample(frac=0.5, random_state=42)

# Create a binary label column
imdb_data['label'] = (imdb_data['sentiment'] == 'positive').astype(int)

# Check balance of dataset
print(imdb_data['sentiment'].value_counts())

# Check word count of each review
imdb_data['word_count'] = imdb_data['review'].apply(lambda x: len(x.split()))

# Set styling as seaborn for convenience
sns.set(style='whitegrid')

# Plot word count distributions
plt.figure(figsize=(12, 6))

# Overall review word count distribution
sns.histplot(imdb_data['word_count'], bins=50, kde=True, color='skyblue')
plt.title('Overall Review Token Length Distribution')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()

# Positive review word count distribution
plt.figure(figsize=(12, 6))
sns.histplot(imdb_data[imdb_data['label'] == 1]['word_count'], bins=50, kde=True, color='green')
plt.title('Positive Review Token Length Distribution')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
```

```
# Negative review word count distribution
plt.figure(figsize=(12, 6))
sns.histplot(imdb_data[imdb_data['label'] == 0]['word_count'], bins=50, kde=True, color='red')
plt.title('Negative Review Token Length Distribution')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
```

positive 12517

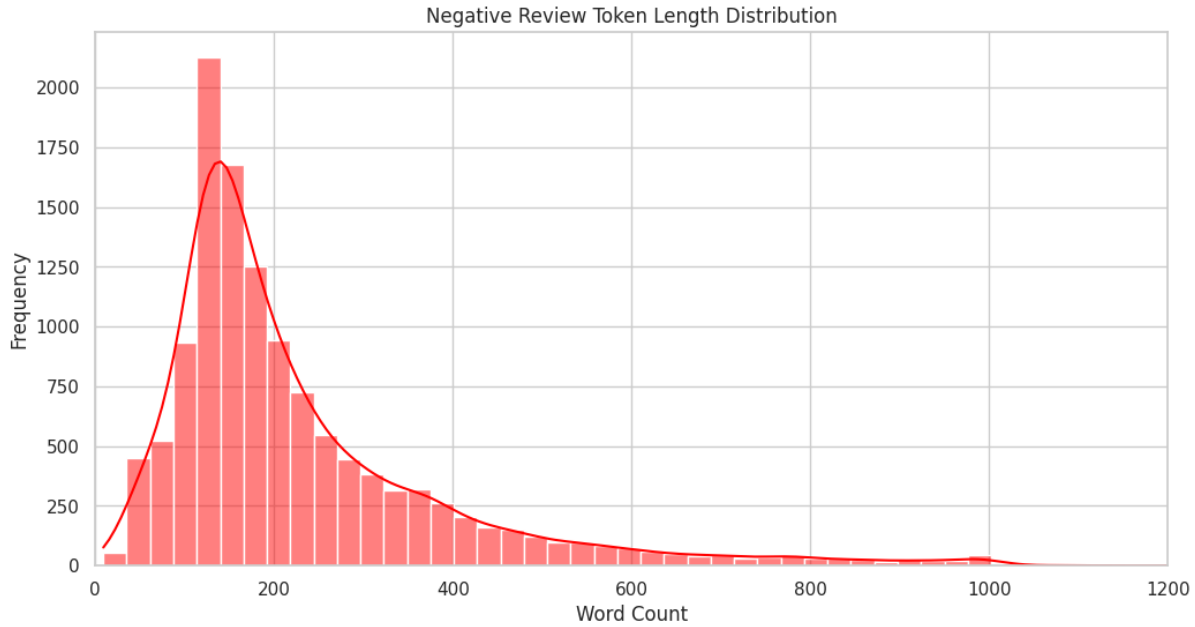
negative 12483





Politechnika Wrocławska

Autor: Filip Strózik 260377

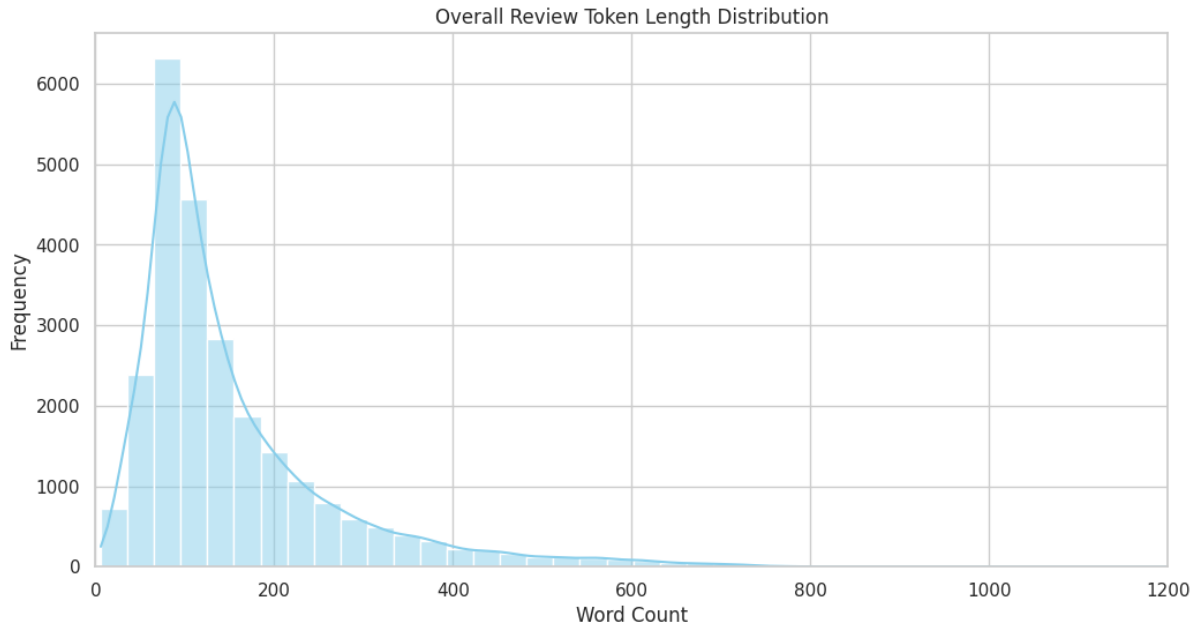


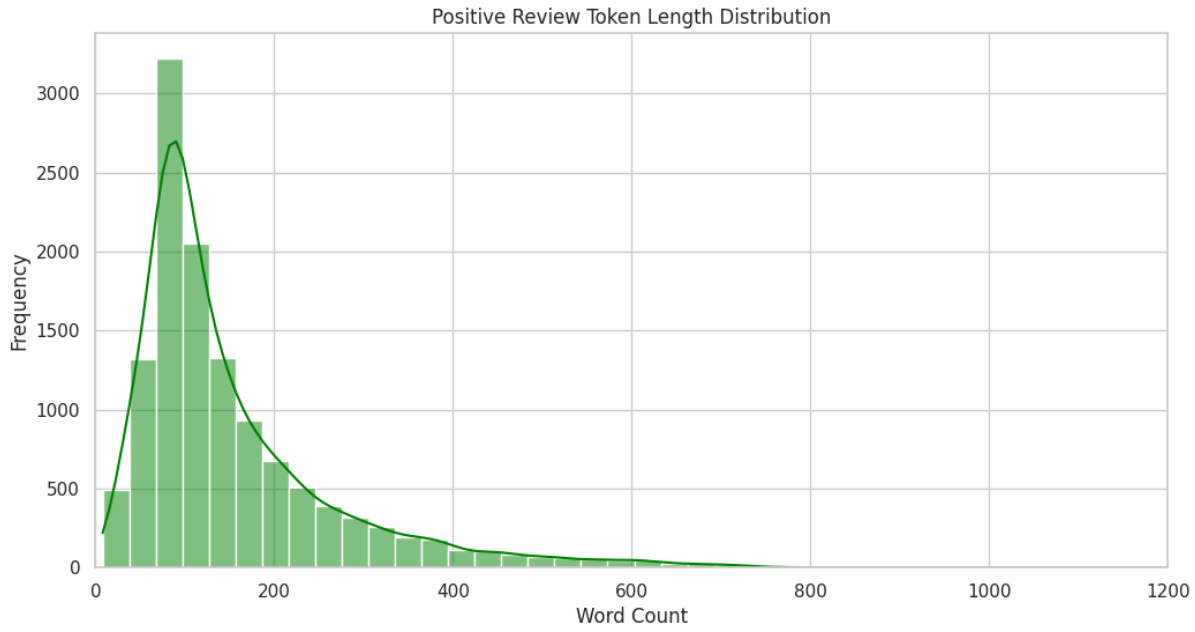
W ramach przeprocesowania danych wykonano:

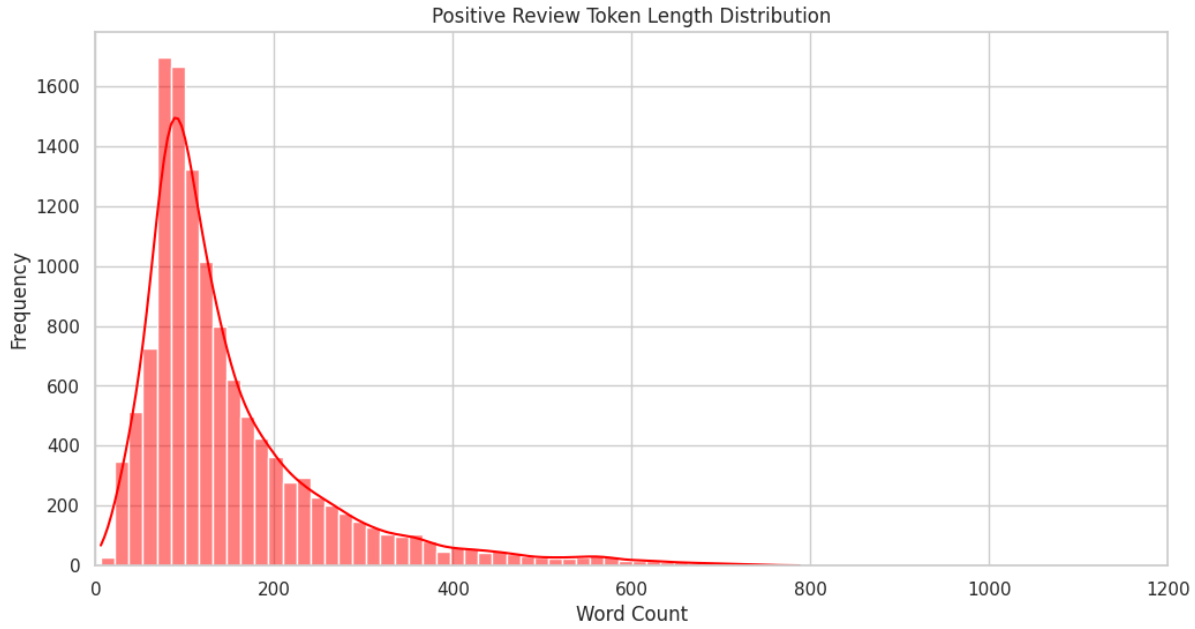
1. Oczyszczenie tekstu z:
   1.1. linków,
   1.2. znaków poza zakresem ASCII oraz emoji,
   1.3. nadmiarowych białych znaków,
   1.4. powtarzających się znaków.
2. Przetworzenie tekstu
   2.1. Tokenizacja tekstu i usunięcie tzw. Stopwords (słów, które nie niosą dużo znaczenia)
   2.2. Lematyzacja słów (przywracanie do formy podstawowej)
   2.3. Połączenie przetworzonych słów w jeden ciąg tekstowy.
3. Zapis do pliku CSV.

```
def clean_text(text):
    text = re.sub(r'https?://\S+|www\.\S+|<[^>]+>', '', text)

    text = re.sub(r'[^\x00-\x7f]|['
                  u'\U0001F600-\U0001F64F'
                  u'\U0001F300-\U0001F5FF'
                  u'\U0001F680-\U0001F6FF'
                  u'\U0001F1E0-\U0001F1FF'
                  u'\U00002702-\U000027B0'
                  u'\U000024C2-\U0001F251'
                  ']+', '', text)

    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra whitespaces

    # Correct repeated characters (e.g., Woooow -> Wow)
    text = re.sub(r'(.)\1+', r'\1\1', text)

    return text

def preprocess_text(text):
    # Tokenize and remove stopwords
    tokens = [word for word in word_tokenize(text) if word.lower() not in stopwords]

    # Lemmatize
    lemmatizer = WordNetLemmatizer()
    lemmas = [lemmatizer.lemmatize(token) for token in tokens]
```

```
    return ' '.join(lemmas)

imdb_data['cleaned'] = imdb_data['review'].progress_apply(clean_text)
imdb_data['processed'] = imdb_data['cleaned'].progress_apply(preprocess_text)

imdb_data[['processed', 'label']].to_csv('./imdb_processed.csv', index=False, header=True)

plt.figure(figsize=(12, 6))
sns.displot(imdb_data[imdb_data['label'] == 1]['processed'].apply(lambda x: len(x.split())),
kde=False, color='green', label='Positive')
plt.title('Positive Review Word Count Distribution')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.legend()
plt.show()

plt.figure(figsize=(12, 6))
sns.displot(imdb_data[imdb_data['label'] == 0]['processed'].apply(lambda x: len(x.split())),
kde=False, color='red', label='Negative')
plt.title('Negative Review Word Count Distribution')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```







Po przetworzeniu danych zdecydowano na badanie długości niepełnej sekwencji dla 100 i 200 wyrazów.

Budowa słownika i kodowanie recenzji

Przetwarzanie recenzji do postaci listy słów

```
reviews = data.processed.values
words = ' '.join(reviews)
words = words.split()
```

Budowa słownika

```
counter = Counter(words)
vocab = sorted(counter, key=counter.get, reverse=True)
int2word = dict(enumerate(vocab, 1))
int2word[0] = '<PAD>'
word2int = {word: id for id, word in int2word.items()}
```

Padowanie cech

```
def pad_features(reviews, pad_id, seq_length):
    features = np.full((len(reviews), seq_length), pad_id, dtype=int)

    for i, row in enumerate(reviews):
        features[i, :len(row)] = np.array(row)[:seq_length]

    return features

seq_length_100 = 100
seq_length_200 = 200

features_100 = pad_features(reviews_enc, pad_id=word2int['<PAD>'], seq_length=seq_length_100)
features_200 = pad_features(reviews_enc, pad_id=word2int['<PAD>'], seq_length=seq_length_200)
```

Podział na zbiór treningowy i testowy

```
seq_lengths = [100, 200]
test_size = 0.2
features = [pad_features(reviews_enc, pad_id=word2int['<PAD>'], seq_length=seq_length) for
seq_length in seq_lengths]
train_test_splits = [train_test_split(feature, labels, test_size=test_size, random_state=2)
for feature in features]
```

Tworzenie zbiorów danych TensorDataset i DataLoader

```
batch_size = 400

trainsets = [TensorDataset(torch.from_numpy(X_train), torch.from_numpy(Y_train)) for X_train,
_, Y_train, _ in train_test_splits]
testsets = [TensorDataset(torch.from_numpy(X_test), torch.from_numpy(Y_test)) for _, X_test,
_, Y_test in train_test_splits]

train_loaders = [DataLoader(trainset, shuffle=True, batch_size=batch_size) for trainset in
trainsets]
test_loaders = [DataLoader(testset, shuffle=False, batch_size=batch_size) for testset in
testsets]
```

Architektura modelów sieci RNN i LSTM

```
class SentimentBaseModel(nn.Module):
    def __init__(self, vocab_size, output_size, hidden_size, embedding_size, n_layers,
rnn_type):
        super(SentimentBaseModel, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        # Create RNN layer based on the specified type
        rnn_types = {'LSTM': nn.LSTM, 'RNN': nn.RNN}
        if rnn_type not in rnn_types:
            raise ValueError("Invalid RNN type. Supported types: 'LSTM' or 'RNN'.")
        self.rnn = rnn_types[rnn_type](embedding_size, hidden_size, n_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x, hidden):
        x = x.long()
        x = self.embedding(x)
        o, hidden = self.rnn(x, hidden)
        o = o[:, -1, :]
        o = self.fc(o)
        o = self.sigmoid(o)

        return o, hidden

    def initHidden(self, batch_size):
```

```
        return torch.zeros(self.n_layers, batch_size, self.hidden_size),
torch.zeros(self.n_layers, batch_size, self.hidden_size)

class SentimentLSTMModel(SentimentBaseModel):
    def __init__(self, vocab_size, output_size, hidden_size, embedding_size, n_layers):
        super(SentimentLSTMModel, self).__init__(vocab_size, output_size, hidden_size,
embedding_size, n_layers, 'LSTM')

    def initHidden(self, batch_size):
        return torch.zeros(self.n_layers, batch_size, self.hidden_size),
torch.zeros(self.n_layers, batch_size, self.hidden_size)

class SentimentRNNModel(SentimentBaseModel):
    def __init__(self, vocab_size, output_size, hidden_size, embedding_size, n_layers):
        super(SentimentRNNModel, self).__init__(vocab_size, output_size, hidden_size,
embedding_size, n_layers, 'RNN')

    def initHidden(self, batch_size):
        return torch.zeros(self.n_layers, batch_size, self.hidden_size)
```

Recurrent Neural Network (RNN):

RNN to rodzaj sieci neuronowej, która jest zdolna do przechowywania informacji o poprzednich stanach, co jest istotne w przypadku sekwencji danych, takich jak tekst. Jednakże, tradycyjne RNN mogą mieć problem z długoterminowym zapamiętywaniem informacji z powodu zanikającego lub eksplodującego gradientu.

Long Short-Term Memory (LSTM):

LSTM to rodzaj rozwinięcia RNN, stworzone specjalnie, aby przeciwdziałać problemowi zanikającego gradientu. Posiada ona specjalne bramki (gates), które kontrolują przepływ informacji, eliminując problem utraty informacji na długie odległości. Dzięki tym bramkom, LSTM jest zdolna do przechowywania i odczytywania informacji na dłuższe dystanse, co jest istotne w analizie sekwencji tekstu.

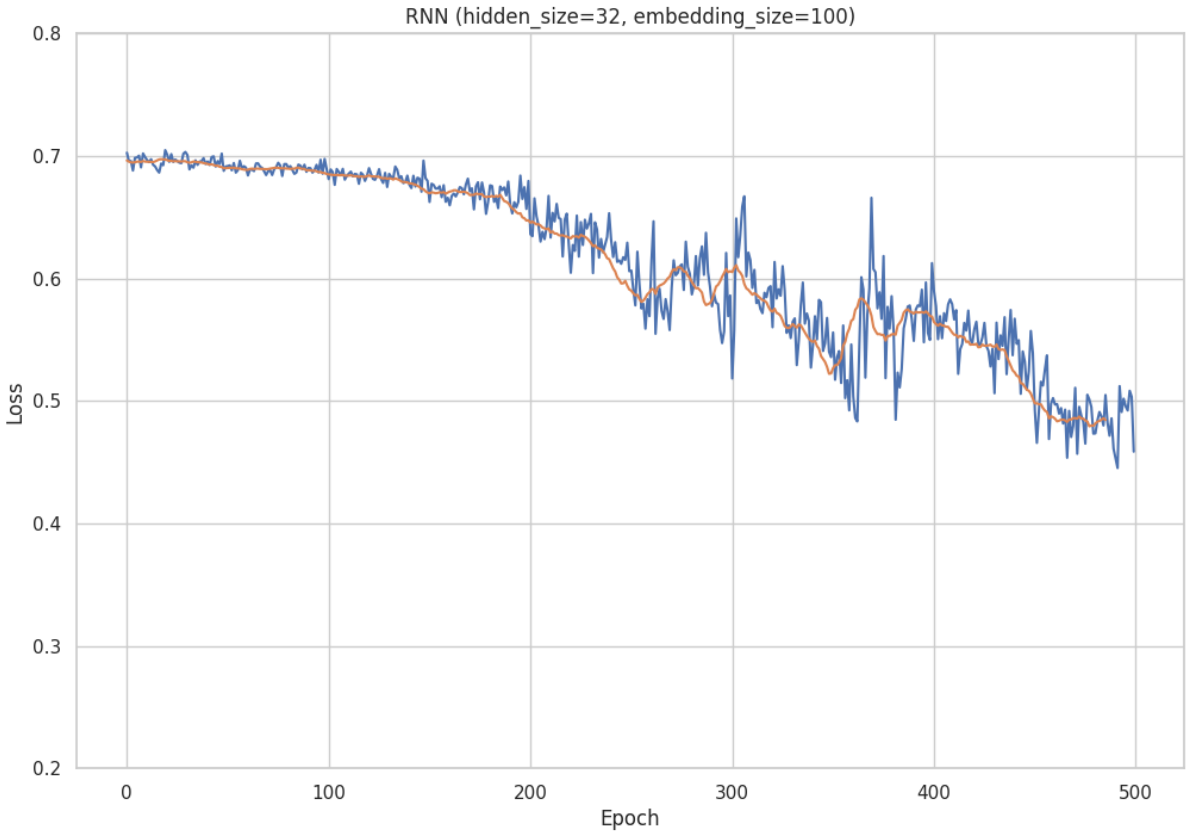
Politechnika Wrocławska

Autor: Filip Strózik 260377

Porównanie domyślnej konfiguracji RNN vs LSTM

```
RNN (hidden_size=32, embedding_size=100)
Epoch: 1/10, Loss: 0.6914
Epoch: 2/10, Loss: 0.6893
Epoch: 3/10, Loss: 0.6796
Epoch: 4/10, Loss: 0.6796
Epoch: 5/10, Loss: 0.6058
Epoch: 6/10, Loss: 0.5860
Epoch: 7/10, Loss: 0.5358
Epoch: 8/10, Loss: 0.6125
Epoch: 9/10, Loss: 0.5389
Epoch: 10/10, Loss: 0.4586
```

```
Classification Report:
              precision    recall  f1-score   support

    Negative       0.61      0.48      0.54      2553
    Positive       0.56      0.68      0.61      2447

    accuracy                           0.58      5000
   macro avg       0.58      0.58      0.57      5000
weighted avg       0.58      0.58      0.57      5000
```

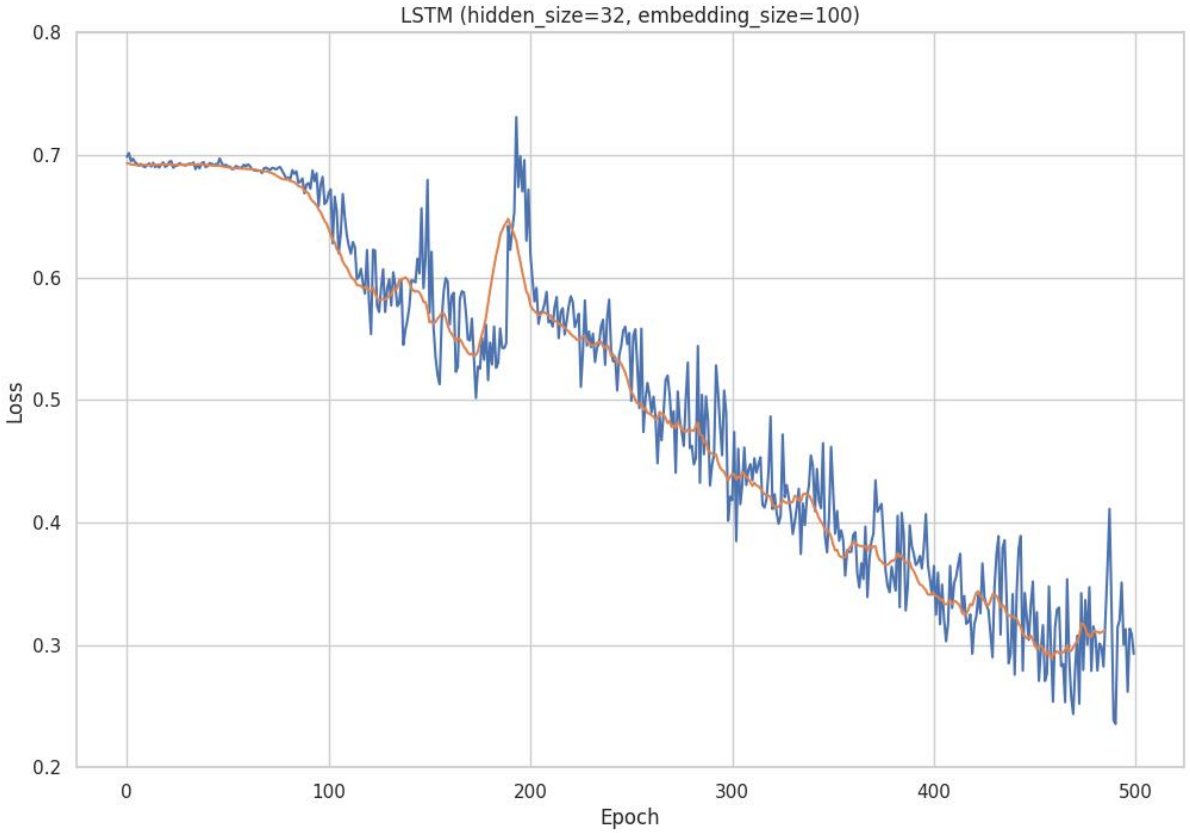


Politechnika Wrocławska

Autor: Filip Strózik 260377

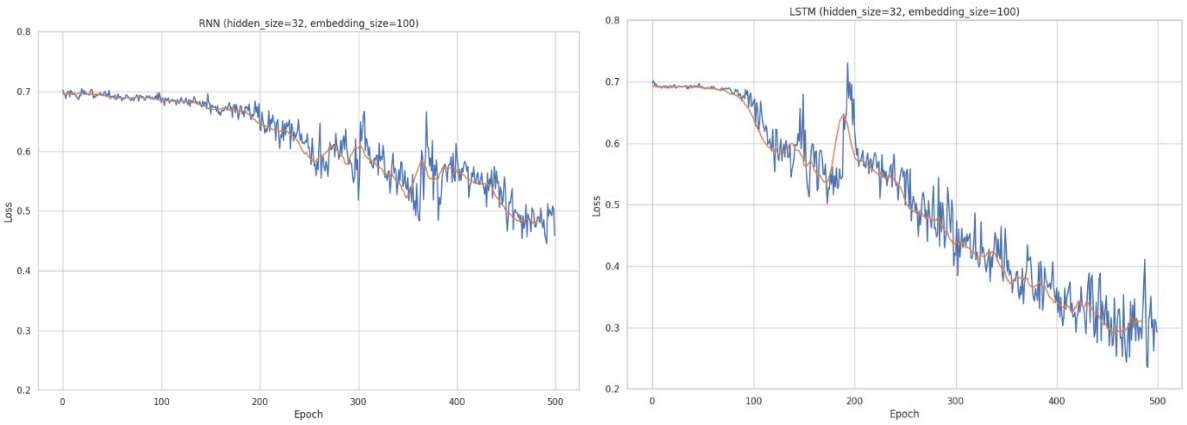
```
LSTM (hidden_size=32, embedding_size=100)
Epoch: 1/10, Loss: 0.6919
Epoch: 2/10, Loss: 0.6619
Epoch: 3/10, Loss: 0.6795
Epoch: 4/10, Loss: 0.6717
Epoch: 5/10, Loss: 0.5547
Epoch: 6/10, Loss: 0.4211
Epoch: 7/10, Loss: 0.4618
Epoch: 8/10, Loss: 0.3409
Epoch: 9/10, Loss: 0.3517
Epoch: 10/10, Loss: 0.2926
```

```
Classification Report:
              precision    recall  f1-score   support

    Negative       0.76      0.78      0.77      2553
    Positive       0.76      0.74      0.75      2447

    accuracy                           0.76      5000
   macro avg       0.76      0.76      0.76      5000
weighted avg       0.76      0.76      0.76      5000
```



Politechnika Wrocławska

Autor: Filip Strózik 260377



Po porównaniu obu wykresów można zauważyć, że LSTM dla zadanych hiperparametrów osiąga niższy poziom kosztu niż model z warstwą RNN. Można zauważyć w tym przypadku „skoki” funkcji kosztu w LSTM, związane z doborem optymalizatora momentum Adam.

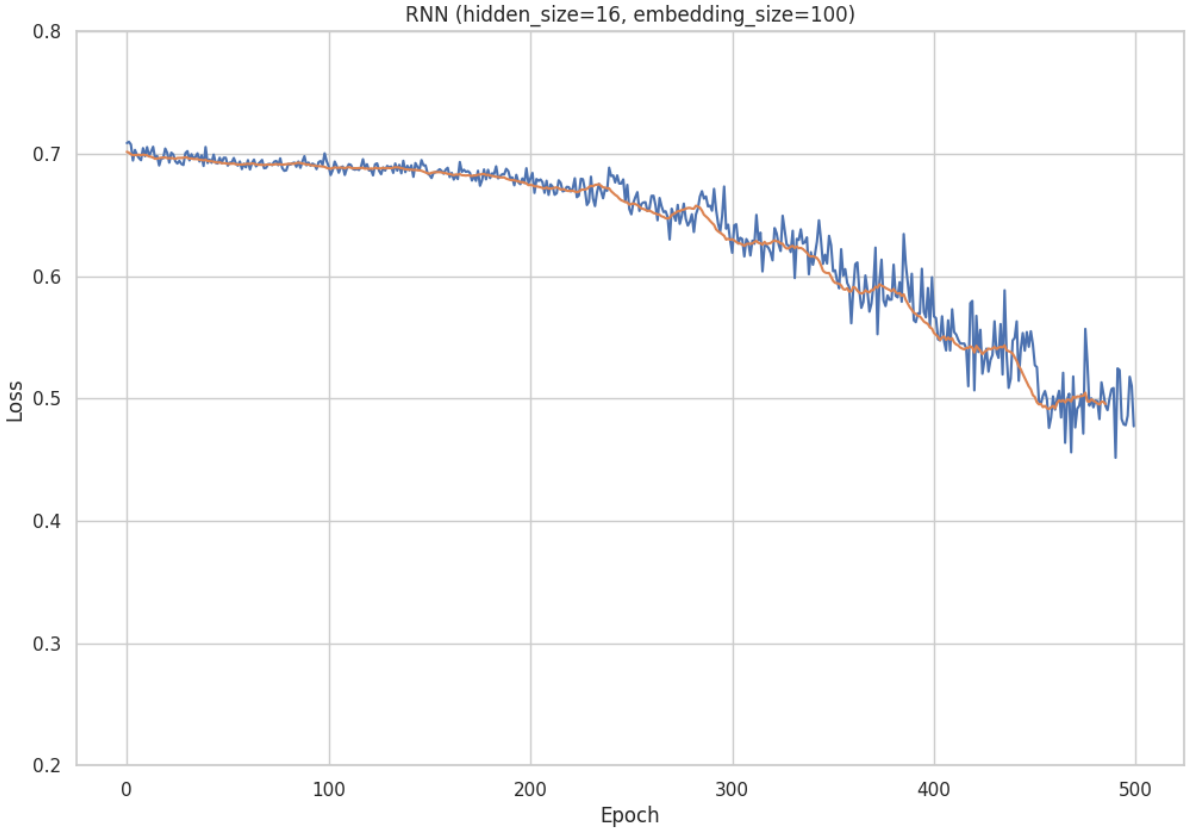
Politechnika Wrocławska

Autor: Filip Strózik 260377

Porównanie wymiaru warstwy rekurencyjnej

```
RNN (hidden_size=16, embedding_size=100)
Epoch: 1/10, Loss: 0.6967
Epoch: 2/10, Loss: 0.6939
Epoch: 3/10, Loss: 0.6840
Epoch: 4/10, Loss: 0.6760
Epoch: 5/10, Loss: 0.6549
Epoch: 6/10, Loss: 0.6293
Epoch: 7/10, Loss: 0.6253
Epoch: 8/10, Loss: 0.5990
Epoch: 9/10, Loss: 0.5441
Epoch: 10/10, Loss: 0.4772
```

```
Classification Report:
              precision    recall  f1-score   support

    Negative       0.56      0.35      0.43      2553
    Positive       0.51      0.70      0.59      2447

    accuracy                           0.53      5000
   macro avg       0.53      0.53      0.51      5000
weighted avg       0.53      0.53      0.51      5000
```



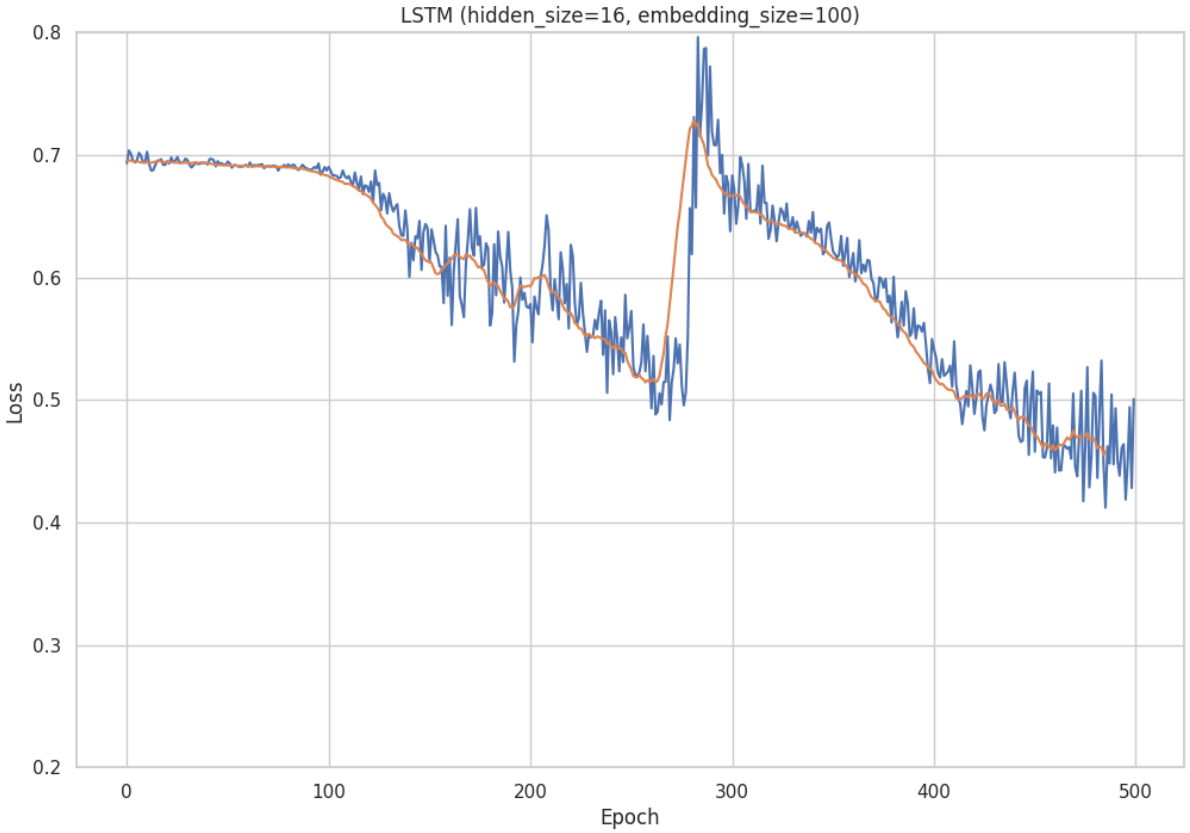
Politechnika Wrocławska

Autor: Filip Strózik 260377

```
LSTM (hidden_size=16, embedding_size=100)
Epoch: 1/10, Loss: 0.6920
Epoch: 2/10, Loss: 0.6874
Epoch: 3/10, Loss: 0.6403
Epoch: 4/10, Loss: 0.5751
Epoch: 5/10, Loss: 0.5594
Epoch: 6/10, Loss: 0.6377
Epoch: 7/10, Loss: 0.6318
Epoch: 8/10, Loss: 0.5496
Epoch: 9/10, Loss: 0.5231
Epoch: 10/10, Loss: 0.5006
```

Classification Report:

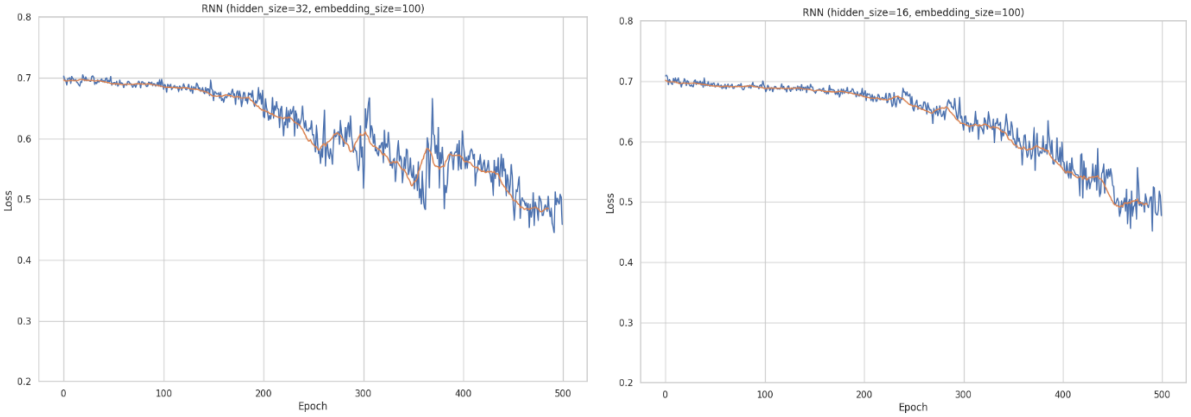
	precision	recall	f1-score	support
Negative	0.70	0.80	0.75	2553
Positive	0.75	0.65	0.70	2447
accuracy			0.72	5000
macro avg	0.73	0.72	0.72	5000
weighted avg	0.73	0.72	0.72	5000



Politechnika Wrocławska

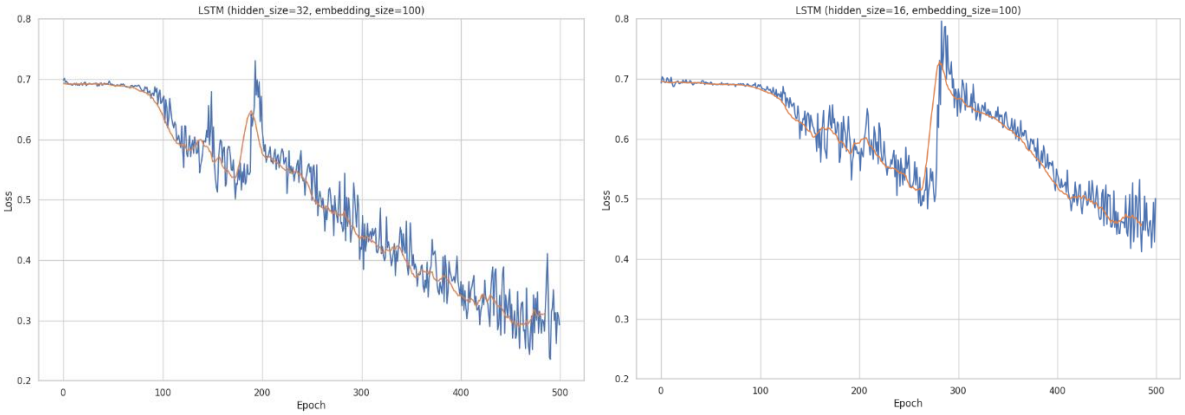
Autor: Filip Strózik 260377

RNN (32 vs 16, 100)



W tym porównaniu mniejszy wymiar warstwy sprawił spowolnienie oraz ustabilizowanie się procesu uczenia się. Ograniczenie pojemności modelu nie sprawiło w tym przypadku aby wymagał większej liczby epok aby uzyskać podobny wynik.

LSTM (32 vs 16, 100)



W przypadku LSTM, ograniczenie rozmiaru warstwy rekurencyjnej sprawiło, skok związany z optymalizatorem momentum, został opóźniony, przez co model wymagałby więcej kroków uczenia aby zbliżyć się do wyników z domyślnym rozmiarem warstwy rekurencyjnej.

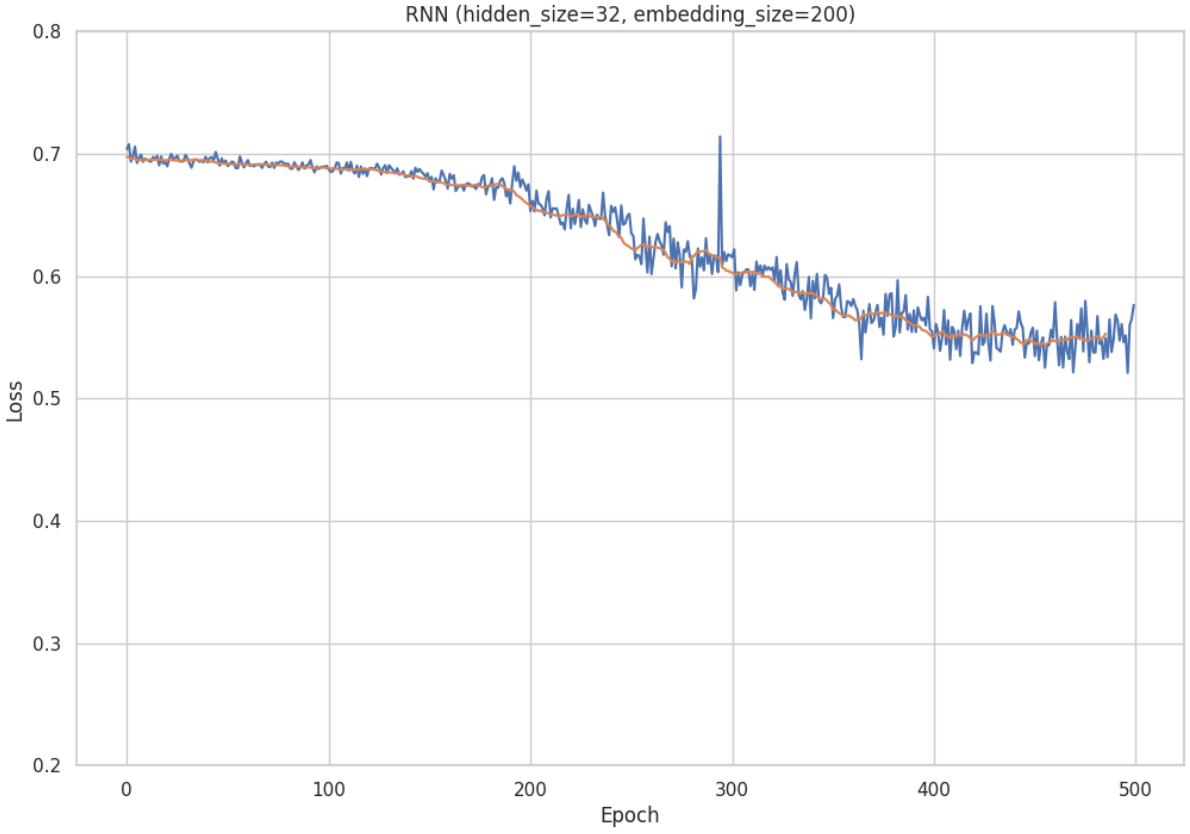
Politechnika Wrocławska

Autor: Filip Strózik 260377

Porównanie rozmiaru sekwencji

```
RNN (hidden_size=32, embedding_size=200)
Epoch: 1/10, Loss: 0.6944
Epoch: 2/10, Loss: 0.6901
Epoch: 3/10, Loss: 0.6841
Epoch: 4/10, Loss: 0.6749
Epoch: 5/10, Loss: 0.6510
Epoch: 6/10, Loss: 0.6168
Epoch: 7/10, Loss: 0.5903
Epoch: 8/10, Loss: 0.5549
Epoch: 9/10, Loss: 0.5578
Epoch: 10/10, Loss: 0.5764
```

```
Classification Report:
              precision    recall  f1-score   support

    Negative       0.51      0.88      0.64      2553
    Positive       0.47      0.11      0.18      2447

    accuracy                           0.50      5000
   macro avg       0.49      0.49      0.41      5000
weighted avg       0.49      0.50      0.42      5000
```

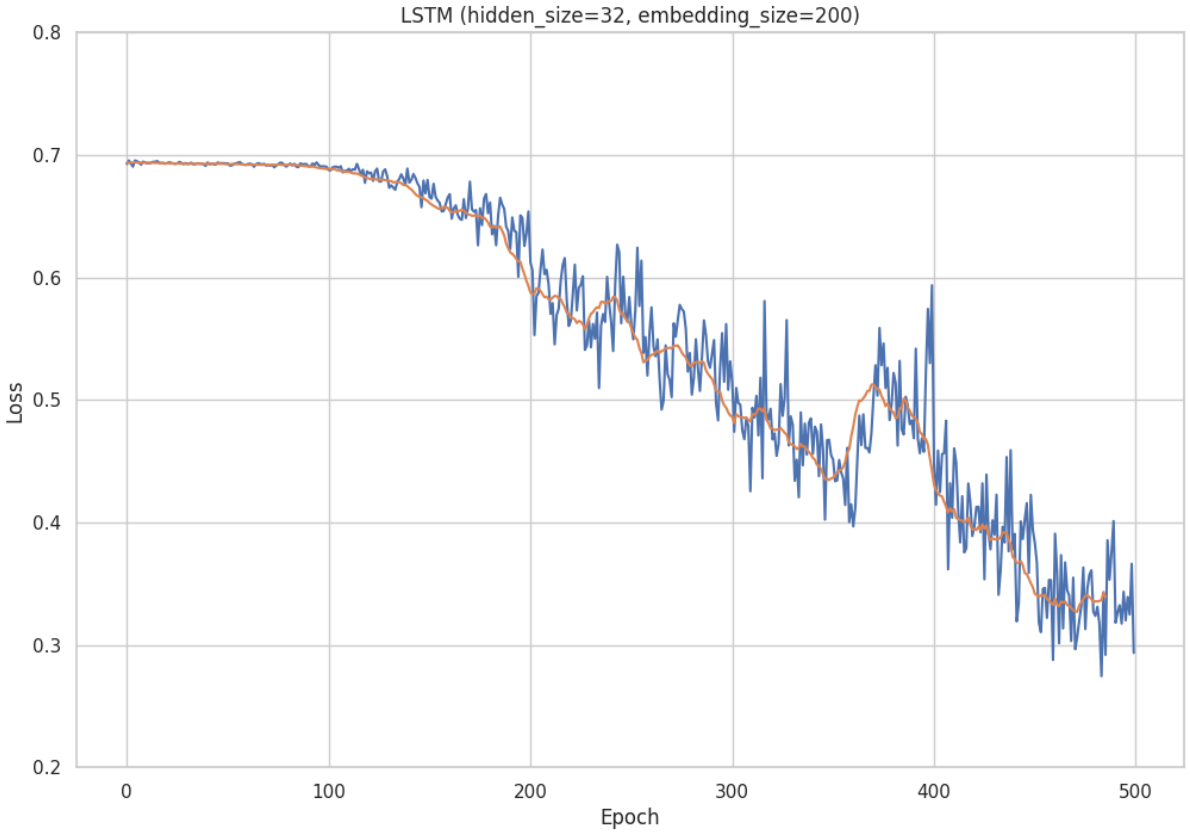


Politechnika Wrocławska

Autor: Filip Strózik 260377

```
LSTM (hidden_size=32, embedding_size=200)
Epoch: 1/10, Loss: 0.6931
Epoch: 2/10, Loss: 0.6903
Epoch: 3/10, Loss: 0.6796
Epoch: 4/10, Loss: 0.6538
Epoch: 5/10, Loss: 0.5840
Epoch: 6/10, Loss: 0.5315
Epoch: 7/10, Loss: 0.4549
Epoch: 8/10, Loss: 0.5936
Epoch: 9/10, Loss: 0.3937
Epoch: 10/10, Loss: 0.2935
```

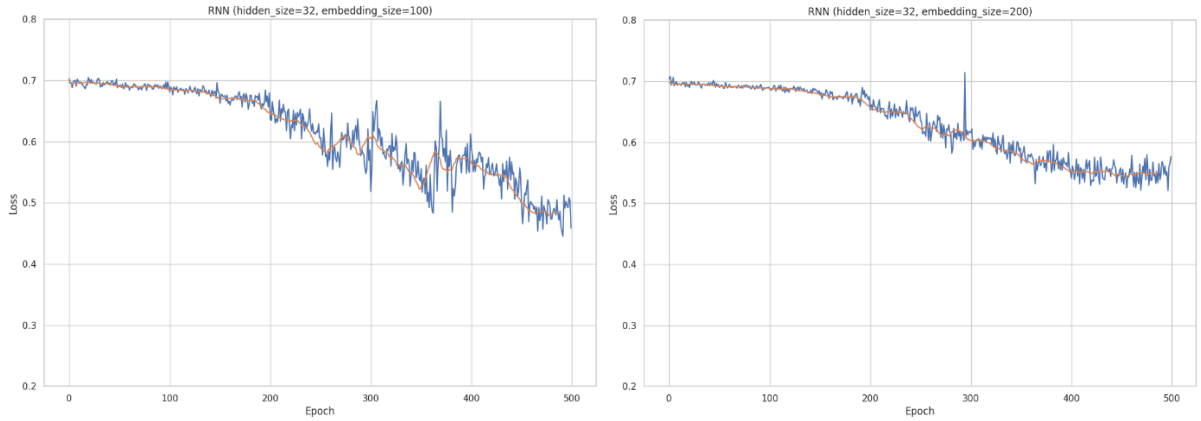
```
Classification Report:
              precision    recall  f1-score   support

    Negative       0.73      0.73      0.73      2553
    Positive       0.72      0.71      0.71      2447

    accuracy                           0.72      5000
   macro avg       0.72      0.72      0.72      5000
weighted avg       0.72      0.72      0.72      5000
```



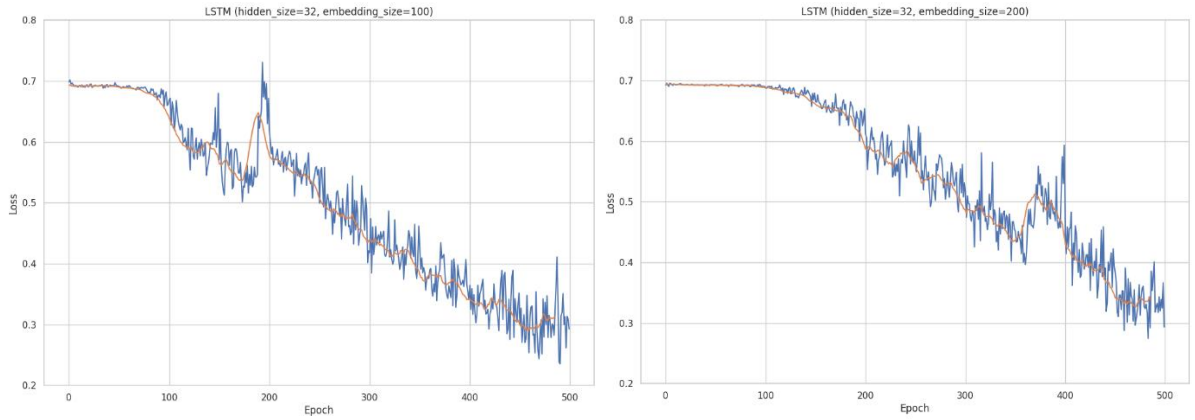
Politechnika Wrocławska

Autor: Filip Strózik 260377

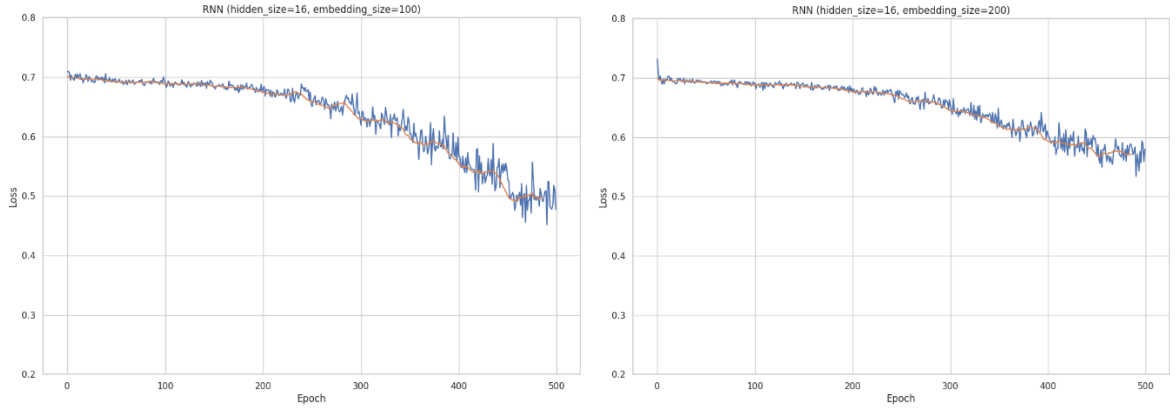
RNN (32, 100 vs 200)



LSTM (32, 100 vs 200)



RNN (16, 100 vs 200)

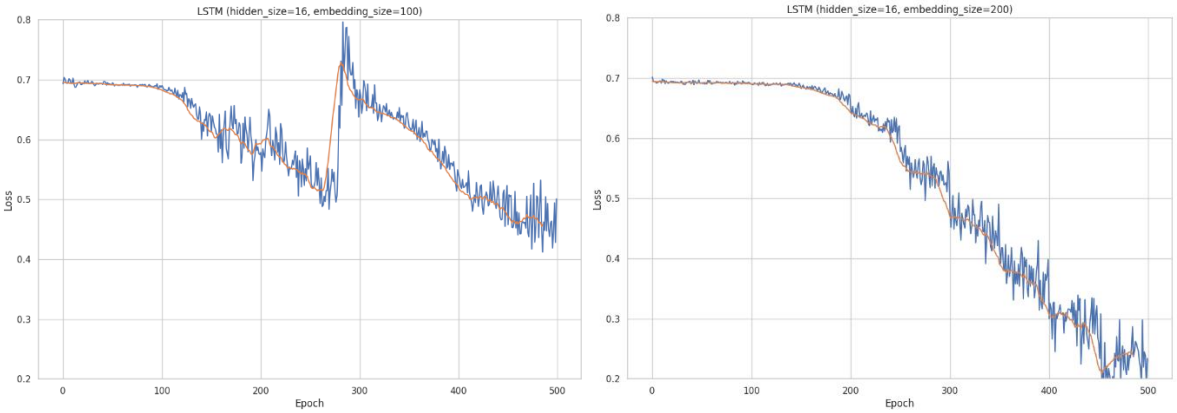


Obserwując wykresy można zauważyć, że zwiększenie rozmiaru sekwencji sprawia, ogólne spowolnienie uczenia, ponieważ dłuższe sekwencje oznaczają więcej kroków czasowych co może być kosztowne obliczeniowo. W przypadku RNN można zauważyć problem zanikającego gradientu. W przypadku LSTM, lepiej radzi sobie przy dłuższej sekwencji niż RNN.

Politechnika Wrocławska

Autor: Filip Strózik 260377

LSTM (16, 100 vs 200)



Oprócz skoku prawdopodobnie związanego z użyciem optymalizatorów momentum, które przyspieszają uczenie, ale mogą powodować takie skoki w przypadku, gdy optymalizator przeskoczy minimum lokalne, LSTM poradził sobie najlepiej z uczeniem w przypadku zwiększenia rozmiaru sekwencji. Widać jego dobre przystosowanie do dłuższych rozmiarów sekwencji niż RNN.