Sztuczna inteligencja i inżynieria wiedzy

Lista 5

Konrad Kiełczyński 260409

1.Przekształcenie danych w trenujące i walidujące

1. Odczyt danych z pliku Excel. Dane ocen są odczytywane z pliku Excel ’jester-data.xls’.
2. Przetwarzanie danych. Usunięcie pierwszej kolumny będącą liczbą ocen żartów przez osobę. Oraz zmienienie wartości 99 na NaN by nie były one brane pod uwagę w średniej ocenie żartu.

```
ratings_data = pd.read_excel('data/jester-data.xls', header=None)
ratings_data = ratings_data.iloc[:, 1:].replace(99, float('nan'))
ratings = ratings_data.mean()
```

3. Przygotowanie tekstów żartów: Teksty żartów pobierane są za pomocą metody z biblioteki do web-scrappingu BeautifulSoup. Następnie dokonujemy ekstrakcji cech z tych danych. Jest to wymagane ponieważ dane do uczenia sieci MLP muszą mieć postać wektorową. Ten zapis pozwala za pomocą wartości liczbowych określenia sensu zdania. Jest to osiągnięte przez SentenceTrasformer z modelem BERT.

```
model = SentenceTransformer('bert-base-cased')
embeddings = model.encode(jokes)
```

4. Podział danych na zbiór treningowy i walidacyjny jest w proporcji 80/20.

```
X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=test_size, random_state=RANDOM_STATE)
```

2.Główny algorytm do uczenia sieci neuronowej

```
def train_mlp(X, y, solver='sgd', alpha=0.0, learning_rate='constant',
test_size=0.2, epochs=500, learning_rate_init=0.001,
hidden_layer_sizes=(100,)):
    X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=test_size, random_state=RANDOM_STATE)

    mlp = MLPRegressor(
        solver=solver,
        alpha=alpha,
        learning_rate=learning_rate,
        learning_rate_init=learning_rate_init,
        hidden_layer_sizes=hidden_layer_sizes,
        random_state=RANDOM_STATE)

    train_losses = []
    val_losses = []

    for epoch in range(epochs):
        mlp.partial_fit(X_train, y_train)
        train_loss = mean_squared_error(y_train, mlp.predict(X_train))
        val_loss = mean_squared_error(y_val, mlp.predict(X_val))
        train_losses.append(train_loss)
        val_losses.append(val_loss)

    return train_losses, val_losses
```

Na początku, dane wejściowe (X) i odpowiadające im etykiety (y) są dzielone na zbiór treningowy i zbiór walidacyjny.

Następnie tworzony jest model sieci MLP z parametrami (w nawiasach domyślne wartości):

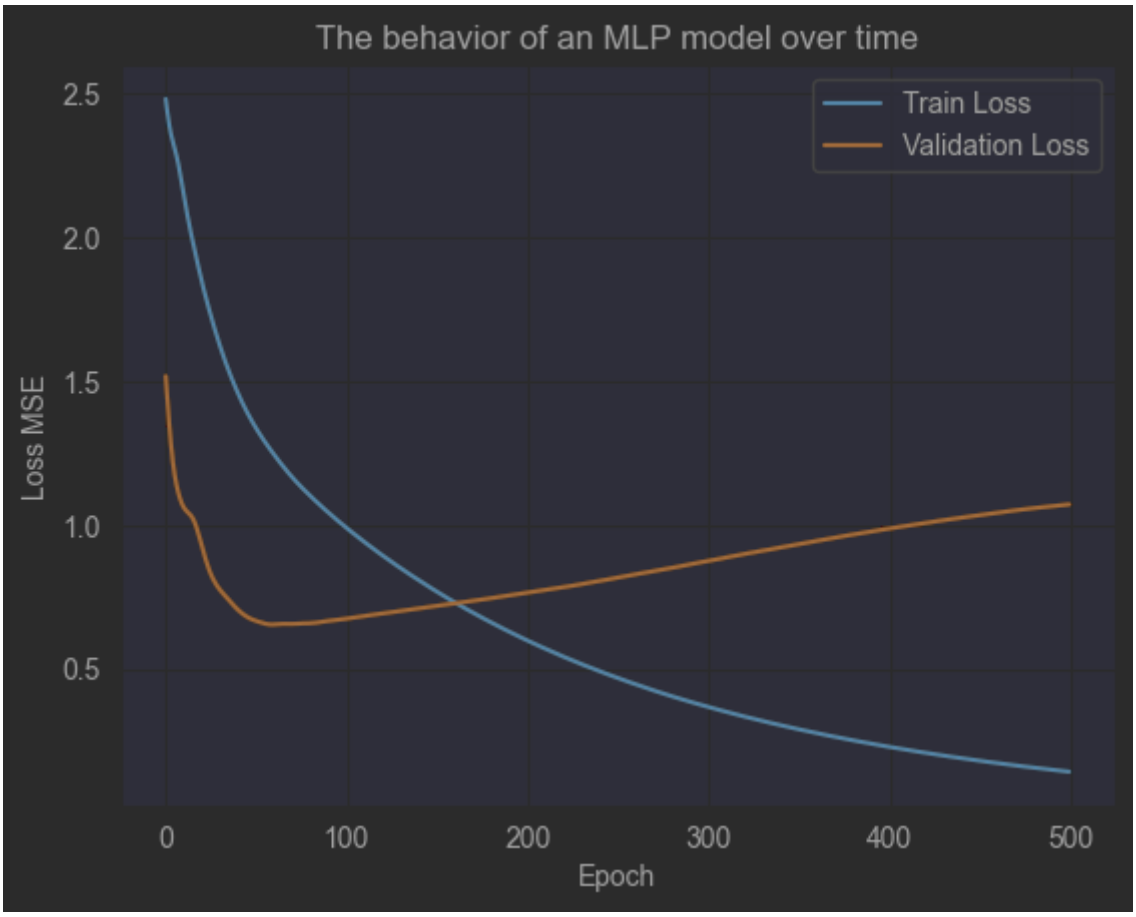
- **solver** (domyślnie 'sgd'): Określa optymalizator używany podczas treningu sieci. 'sgd' oznacza stochastyczną metodę gradientową, która aktualizuje wagi w oparciu o pojedyncze próbki danych treningowych.
- **alpha** (domyślnie 0.0): Parametr regularyzacji, który kontroluje wpływ kary za duże wagi na funkcję kosztu podczas treningu sieci.
- **learning_rate** (domyślnie 'constant'): Określa strategię aktualizacji szybkości uczenia się w trakcie treningu.
- **epochs** (domyślnie 500): Liczba epok treningowych, czyli liczba razy, jaką sieć neuronowa przejdzie przez cały zbiór treningowy podczas treningu.
- **learning_rate_init** (domyślnie 0.001): Początkowa wartość szybkości uczenia się. Określa, jak szybko wagi sieci będą aktualizowane na początku treningu.
- **hidden_layer_sizes** (domyślnie (100,)): Określa architekturę sieci neuronowej, a konkretnie rozmiary ukrytych warstw. Jest to krotka liczb całkowitych, gdzie każda liczba reprezentuje liczbę neuronów w danej ukrytej warstwie. Domyślnie używana jest jedna ukryta warstwa o rozmiarze 100 neuronów.

Następnie sieć trenowana jest przez x epok jest to osiągane przez uczenie modelu za pomocą metody **partial_fit** na modelu. Pozwala to na stopniowe aktualizowanie wag sieci w miarę postępu treningu. Po wykonaniu każdej z iteracji do 2 list zapisywana jest wartość funkcji celu (a konkretniej wartość błędu średnio kwadratowego MSE) z wartości predykcyjnej dla wartości uczących a następnie walidacyjnych.

Jako argumenty zwracana jest cała lista zawierające funkcje celu prze x epok **train_losses, val_losses**

3.Zachowanie modelu MLP w czasie

Podczas monitorowania działania podstawowego modelu MLP o domyślnej konfiguracji hiperparametrów na zbiorze danych Jester, obserwuje się pewne interesujące zjawiska na wykresach wartości funkcji kosztu.

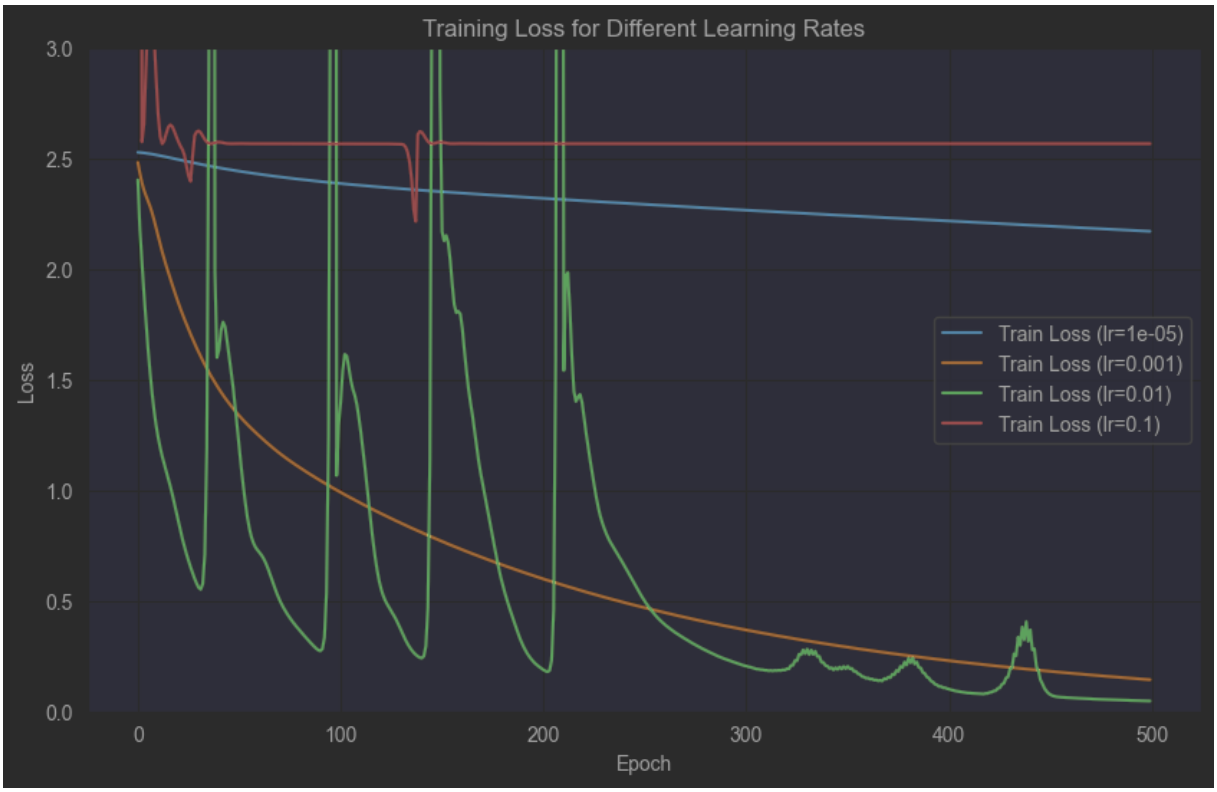


Na wykresie wartości funkcji kosztu na zbiorze uczącym (Train Loss) można zaobserwować ciągły spadek wraz z kolejnymi epokami uczenia. Taki trend wskazuje na coraz lepsze dopasowanie modelu do danych treningowych i redukcję błędu predykcji.

Natomiast na wykresie wartości funkcji kosztu na zbiorze walidacyjnym (Validation Loss) widoczne jest początkowe zmniejszanie się wartości funkcji kosztu, co sugeruje skuteczną generalizację modelu na nowe dane. Niemniej jednak, w pewnym momencie wartość funkcji kosztu na zbiorze walidacyjnym zaczyna delikatnie wzrastać, co może wskazywać na przeuczenie modelu. Może to być spowodowane zbyt długim trwaniem procesu uczenia, co prowadzi do nadmiernego dopasowania modelu do danych uczących.

4.Wpływ tempa uczenia na wyniki

Badając wpływ tempa uczenia (learning rate) na osiągane wyniki, powtórzony został proces uczenia dla czterech różnych wartości tego parametru. Przeanalizowano zachowanie modelu dla poszczególnych wartości tempa uczenia, uwzględniając zarówno wartości funkcji kosztu dla zbioru uczącego, jak i walidacyjnego



Na tym wykresie można zauważyć, że dla wartości tempa uczenia równego 1e-05, wartość funkcji kosztu maleje, ale utrzymuje się na stosunkowo wysokim poziomie. W przypadku wartości 0.001, wartość funkcji kosztu maleje i osiąga bardzo dobre wyniki. Natomiast dla tempa uczenia wynoszącego 0.01, wartość funkcji kosztu maleje na początku, ale później gwałtownie wzrasta, co powtarza się kilkukrotnie. Dla wartości 0.1 również obserwujemy malejącą wartość funkcji kosztu, ale następują gwałtowny wzrost.

Na wykresie wartości funkcji kosztu dla zbioru walidacyjnego widzimy podobne zależności. Dla tempa uczenia równego 1e-05 wartość funkcji kosztu maleje, ale utrzymuje się na wyższym poziomie niż dla innych wartości. Dla wartości 0.001 wartość funkcji kosztu maleje i osiąga bardzo dobre wyniki, ale dla tempa uczenia wynoszącego 0.01 i 0.1 obserwujemy nagłe wzrosty i spadki.

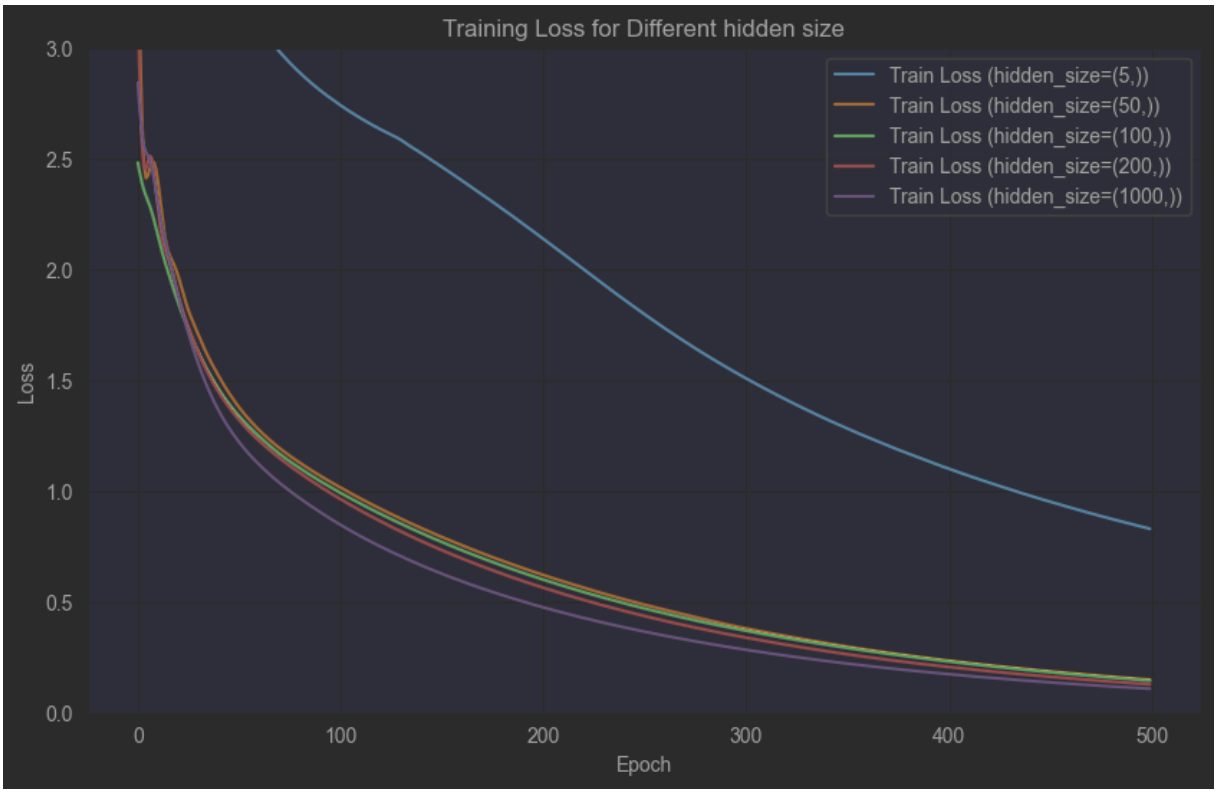
Gdy tempo uczenia jest zbyt niskie (1e-05), model ma trudności w dostatecznym dopasowaniu się do danych treningowych, co prowadzi do wyższych wartości funkcji kosztu. Jeśli learning rate jest zbyt mały, to proces uczenia będzie bardzo wolny. Aktualizacje wag będą miały niewielki wpływ na poprawę wyników sieci, co może prowadzić do długiego czasu uczenia i trudności w osiągnięciu dobrych rezultatów.

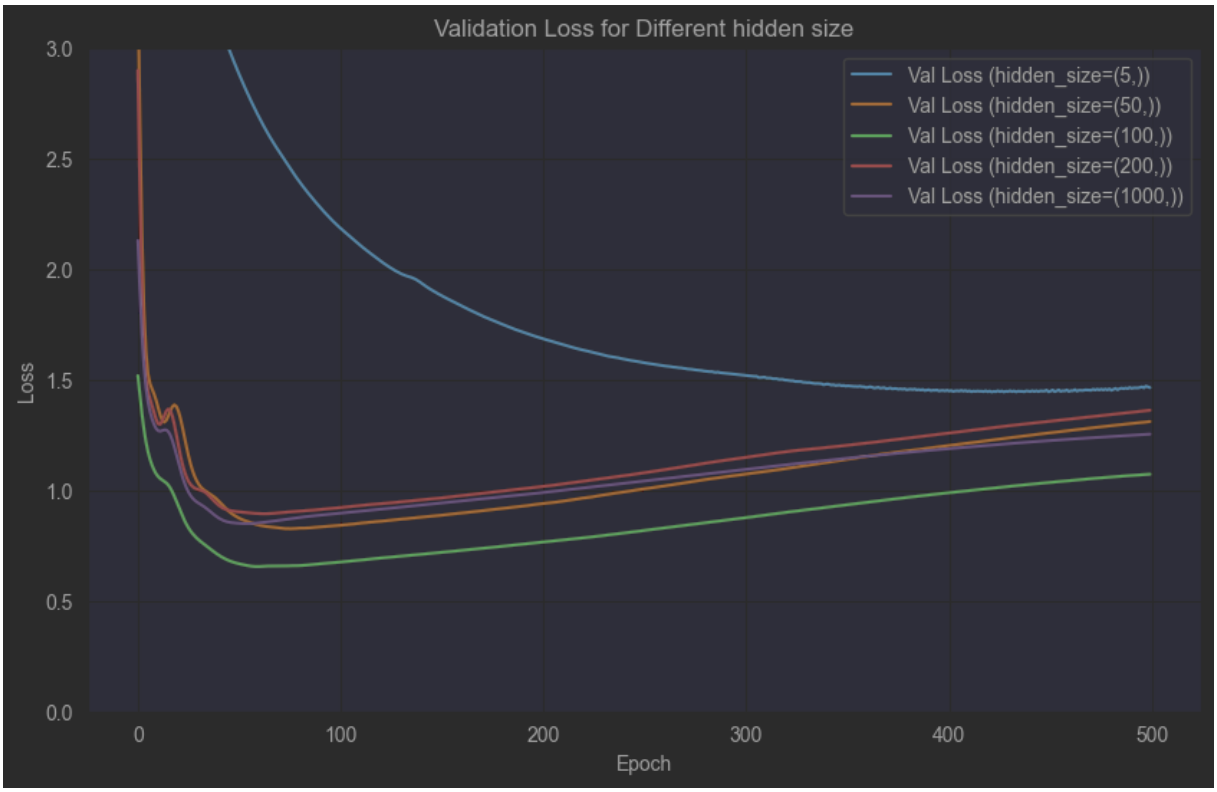
Dla umiarkowanego tempa uczenia (0.001), model osiąga najlepsze wyniki.

Gdy tempo uczenia jest zbyt wysokie (0.01 i 0.1), może to prowadzić do problemów. Wagi sieci mogą oscylować wokół optymalnej wartości, a proces uczenia może się destabilizować (co widzimy na wykresach harmoniczne wartości na wykresach). Może to prowadzić do trudności z osiągnięciem konwergencji lub nawet do rozbieżności, gdzie wartości wag rosną do bardzo dużych wartości i sieć przestaje dawać sensowne wyniki.

5.Wpływ rozmiaru modelu (liczby neuronów)

Aby zbadać wpływ rozmiaru modelu Multi-Layer Perceptron (MLP) na jakość działania, przeprowadzimy 5 eksperymentów, różniące się liczbą neuronów w warstwach ukrytych.





Z wykresu widzimy że model z 5 neuronami w ukrytej warstwie nie radzi sobie dobrze z predykcją w porównaniu z resztą. W takim przypadku model może mieć trudności z wyodrębnieniem złożonych wzorców w danych uczących i nie będzie w stanie nauczyć się wystarczająco skomplikowanych zależności. Występuje tutaj niedouczenie.

Najlepiej radzi sobie model ze 100 neuronami dokonuje on najlepszych predykcji.

W przypadku większej liczby neuronów wyniki predykcji na danych testowych są gorsze. Dzieje się tak dlatego ze model może zapamiętać szumy lub przypadkowe fluktuacje w danych uczących, co prowadzi do doskonałego dopasowania do zbioru uczącego, ale ogólnie nie generalizuje się na nowe dane. Nadmierne dopasowanie można rozpoznać, gdy model ma znacznie lepsze wyniki na zbiorze uczącym niż na zbiorze walidacyjnym. Widać to na 1 wykresie gdzie dane uczące osiąganją najlepsze wyniki przy 1000 neuronów, lecz nie jest to odwzorowane w walidacji.

Wniosek jest taki, że istnieje punkt, w którym model osiąga optymalne dopasowanie do danych. Zbyt mały model będzie niedouczony, a zbyt duży model będzie nadmiernie dopasowany. Optymalny rozmiar modelu MLP zależy od konkretnego problemu, ilości dostępnych danych i złożoności danych uczących.

6.Wybranie najlepszego modelu i testy

Na podstawie eksperymentów wybieram model z 100 neurnoami w warstwie ukrytej jaki z learning_rate 0.001. Uczę model na 100 epokach oraz na całym zbiorze.

```
best_estimator = MLPRegressor(
        solver='sgd',
        alpha=0.0,
        learning_rate='constant',
        learning_rate_init=0.001,
        hidden_layer_sizes=(100,),
        random_state=RANDOM_STATE
)

for _ in range(100):
    best_estimator.partial_fit(embeddings, ratings)
```

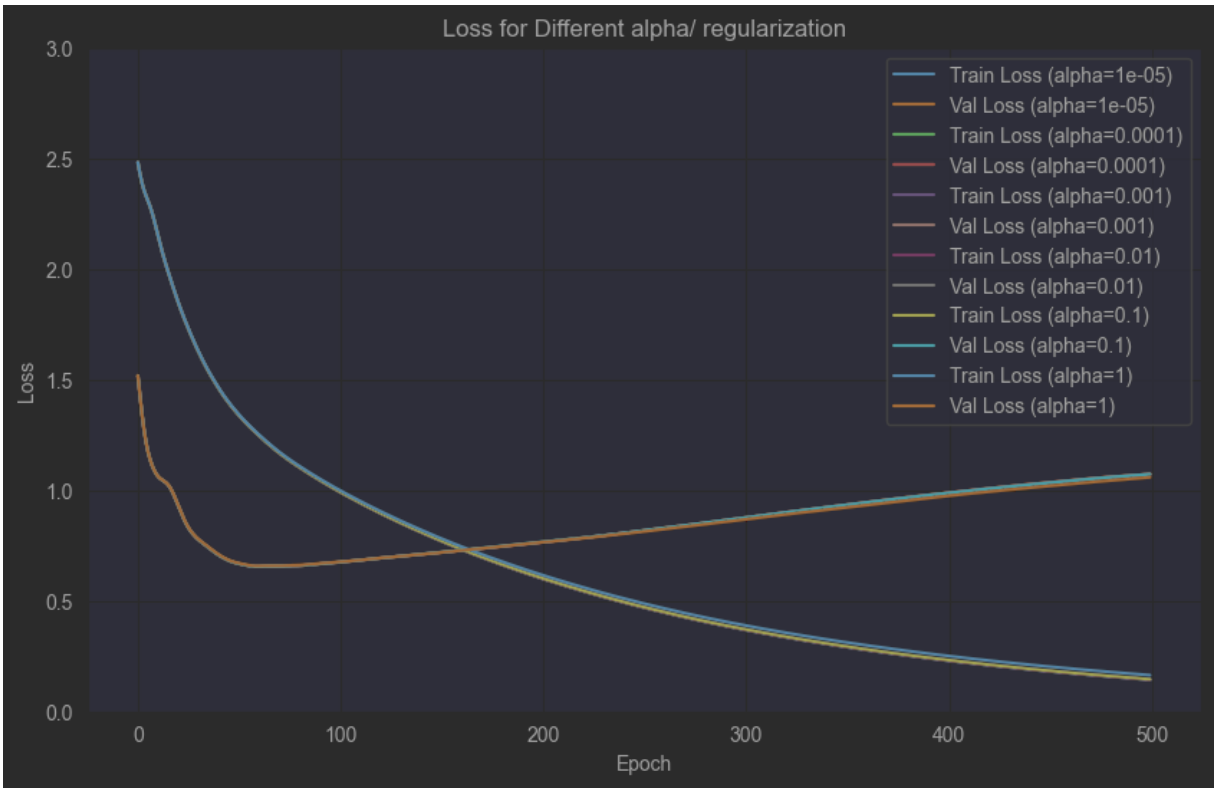
```
Rating -0.44216233491897583: Why don't scientists trust atoms? Because they make up everything!
Rating -0.4597337245941162: Why did the scarecrow win an award? Because he was outstanding in his field!
Rating -0.5650926232337952: I used to play piano by ear, but now I use my hands.
Rating -0.4597337245941162: Why did the scarecrow win an award? Because he was outstanding in his field!
Rating 0.0818813145160675: Why don't you ever see elephants hiding in trees? Because they're really good at it!
Rating -0.18995803594589233: Why don't oysters donate to charity? Because they are shellfish!
Rating 0.25981634855270386: Did you hear about the mathematician who's afraid of negative numbers? He'll stop at nothing to avo
id them!
Rating 1.6610400676727295: Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lore
m ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum Lorem ipsum
Lorem ipsum Lorem ipsum Lorem ipsum
```

Tak wygląda predykcja I nie sprostała ona moim oczekiwanią. Ciekawym faktem godnym zauważenia jest to że im dłuższy jest żart (Lorem ipsum) nie musi mieć sensu tym śmieszniejszy jest według modelu sieci.

7.Wpływ parametru alpha/ regularyzacji

Regularyzacja jest techniką używaną w uczeniu maszynowym w celu zmniejszenia przeuczenia (overfittingu) modelu. Polega na dodaniu kary do funkcji straty, aby zmniejszyć wpływ wag na wynik modelu. W przypadku regularyzacji L2, kara jest proporcjonalna do kwadratu normy wag modelu.

Parametr alfa kontroluje siłę regularyzacji. Im większa wartość alfa, tym większa jest kara nałożona na wagi, co prowadzi do większego ograniczenia modelu. Zbyt duża wartość alfa może spowodować niedouczenie modelu, a zbyt mała wartość alfa może prowadzić do przeuczenia.



W ramach eksperymentów przeprowadzonych w celu badania wpływu parametru alfa na wydajność modelu MLP, zauważono minimalne zmniejszenie wartości funkcji kosztu na zbiorze walidacyjnym przy zwiększającej się wartości alfa. Wartości funkcji kosztu dla różnych wartości alfa różniły się nieznacznie.

Analizując wyniki, można sugerować, że modele MLP używane do badania mogły już osiągnąć niski poziom przeuczenia. Brak znaczącej poprawy wartości funkcji kosztu na zbiorze walidacyjnym wskazuje na to, że parametr alfa nie przynosi istotnych korzyści w kontekście optymalizacji tego modelu.

Warto zauważyć, że ograniczenia zbioru danych mogły mieć istotny wpływ na niewielkie różnice w wynikach dla różnych wartości alfa. Jeżeli zbiór danych jest ograniczony pod względem ilości lub jakości, to zmiany parametru alfa mogą mieć mniejszy wpływ na wydajność modelu.

Solution:
https://github.com/Golden3x11/AI/tree/main/ml