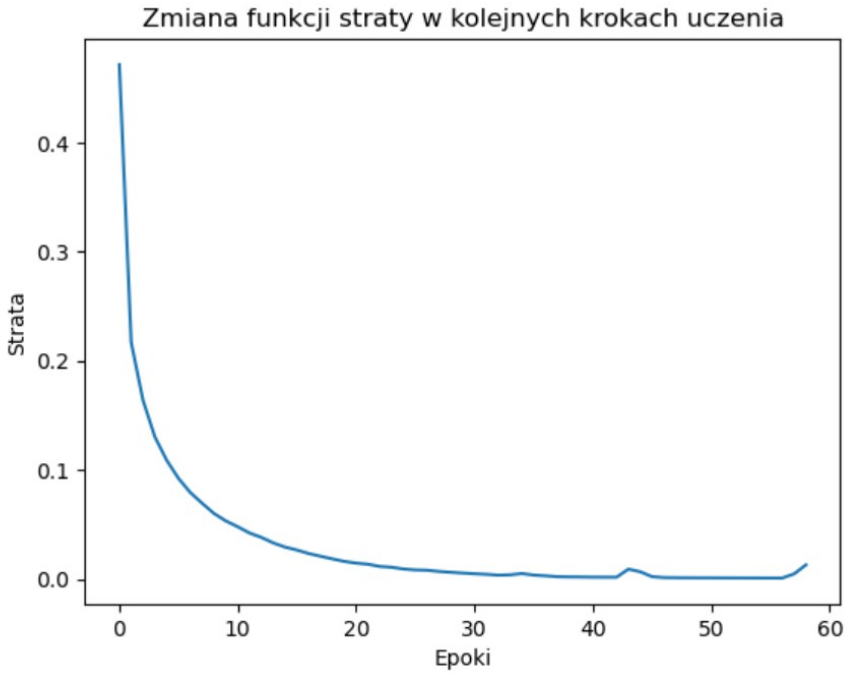
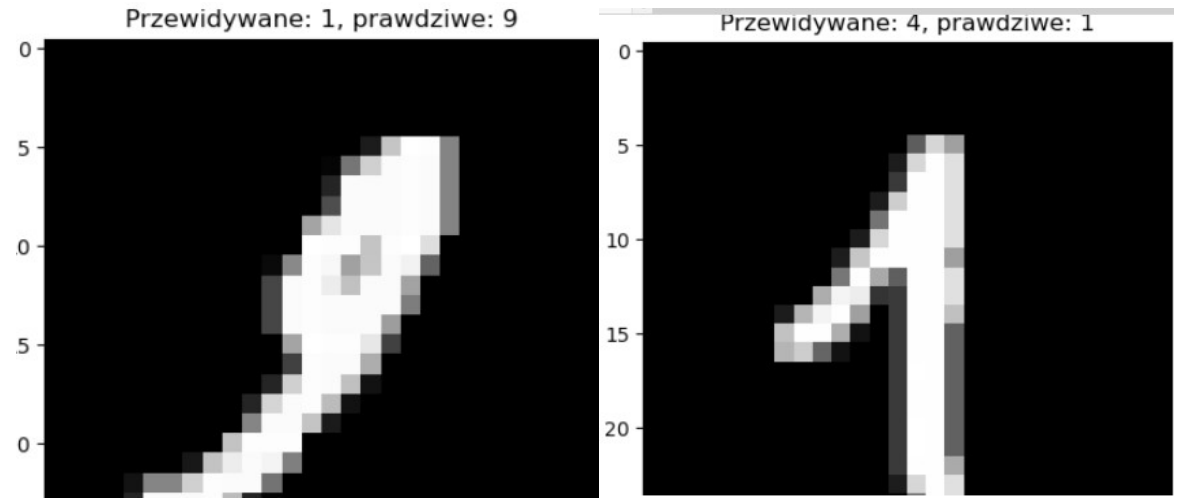
UNIWERSYTET WSB MERITO WROCŁAW	
Laboratorium z Podstaw Uczenia Maszynowego	
Ćwiczenie nr 2	DATA: 23.06.2023
Temat: Sieć jednokierunkowa MLP	
Nazwisko i imię	Robert Jaworski
Nr indeksu	81591
Grupa:	2
Prowadzący	*Prof. dr hab. Urszula Markowska-Kaczmar*

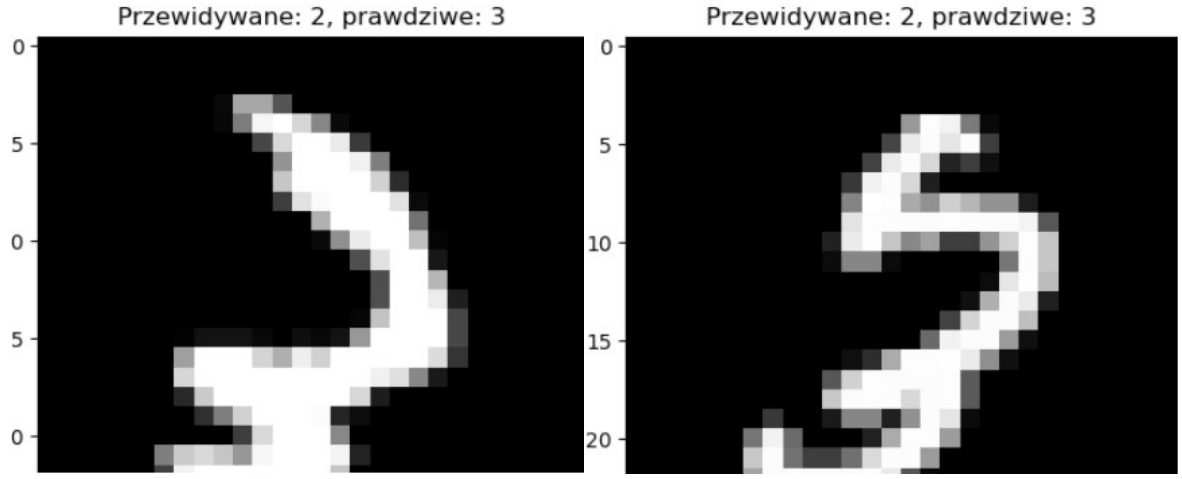
1.Pokazać macierz pomyłek i krzywą uczenia oraz dokładność

```
Dokładność: 0.9746666666666667
Macierz pomyłek:
[[1668    0    8    2    1    3    7    1    8    3]
 [   0 1915    8    1    6    1    2    2    5    1]
 [   4    5 1693    4    3    2    3    5    3    0]
 [   3    1   12 1692    0   19    0    5   19    9]
 [   1    4    8    0 1637    0    5    8    2   17]
 [   4    2    2   13    3 1512    8    1    7    4]
 [   6    2    0    1    3   11 1664    0    7    0]
 [   2    3    7    2    8    2    1 1761    1   10]
 [   3    6    6    9    5    8    7    9 1621    8]
 [   5    0    1    9   21    8    1   10   10 1650]]
```



2.Pokazać błędnie sklasyfikowane wzorce (pierwszych pięć)





Na powyższych przykładach, widać wyraźnie, skąd mogły się pojawić błędy przy rozpoznaniu. Da się zauważyć, że znaczące zniekształcenia wzorców mogą prowadzić do błędnego rozpoznania. Wyjątek może stanowić “1” rozpoznana jako “4”.

3.Wykonać eksperymenty zmieniając wartości domyślne w ustawieniach – w jednym eksperymencie dozwolona zmiana tylko jednego parametru

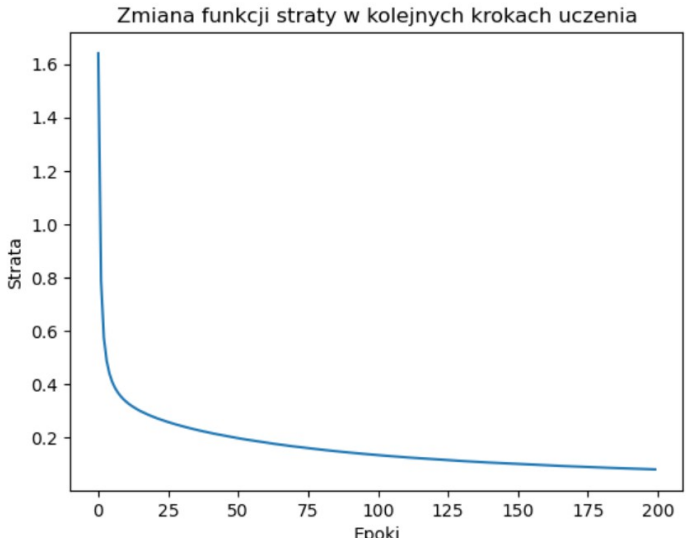
Zmienić sposób uczenia z Adam na SGD – przeanalizować wyniki (dokładność, macierz pomyłek), podać wykres funkcji straty w kolejnych krokach uzasadnić co może być przyczyną takich wyników 10 pkt

Zmiana w kodzie:

```
# 200 iteracji
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='sgd' alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

Wyniki dla solver=’sgd’:

```
Dokładność: 0.9655072463768116
Macierz pomyłek:
[[1656    0    8    4    2   10    7    1    9    4]
 [   1 1911    8    2    5    0    3    2    7    2]
 [   5    5 1674    8    5    2    3    9   10    1]
 [   4    1   22 1666    1   25    0    7   25    9]
 [   3    3    9    1 1623    0   12    3    3   25]
 [  10    4    4   14    2 1492   10    1   10    9]
 [   5    2    0    1    5   16 1659    0    6    0]
 [   3    5    9    0    7    2    1 1759    0   11]
 [   2   18    9   19    7   11    9    8 1590    9]
 [   8    3    1   12   32    5    2   14   13 1625]]
```



SGD przeszedł przez maksymalną ilość epok (200). Wyniki są jednak gorsze od tych z

solverem Adam, który osiągnął lepsze wyniki w krótszym czasie (60 epok).
W przypadku Adam, dobre wyniki osiągane są wcześniej i dalsze uczenie jest przerywane, bo próba poprawy modelu wprowadza na tym etapie jedynie nieznaczne polepszenie.

1. Szybkość uczenia:
    - Adam: Adam automatycznie dostosowuje współczynnik uczenia dla każdego parametru w oparciu o estymację momentu. Może to pomóc w skutecznym uczeniu zróżnicowanych zestawów danych.
    - SGD: SGD korzysta z ustalonego współczynnika uczenia, który jest zdefiniowany na początku procesu uczenia. Może być konieczne doświadczalne strojenie współczynnika uczenia, aby znaleźć optymalną wartość.
2. Zbieżność:
    - Adam: Dzięki adaptacyjnemu dostosowaniu współczynnika uczenia i uwzględnieniu momentu i odchylenia standardowego gradientów, Adam może zbiegać szybciej niż SGD w niektórych przypadkach.
    - SGD: SGD jest prostszy i bardziej deterministyczny w porównaniu do Adam. W niektórych przypadkach może wymagać więcej iteracji, aby osiągnąć zbieżność.
3. Pamięć:
    - Adam: Adam przechowuje informacje o przeszłych momentach i odchyleniach standardowych gradientów, co pozwala na uwzględnienie historycznych zmian podczas aktualizacji wag.
    - SGD: SGD nie przechowuje żadnych historycznych informacji o gradientach. Aktualizacja wag jest wykonywana na podstawie gradientu obliczonego dla bieżącego przykładu.
4. Aktualizacja wag:
    - Adam: Adam wykorzystuje adaptacyjne oszacowanie momentu, które uwzględnia zarówno pierwszy moment (średnia gradientów) jak i drugi moment (odchylenie standardowe gradientów). Korzysta z eksponencjalnie średniej ważonej tych momentów, co pozwala na adaptacyjne dostosowanie współczynnika uczenia dla każdego parametru.
    - SGD: SGD aktualizuje wagi na podstawie gradientu funkcji kosztu dla pojedynczych przykładów treningowych. W każdej iteracji losowo wybierany jest pojedynczy przykład, a aktualizacja wag jest wykonywana w oparciu o gradient dla tego przykładu.

Adam jest popularny ze względu na swoją adaptacyjność i skuteczność w wielu przypadkach, ale SGD nadal może być użyteczny w niektórych scenariuszach, szczególnie przy dużych zbiorach danych. Ostateczny wybór powinien być oparty na eksperymentach i porównaniu wyników dla konkretnego zadania.

4.Zmienić liczbę neuronów w warstwie

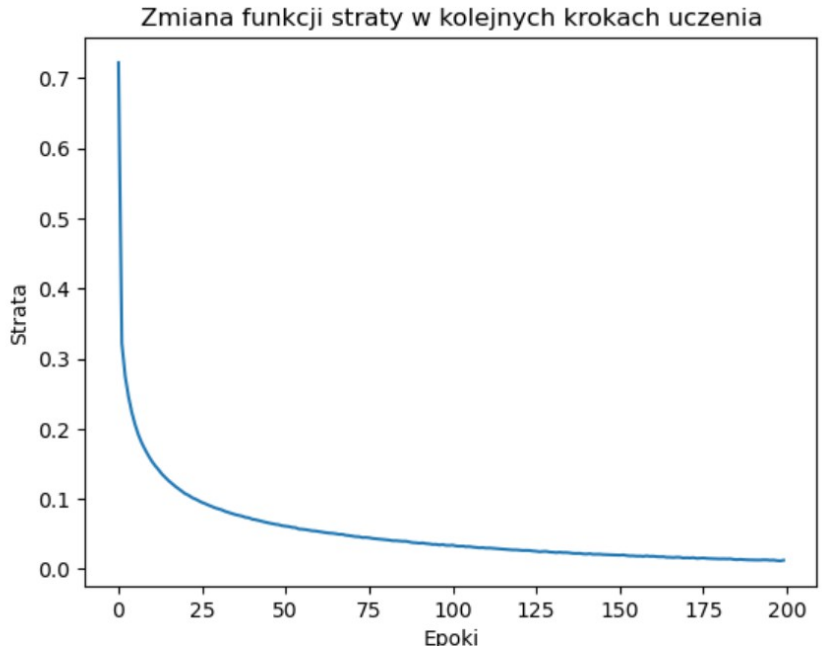
Wracamy do parametru solver=”Adam”

```
mlp = MLPClassifier(hidden_layer_sizes=(20,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

a) Dużo mniejszą ilość neuronów: **20** neuronów

Model uczył się przez maksymalną ustawioną ilość epok - 200

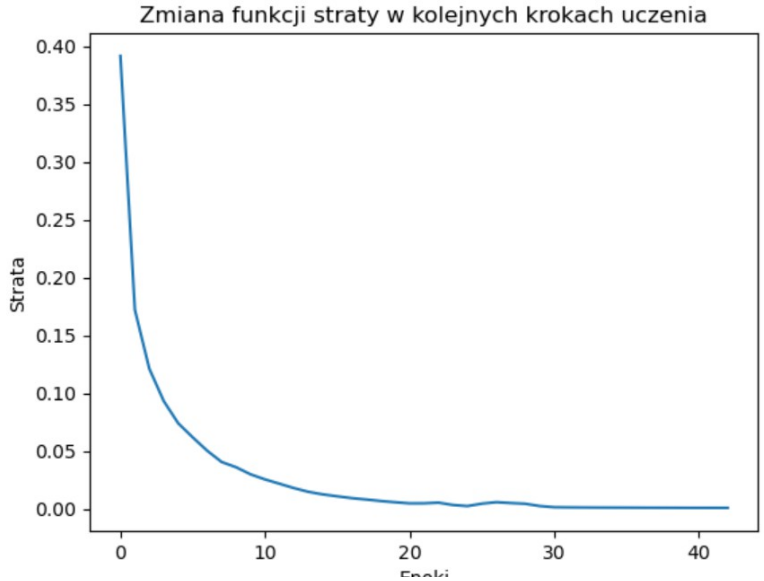
```
Dokładność: 0.9473623188405798
Macierz pomyłek:
[[1642    0    9    6    2   12   10    3   14    3]
 [   0 1907    7    2    4    3    4    4    8    2]
 [  12    6 1630   19    6    5    6   15   21    2]
 [   4    2   39 1598    1   58    0   17   30   11]
 [   3    4   11    3 1581    4   14   12   14   36]
 [   4    5    4   24    6 1454   18    6   22   13]
 [   6    4    3    1   12   13 1639    0   15    1]
 [   6    6    5    6   11    5    0 1722    5   31]
 [   6   16   13   17   11   16    7    6 1573   17]
 [   5    0    1   13   45   18    1   19   17 1596]]
```



b) Dużo większą ilość neuronów: **200** neuronów

Ilość epok - 43

```
Dokładność: 0.9790144927536232
Macierz pomyłek:
[[1671    1    6    1    0    1    7    4    7    3]
 [   0 1921    5    2    4    0    3    2    4    0]
 [   6    4 1697    6    2    0    2    2    3    0]
 [   2    2   11 1704    0   15    0    6   13    7]
 [   1    5    5    0 1637    0    8    5    2   19]
 [   5    0    2   12    1 1515    4    3    8    6]
 [   4    1    0    1    3    9 1671    0    5    0]
 [   2    3    3    0    2    1    1 1776    1    8]
 [   1    8    6    6    3    6    6    8 1633    5]
 [   3    0    1    5   17    7    3    7    9 1663]]
```



Widać, że niewielka ilość warstw w przykładzie a) wymusza na modelu wykorzystanie większej ilości epok do nauki przez co wykorzystuje maksymalną dozwoloną przez nas ich ilość - 200. Model ze zwiększoną ilością warstw (200) wyuczył się szybciej, bo w ciągu 43 epok i przerwał dalszą naukę, która nie wprowadzała już znacznej poprawy.

Porównując to do wyników z pierwszego zdania widać, że zwiększenie ilości warstw poprawia dokładność modelu i zmniejsza ilość potrzebnych epok do wyuczenia. Obserwowana poprawa dokładności wynosi tutaj 0.005.

5. Przy ustalonej liczbie neuronów włączyć momentum

Momentum jest techniką optymalizacji używaną w algorytmach uczenia maszynowego, w tym w sieciach neuronowych. Jest to dodatkowy parametr, który wpływa na proces aktualizacji wag podczas uczenia modelu.

Przez całą długość zadania parametry były modelu były ustawione wedle polecenia. Momentum jest aktywne, gdy jego wartość wynosi więcej niż zero, domyślnie jest ustawione na 0.9 . Momentum aktywne jest tylko dla solver=”sgd”. To znaczy, że przy każdym badaniu z solverem SGD momentum było aktywne.

momentum : *float, default=0.9*

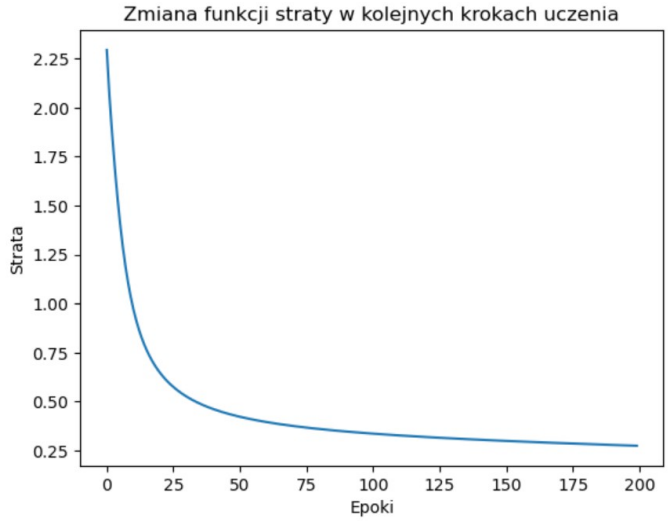
Momentum for gradient descent update. Should be between 0 and 1. Only used when solver='sgd'.

źródło: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Momentum = 0

```
# 200 iteracji
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='sgd', alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.0, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

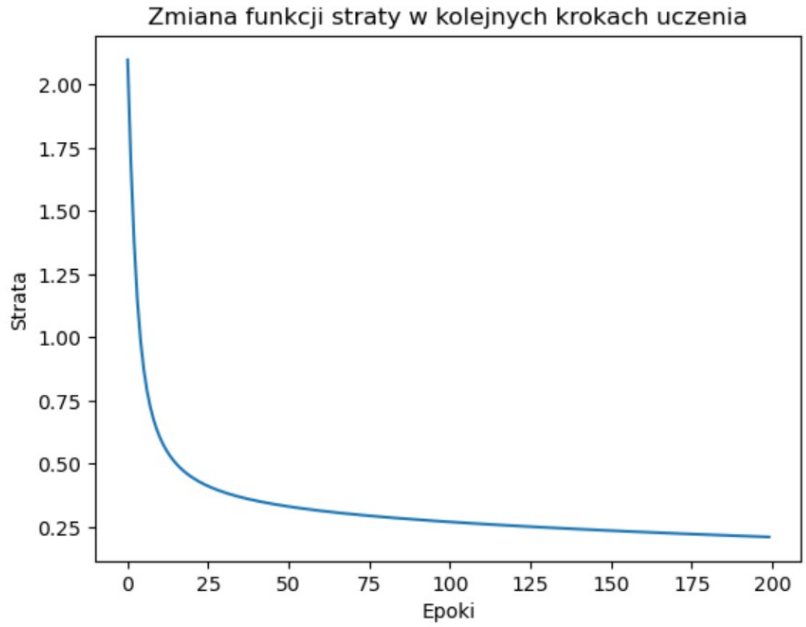
```
Dokładność: 0.9187826086956522
Macierz pomyłek:
[[1635    0    6    7    3   17   10    3   19    1]
 [   1 1882   10    8    3    2    4    6   21    4]
 [  11   12 1553   28   28    2   26   21   33    8]
 [  12    4   38 1560    1   60    7   22   36   20]
 [   5    6   13    3 1558    1   19    4   17   56]
 [  18    8   13   58   21 1356   21    7   40   14]
 [   9    6   16    0   13   30 1610    0   10    0]
 [  12   11   25    5   16    3    0 1686    4   35]
 [   9   30   20   34    7   39   18    8 1486   31]
 [  15    6    5   25   67   10    2   45   17 1523]]
```



Momentum = 0.5

```python
# 200 iteracji
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='sgd', alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.5, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

```
Dokładność: 0.9362898550724638
Macierz pomyłek:
[[1646    0    2    6    4   11   10    5   15    2]
 [   1 1895   10    4    2    3    3    3   17    3]
 [   8   10 1601   23   24    1    9   18   21    7]
 [  11    4   33 1600    1   43    5   16   30   17]
 [   5    7   11    1 1573    1   15    6   10   53]
 [  17    4    9   34   15 1423   18    4   23    9]
 [   9    4    6    0   12   29 1626    0    7    1]
 [   9   10   18    4   14    2    0 1720    1   19]
 [   5   28   17   30   10   32   18   12 1507   23]
 [  12    7    2   22   55    8    2   33   14 1560]]
```



Wprowadzenie momentum ma kilka korzyści. Po pierwsze, pomaga przyspieszyć proces uczenia, zwłaszcza gdy gradienty mają ciągłość i są zgodne w kolejnych iteracjach. Po drugie, momentum może pomóc w uniknięciu utknięcia w lokalnym minimum, ponieważ umożliwia modelowi "przeskakiwanie" przez pewne płaskie obszary w funkcji kosztu.

Widać, że obecność momentum poprawia dokładność modelu, a im większa wartość tym lepsza dokładność.

W praktyce, momentum w algorytmach uczenia maszynowego jest implementowane jako ważona suma aktualizacji wag z poprzednich iteracji. W każdej iteracji, aktualizacja wag jest dodawana do tej ważonej sumy, co daje efekt "momentum", który sprawia, że aktualizacje wag kierują się w kierunku, który był preferowany w poprzednich iteracjach.

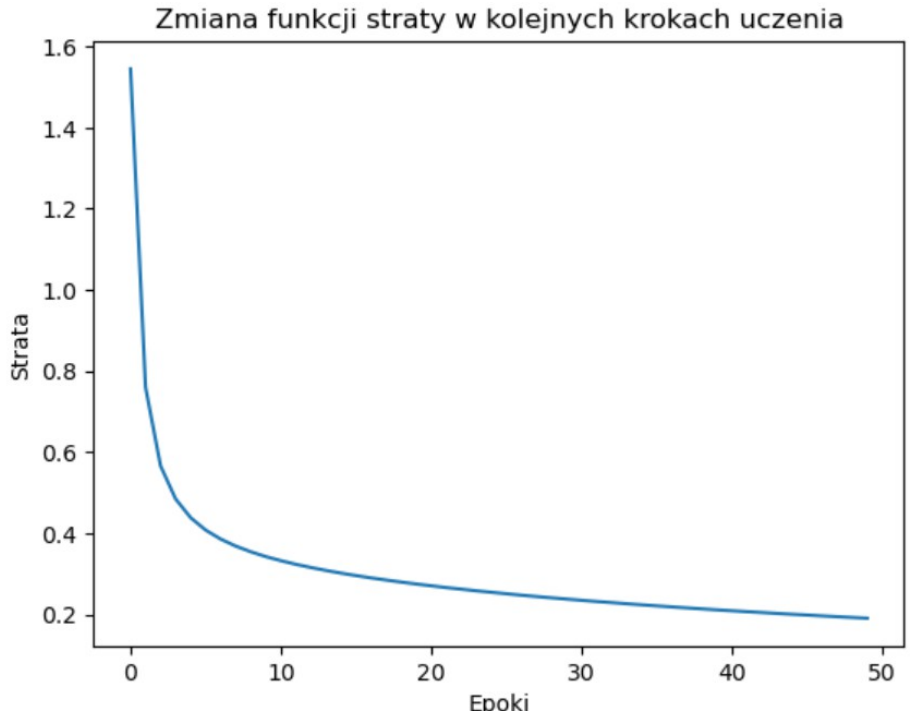
5.2. Zbadać wynik uczenia w zależności od maksymalnego czasu uczenia

Według części pierwszej momentum ma być włączone do tego zadania co wymusza używanie SGD wedle wcześniej zdobytych i przedstawionych informacji

a) max_iter na 50

```
# 200 iteracji
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='sgd', alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=50, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

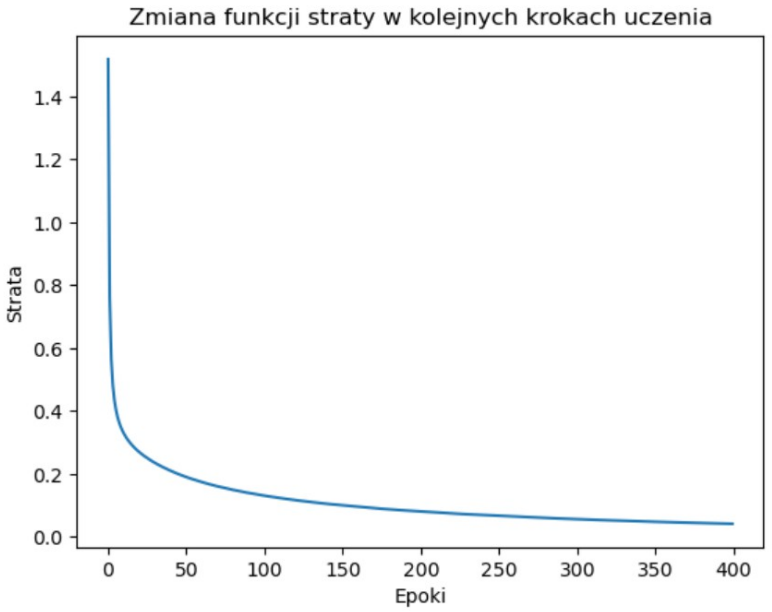
```
Dokładność: 0.9411594202898551
Macierz pomyłek:
[[1649    0    4    6    4   12    8    2   13    3]
 [   1 1904    9    4    2    1    3    4   10    3]
 [   7   10 1616   20   18    6    6   15   20    4]
 [  12    5   29 1597    1   47    4   16   32   17]
 [   4    6   11    0 1578    4   17    5    9   48]
 [  18    6    8   31   11 1433   20    4   16    9]
 [  10    3    4    1    9   27 1634    0    6    0]
 [   8    8   15    4   12    1    0 1731    1   17]
 [   4   24   13   32    9   30   14   11 1527   18]
 [  11    8    3   23   50   10    2   28   14 1566]]
```



b) max_iter na 400

```python
# 200 iteracji
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='sgd', alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=400, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

```
Iteration 400, loss = 0.04199513
Dokładność: 0.9721739130434782
Macierz pomyłek:
[[1663    1    4    4    3    5    7    3    6    5]
 [   1 1916    7    0    5    0    3    2    5    2]
 [   7    3 1687    7    2    1    3    6    5    1]
 [   3    0   14 1681    0   27    0    7   21    7]
 [   2    3    8    1 1632    1   11    5    2   17]
 [   6    4    6    9    2 1510    3    0    8    8]
 [   4    2    1    2    6    6 1665    0    8    0]
 [   3    4    3    1    3    4    1 1767    3    8]
 [   3   10    5   13    6   16   10    8 1603    8]
 [   6    1    1    9   23    6    1   11   11 1646]]
```



Na podstawie przeprowadzonego eksperymentu, badającego wynik uczenia w zależności od maksymalnego czasu uczenia, można wyciągnąć kilka wniosków:

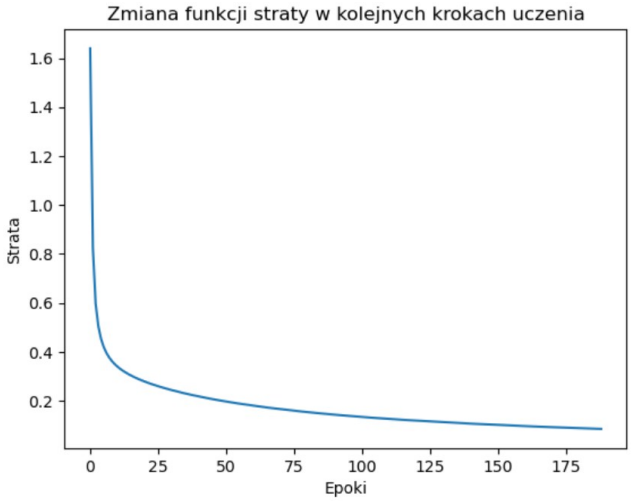
1. Wzrost maksymalnego czasu uczenia ma tendencję do zwiększania dokładności modelu. Oznacza to, że dłuższy czas uczenia pozwala modelowi na lepsze dopasowanie do danych i osiągnięcie wyższej dokładności predykcji.
2. Wartość maksymalnego czasu uczenia ma wpływ na osiągane rezultaty. Dla krótkiego czasu uczenia, model może nie zdążyć nauczyć się wystarczająco dużo informacji i osiągnąć niższą dokładność. Z kolei dla zbyt długiego czasu uczenia istnieje ryzyko przeuczenia modelu na zbiór treningowy, co może prowadzić do obniżenia dokładności na zbiorze testowym.
3. Istnieje punkt, w którym dalsze zwiększanie maksymalnego czasu uczenia nie przynosi znaczącego wzrostu dokładności. Może to sugerować, że model osiągnął już swoją maksymalną wydajność i dalsze uczenie nie przynosi istotnych korzyści.

4. Optymalny czas uczenia może zależeć od specyfiki problemu i zbioru danych. Nie ma uniwersalnej wartości, która będzie działała najlepiej dla każdego przypadku.

5.3. Zbadać wynik uczenia w zależności od hiperparametru early_stopping

```
# 200 iteracji
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='sgd', alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=True, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

```
Iteration 184, loss = 0.08761261
Validation score: 0.964058
Iteration 185, loss = 0.08726078
Validation score: 0.964444
Iteration 186, loss = 0.08685413
Validation score: 0.965024
Iteration 187, loss = 0.08651087
Validation score: 0.964251
Iteration 188, loss = 0.08608384
Validation score: 0.964058
Iteration 189, loss = 0.08577435
Validation score: 0.965024
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Dokładność: 0.9637681159420289
Macierz pomyłek:
[[1661    0    6    4    2    5    9    3    7    4]
 [   1 1910   11    1    4    0    4    3    6    1]
 [   5    6 1676    8    6    0    2    9    6    4]
 [   6    4   26 1656    0   23    1   10   23   11]
 [   3    5   10    2 1607    0    9    6    4   36]
 [  11    2    4   17    3 1488    9    1   11   10]
 [   5    2    1    1    7   12 1662    0    4    0]
 [   4    4    9    1    7    3    3 1752    1   13]
 [   1   10    9   19    8   11   13    9 1586   16]
 [   7    4    2   10   29    5    2   17   12 1627]]
```



1. Dokładność modelu z early_stopping jest porównywalna z dokładnością modelu bez early_stopping. Hiperparametr early_stopping pozwala na zatrzymanie uczenia, gdy nie następuje poprawa wyników na zbiorze walidacyjnym, co może zapobiec przeuczeniu.

2. Użycie early_stopping może prowadzić do wcześniejszego zakończenia procesu uczenia, co może przyspieszyć trening modelu. Dzięki zatrzymaniu uczenia, gdy nie ma dalszej znacznej poprawy, nie ma potrzeby kontynuowania iteracji przez pełną liczbę epok.

Warto zauważyć, że wpływ early_stopping może zależeć od konkretnego zadania i danych. Należy eksperymentować z różnymi wartościami tego hiperparametru oraz innymi parametrami modelu, aby znaleźć optymalną konfigurację dla konkretnego problemu.

Aktywacja early stopping wyświetla też dokładność modelu dla każdej z epok.

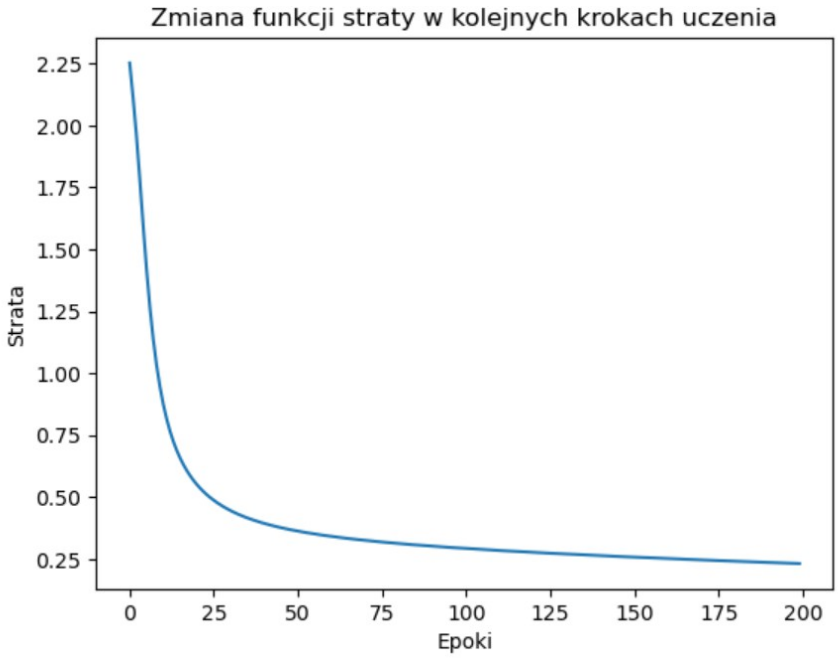
early_stopping : ***bool, default=False***

Whether to use early stopping to terminate training when validation score is not improving. If set to true, it will automatically set aside 10% of training data as validation and terminate training when validation score is not improving by at least tol for `n_iter_no_change` consecutive epochs. The split is stratified, except in a multilabel setting. If early stopping is False, then the training stops when the training loss does not improve by more than tol for n_iter_no_change consecutive passes over the training set. Only effective when solver='sgd' or 'adam'.

5.4. Zmień funkcję aktywacji z domyślnej na sigmoidalną. Na podstawie otrzymanych wyników badań określ zauważone różnice.

```python
# 200 iteracji
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='logistic', solver='sgd', alpha=0.0001, batch_size='auto', learning_rate='constant',
                    learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=True,
                    warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9,
                    beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

```
Dokładność: 0.9298550724637681
Macierz pomyłek:
[[1644    0    4    5    2   14   10    4   16    2]
 [   0 1881   11    7    3    5    4    5   23    2]
 [   9   11 1592   22   24    1   11   17   29    6]
 [   8    5   33 1588    1   49    7   17   37   15]
 [   5    7   11    3 1560    1   24    1   12   58]
 [  24    6    6   44   19 1386   23    3   29   16]
 [  10    5    9    0   10   34 1620    0    6    0]
 [   9    8   16    5   14    3    0 1708    3   31]
 [   5   26   16   32    7   37   17    9 1511   22]
 [  14    7    4   23   54    9    2   36   16 1550]]
```



Zauważyć można że wybranie funkcji sigmoidalnej pogorszyło dokładność modelu w porównaniu do dokładnie tego samego modelu ale z funkcją linearną. Może to oznaczać, że funkcja sigmoidalna dla badanego problemu będzie gorszym wyborem, dającym gorsze wyniki.

1. Sigmoidalna funkcja aktywacji (np. sigmoid) ma ograniczony zakres wartości wyjściowych (od 0 do 1), co może wpływać na dynamikę uczenia. Może to oznaczać, że model będzie wolniej się uczył, zwłaszcza na początku, gdy wagi są inicjalizowane losowo.
2. Sigmoidalna funkcja aktywacji jest szczególnie przydatna w przypadku problemów binarnej klasyfikacji, gdzie oczekuje się wartości wyjściowej w zakresie 0-1. Jeśli zadanie klasyfikacji jest wieloklasowe, konieczne może być zastosowanie technik kodowania one-hot (np. funkcja softmax) lub innych funkcji aktywacji odpowiednich dla danego problemu.
3. Zmiana funkcji aktywacji może wpływać na sposób, w jaki model reprezentuje dane i podejmuje decyzje. Sigmoidalna funkcja aktywacji jest nieliniowa, co pozwala modelowi na wykrywanie nieliniowych zależności w danych. Jednak może również występować problem zanikającego gradientu, szczególnie dla głębokich sieci neuronowych.
4. Przy zmianie funkcji aktywacji na sigmoidalną istnieje również konieczność dostosowania innych hiperparametrów modelu, takich jak szybkość uczenia, rozmiar warstw ukrytych itp., aby zapewnić odpowiednie i efektywne uczenie.

6.Porównać wyniki z tymi z regresji logistycznej

Model regresji logistycznej

max_iter=	Dokładność modelu
100	0.92
200	0,9196

Regresja logistyczna może być bardziej efektywna dla prostych problemów klasyfikacji binarnej, szczególnie gdy dane są liniowo separowalne. Model ten może osiągnąć wysoką dokładność klasyfikacji przy stosunkowo mniejszej złożoności obliczeniowej.

Regresja logistyczna jest interpretowalna, ponieważ zapewnia miarę istotności zmiennych (w postaci współczynników) i interpretację wyników jako prawdopodobieństwa przynależności do danej klasy. Jest to przydatne, jeśli istnieje potrzeba zrozumienia, jakie zmienne mają wpływ na wynik.

W przypadku danych, dla których nie występuje liniowa separowalność, modele oparte na sieciach neuronowych, które wykorzystują nieliniowe funkcje aktywacji, mogą osiągnąć lepsze wyniki niż regresja logistyczna. Sieci neuronowe są w stanie modelować bardziej złożone zależności między zmiennymi, co może prowadzić do lepszej wydajności klasyfikacji.

Regresja logistyczna może być bardziej podatna na niedopasowanie (underfitting) lub nadmiarowość (overfitting) w przypadku bardziej złożonych danych lub problemów wieloklasowych. Sieci neuronowe, szczególnie te z warstwami ukrytymi, mogą lepiej radzić sobie z elastycznym dopasowaniem do różnych wzorców i zwiększeniem zdolności do generalizacji.

Porównując wyniki z regresją logistyczną widać, że przy podobnej ilości epok dokładność modelu regresji jest gorsza. Dokładność modelu używającego neurony sięgała wyniku 0.97
Jest to różnica rzędu 0.05 (5%) dająca nam model o świetnej dokładności.

7.WNIOSKI

Na podstawie przeprowadzonego ćwiczenia możemy wyciągnąć następujące wnioski:

Sieci neuronowe, zwłaszcza te z warstwami ukrytymi, są zdolne do nauki złożonych

zależności w danych i mogą osiągnąć wysoką dokładność klasyfikacji. W tym konkretnym przypadku, model MLP zastosowany do zbioru danych MNIST osiągnął wynik dokładności na poziomie około 98%, co jest bardzo dobrą wartością.

Wybór optymalnych hiperparametrów ma duże znaczenie dla skuteczności uczenia sieci neuronowych. Przykładowe hiperparametry, takie jak liczba warstw ukrytych, liczba neuronów w warstwach ukrytych, współczynnik uczenia czy maksymalna liczba iteracji, mogą mieć istotny wpływ na wyniki uczenia. Ważne jest przeprowadzenie eksperymentów i optymalizacja tych parametrów, aby uzyskać najlepsze rezultaty.

Dobór funkcji aktywacji ma wpływ na zachowanie i zdolność modelu do nauki. W przypadku tego zadania, funkcja aktywacji ReLU (Rectified Linear Unit) wydaje się być lepszym wyborem niż sigmoidalna funkcja aktywacji, ponieważ osiągnęła wyższą dokładność klasyfikacji.

Wprowadzenie hiperparametru early_stopping może przyczynić się do poprawy wydajności modelu i zmniejszenia ryzyka przeuczenia. W tym ćwiczeniu, włączenie early_stopping prowadziło do zatrzymania procesu uczenia w momencie, gdy model przestał osiągać dalsze poprawki, co mogło przyczynić się do poprawy generalizacji i uniknięcia przeuczenia.

Ważne jest, aby podkreślić, że wnioski te są oparte na konkretnym zadaniu klasyfikacji MNIST i konkretnych ustawieniach hiperparametrów. Wyniki i wnioski mogą się różnić w zależności od innych zbiorów danych, różnych problemów klasyfikacji i różnych konfiguracji modelu. Przeprowadzanie badań porównawczych i eksperymentów jest kluczowe w celu znalezienia optymalnego modelu dla konkretnego zadania.