Projekt

Algorytmy uczenia maszynowego

Dokumentacja

Projekt 1 – perceptron Rosenblatta

Autor:
Krystian Cyga, 259247

Termin: śr11:15

Prowadzący:
Mgr inż. Michał Zmonarski

27 marca 2024

Spis treści

1 Opis projektu

Perceptron Rosenblatta to jedna z pierwszych form sztucznych neuronów opracowanych przez Franka Rosenblatta w latach 50. i 60. XX wieku. Jest to podstawowy model sztucznej jednostki obliczeniowej, inspirowany biologicznymi neuronami w ludzkim mózgu. Perceptron przyjmuje wejścia, przetwarza je i generuje odpowiedź na podstawie pewnej funkcji aktywacji.

Perceptron Rosenblatta to jedna z pierwszych form sztucznych neuronów opracowanych przez Franka Rosenblatta w latach 50. i 60. XX wieku. Jest to podstawowy model sztucznej jednostki obliczeniowej, inspirowany biologicznymi neuronami w ludzkim mózgu. Perceptron przyjmuje wejścia, przetwarza je i generuje odpowiedź na podstawie pewnej funkcji aktywacji.

Perceptron Rosenblatta przyjmuje wejścia $x_1, x_2, \ldots, x_n$ oraz ma wagi $w_1, w_2, \ldots, w_n$, które nadają wagę każdemu z wejść. Suma iloczynów wejść i wag jest przekazywana do funkcji aktywacji, która decyduje, czy neuron ma zostać aktywowany czy nie. Matematycznie, wyjście y perceptronu można zapisać jako:

$$y = \text{activation}\left(\sum_{i=1}^{n} w_i \cdot x_i\right)$$

W przypadku perceptronu Rosenblatta funkcją aktywacji jest funkcja skokowa (np. funkcja Heaviside'a), która zwraca 1, jeśli suma przekracza pewną progową wartość, a w przeciwnym razie zwraca 0. Wzór funkcji aktywacji skokowej:

$$\text{activation}(x) = \begin{cases} 1, & \text{jeśli } x > \text{threshold} \\ 0, & \text{w przeciwnym przypadku} \end{cases}$$

Można również użyć innej funkcji aktywacji, na przykład sigmoidalnej funkcji logistycznej:

$$\text{activation}(x) = \frac{1}{1 + e^{-x}}$$

Problem XOR (Exclusive OR) jest to problem logiczny, który ma za zadanie zwrócić prawdę (1), gdy jedno z dwóch wejść jest prawdziwe, ale nie oba, a w przeciwnym przypadku zwraca fałsz (0). Jest to problem nieliniowy, ponieważ nie można go rozwiązać przy użyciu jednej linii oddzielającej obie klasy (0 i 1) w przestrzeni wejść.

Tabela prawdy dla operacji XOR:

Wejście A	Wejście B	Wynik XOR
0	0	0
0	1	1
1	0	1
1	1	0

1.1 Opis zaimplementowanych funkcji:

- **Activation(x)**: Funkcja aktywacji, która przyjmuje argument x i zwraca 1, jeśli x jest większe lub równe 0, w przeciwnym razie zwraca 0. Wykorzystywana jest do przewidywania wyników na podstawie sumy wag i danych wejściowych.
- **Predict(inputs)**: Metoda służąca do przewidywania wyników dla danych wejściowych inputs. Wykorzystuje funkcję aktywacji do obliczenia przewidywanych wartości na podstawie sumy iloczynu wag i danych wejściowych.
- **Train(trainingInputs, labels)**: Metoda odpowiedzialna za uczenie perceptronu. Iteruje po zestawie danych treningowych trainingInputs i ich etykietach labels, aktualizując wagi przy użyciu algorytmu gradient descent.
- **Liczenie błędu MSE**: Błąd średniokwadratowy to miara błędu, która często jest używana do oceny wydajności modelu w problemach regresji.

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

·MSE oznacza błąd średniokwadratowy
·n to liczba próbek w zbiorze danych
·y_i to rzeczywista wartość (etykieta) dla i-tej próbki
·$\hat{y}_i$ to wartość przewidywana przez model dla i-tej próbki

- **Błąd klasyfikacji**: Błąd klasyfikacji odnosi się do stopnia niezgodności między oczekiwaną wartością a wartością przewidywaną przez perceptron. W przypadku zadania XOR, gdzie mamy cztery przykłady wejściowe i odpowiadające im etykiety wyjściowe, błąd klasyfikacji mówi nam, jak wiele przykładów zostało źle sklasyfikowanych przez perceptron w stosunku do oczekiwanych etykiet.

- **Momentum**: Momentum to technika używana do przyspieszenia procesu uczenia się poprzez wykorzystanie informacji o poprzednich aktualizacjach wag. Momentum pomaga uniknąć utknięcia w lokalnych minimach poprzez zachowanie ruchu, co pozwala na bardziej skuteczne przeszukiwanie przestrzeni rozwiązań.

$$\Delta w_t = \alpha \Delta w_{t-1} - \eta \nabla J(w_t)$$

Δw_t - zmiana wag w aktualnej iteracji
α - współczynnik momentum
Δw_{t-1} - zmiana wag z poprzedniej iteracji
η - współczynnik uczenia (learning rate)
$\nabla J(w_t)$ - gradient funkcji straty w aktualnym położeniu wag

- **Adaptacyjny współczynnik uczenia**: To technika, która dostosowuje wartość współczynnika uczenia w trakcie procesu uczenia w zależności od aktualnych warunków. Celem adaptacyjnego współczynnika uczenia jest zapewnienie optymalnej szybkości uczenia się w różnych etapach procesu uczenia, co może pomóc w przyspieszeniu zbieżności i poprawieniu wydajności modelu. Istnieje wiele różnych metod adaptacyjnego dostosowywania współczynnika uczenia, takich jak AdaGrad, RMSprop, Adam, itp. Każda z tych metod ma swój własny sposób obliczania wartości współczynnika uczenia w zależności od aktualnego gradientu i poprzednich aktualizacji wag.

  Przykładowo, w metodzie AdaGrad, adaptacyjny współczynnik uczenia jest zdefiniowany jako:

$$\eta_t = \frac{\eta}{\sqrt{G_t + \epsilon}}$$

  η_t - adaptacyjny współczynnik uczenia w iteracji t,
  η - początkowa wartość współczynnika uczenia,
  G_t - macierz diagonalna, której elementy to suma kwadratów gradientów dla każdej wagi do iteracji t,
  ϵ - mała wartość dodana dla stabilności.

- **Mini-batch**: W standardowym procesie uczenia maszynowego, dane treningowe są przetwarzane przez model jeden przykład na raz. W przypadku mini-batch, zbiór treningowy jest podzielony na mniejsze grupy, które są przetwarzane równocześnie przez model.

  Każda z tych mniejszych grup, zwanych mini-batchami, zawiera stałą liczbę przykładów treningowych. Proces uczenia polega na iteracyjnym przetwarzaniu tych mini-batchy, obliczaniu gradientu funkcji straty dla każdego mini-batcha, a następnie aktualizacji wag modelu.

1.2 Opis zaimplementowanych zmiennych i stałych:

- **weights**: Wektor wag, który przechowuje wartości wag dla każdego wejścia oraz wagę biasu.

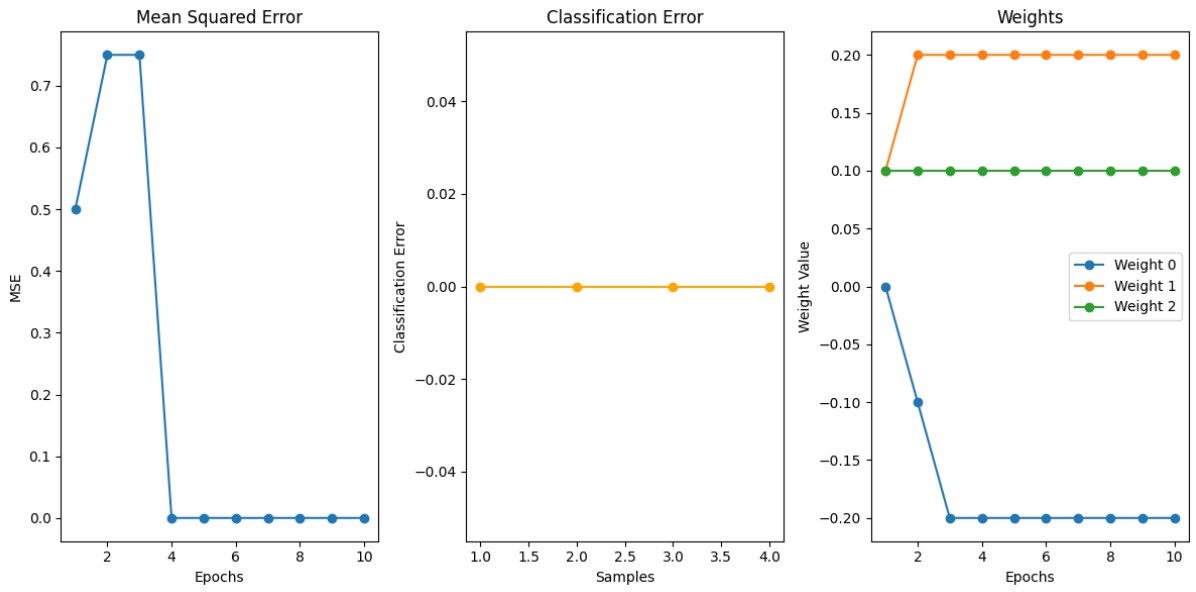
- **learning_rate**: Współczynnik uczenia, który określa jak szybko modele uczą się na podstawie błędu predykcji. Jest to parametr, który można dostosować, aby zoptymalizować proces uczenia.

- **epochs**: Liczba epok, czyli liczba pełnych przebiegów przez cały zestaw danych treningowych podczas uczenia modelu.

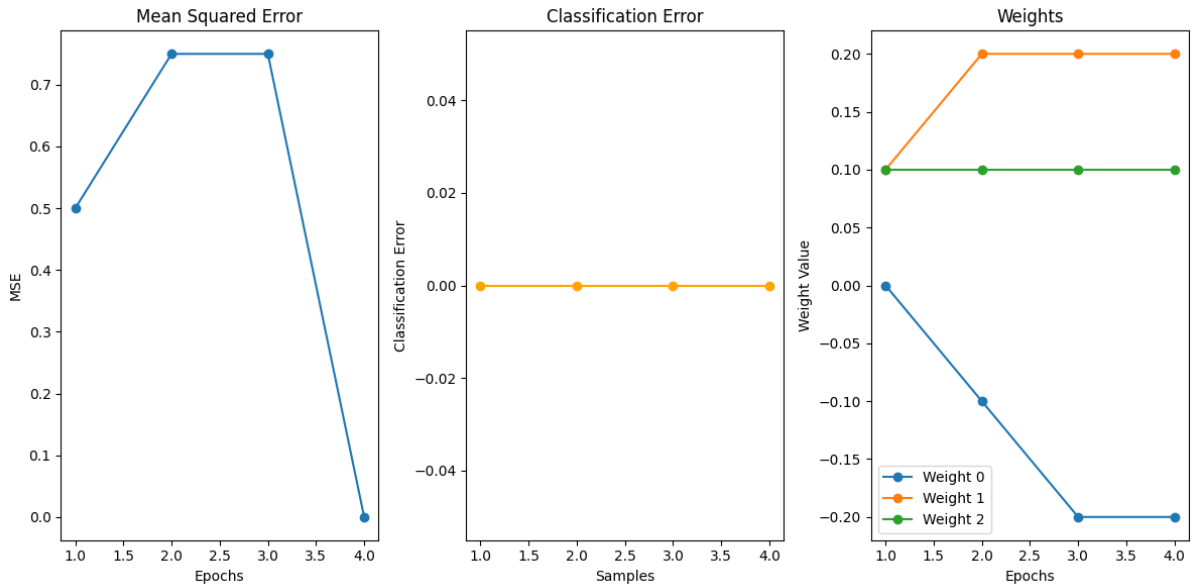
- **progMSE**: Wartość progowa błędu średniokwadratowego (MSE), która określa kryterium zakończenia procesu uczenia. Jeśli błąd MSE spadnie poniżej tej wartości, uczenie zostanie zakończone.

- **momentum_rate**: Współczynnik momentum, który kontroluje, jak bardzo aktualizacje wag z poprzednich iteracji wpływają na aktualne aktualizacje. Pomaga to w przyspieszeniu uczenia się i uniknięciu lokalnych minimów.
- **batch_size**: Rozmiar mini-batcha, czyli liczba próbek treningowych używanych do aktualizacji wag w każdym kroku gradientowym. Ustawienie tego parametru na wartość większą niż 1 wprowadza technikę mini-batch gradient descent, co może przyspieszyć proces uczenia się modelu.
- **loss_history**: Lista przechowująca historię wartości błędu (MSE) dla każdej epoki uczenia.
- **weights_history**: Lista przechowująca historię wartości wag dla każdej epoki uczenia.
- **prev_delta**: Wektor przechowujący poprzednie zmiany wag, które są wykorzystywane do implementacji momentum w procesie uczenia.
- **sum_squared_gradients**: Wektor przechowujący sumę kwadratów gradientów, wykorzystywany w adaptacyjnym współczynniku uczenia.
- **epsilon**: Stała mała wartość dodatkowa, dodawana do mianownika w adaptacyjnym współczynniku uczenia, aby uniknąć dzielenia przez zero.

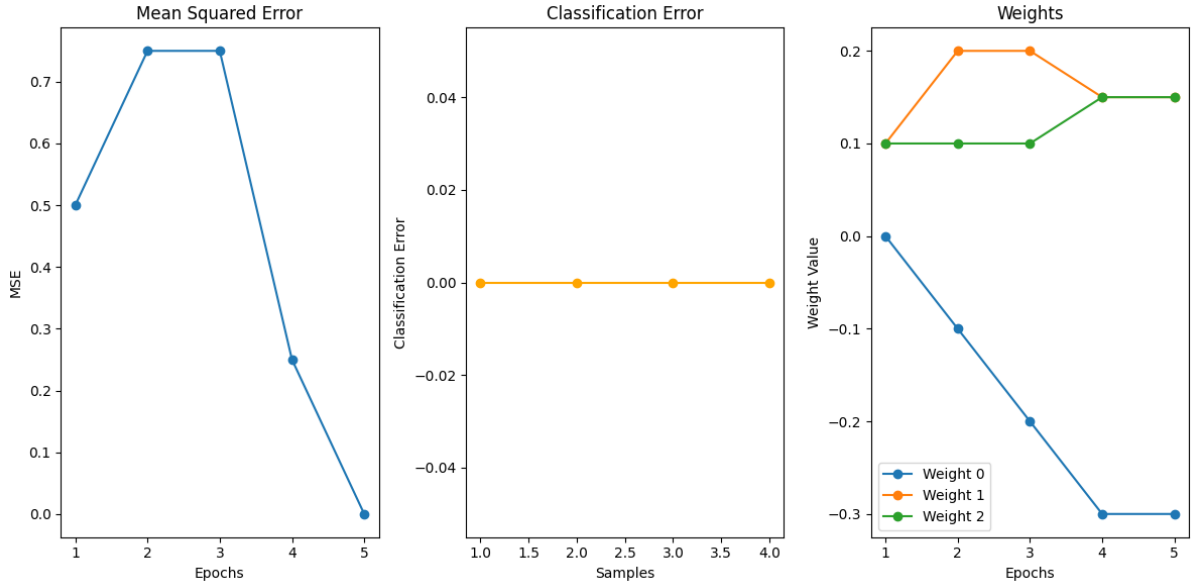
1.3 Wykresy błędu MSE, klasyfikacji i wag:



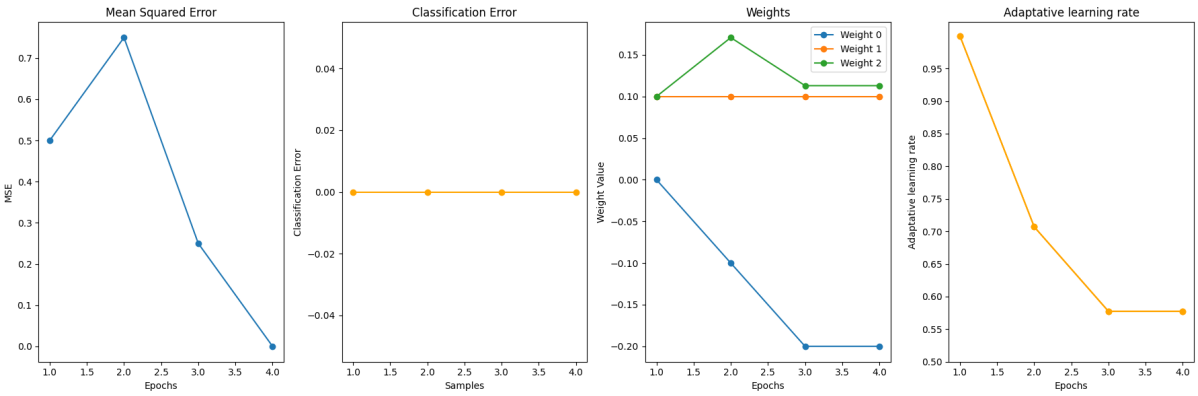
Rysunek 1: Wykresy błędów dla wersji podstawowej



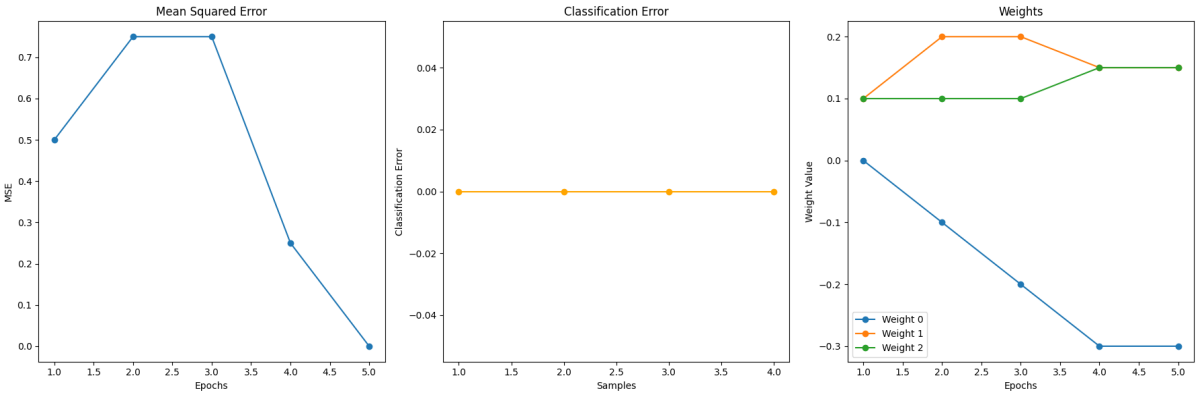
Rysunek 2: Wykresy błędów dla wersji szybszego uczenia



Rysunek 3: Wykresy błędów dla wersji z momentum (0.5)



Rysunek 4: Wykresy błędów dla wersji z adaptacyjnym współczynnikiem uczenia



Rysunek 5: Wykresy błędów dla wersji z mini-batch

1.4 Repozytorium git

Repozytorium projektu - https://github.com/KrystianCyga/Perceptron-Rosenblatta