Praca projektowa z przedmiotu

Sztuczna Inteligencja

Bartłomiej Czajka 169522 2 EF-DI P1

Rzeszów, 2023

Spis treści

1 Opis projektu

1.1 Założenia projektowe

Celem projektu jest realizacja sieci neuronowej uczonej za pomocą algorytmu sieci głębokiej, klasyfikującej chorobę Parkinsona oraz zbadanie wpływu parametrów sieci na proces uczenia. Projekt został zrealizowany w języku Python z wykorzystaniem biblioteki PyTorch.

1.2 Zestaw danych

Zestaw danych uczących został pobrany ze strony http://archive.ics.uci.edu/ml/datasets/Parkinsons. Zawiera on 197 instancji, 23 cechy oraz 2 klasy. Dane są nieuporządkowane, nie ma danych nieokreślonych. Dokładniejszy opis cech zestawu:

- name - Nazwa badanego pacjenta w ASCII i numer nagrania.
- MDVP:Fo(Hz) - Średnia częstotliwość podstawowa głosu.
- MDVP:Fhi(Hz) - Maksymalna częstotliwość podstawowa głosu.
- MDVP:Flo(Hz) - Minimalna częstotliwość podstawowa głosu.
- MDVP:Jitter(%), MDVP:Jitter(Abs), MDVP:RAP, MDVP:PPQ, Jitter:DDP - Kilka miar zmienności częstotliwości podstawowej.
- MDVP:Shimmer, MDVP:Shimmer(dB), Shimmer:APQ3, Shimmer:APQ5, MDVP:APQ, Shimmer:DDA - Kilka miar zmienności amplitudy.
- NHR, HNR - Dwie miary stosunku szumu do składowych tonalnych w głosie.
- status - Stan zdrowia badanego (jeden) - chory na Parkinsona, (zero) - zdrowy.
- RPDE, D2 - Dwie miary złożoności dynamicznej nieliniowej.
- DFA - Wykładnik skalowania fraktalnego sygnału.
- spread1, spread2, PPE - Trzy nieliniowe miary zmienności częstotliwości podstawowej.

1.3 Przygotowanie danych

Sieć ma za zadanie sklasyfikować czy pacjent cierpi na chorobę Parkinsona. Wartość '1' oznacza osobę cierpiącą na chorobę Parkinsona, natomiast '0' oznacza osobę zdrową. Dane zostały uporządkowane i znormalizowane. Normalizacja

została wykonana dla każdej obserwacji przy użyciu wzoru:

$$x_{\text{norm}} = \frac{x_{\text{norm\_max}} - x_{\text{norm\_min}}}{x_{\max} - x_{\min}} \cdot (x - x_{\min}) + x_{\text{norm\_min}}$$

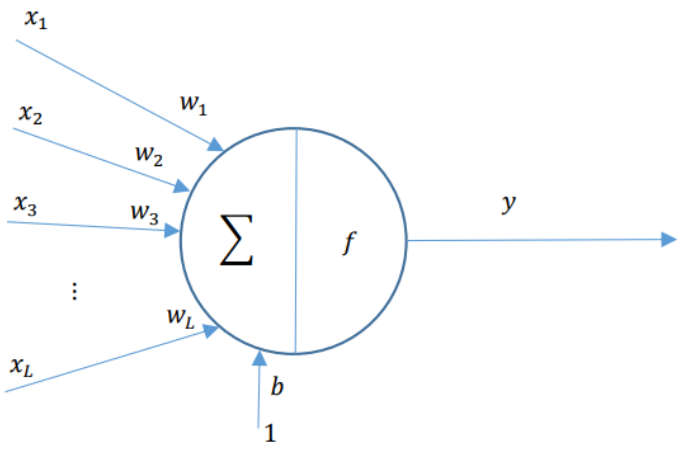
gdzie:

- x_{min} - to wektor zawierający minimalne wartości dla każdej obserwacji.
- x_{max} - to wektor zawierający maksymalne wartości dla każdej obserwacji.
- x_{norm_min} - to wartość minimalna po normalizacji (-1).
- x_{norm_max} - to wartość maksymalna po normalizacji (1).
- x_{norm} - to macierz zawierająca znormalizowane dane.
- x - to macierz zawierająca dane wejściowe.

2 Zagadnienia teoretyczne

2.1 Matematyczny opis sztucznego neuronu

Podstawowym elementem budującym strukturę sieci neuronowej jest neuron. Jest to element, który przetwarza informacje. W pewnym, uproszczonym stopniu jest wzorowany na funkcjonowaniu biologicznej komórki nerwowej. Struktura neuronu zawiera wiele wejść oraz jedno wyjście. Jednym z najważniejszych składników neuronu są wagi, których wartości decydują o zachowaniu neuronu. Są one zazwyczaj ustalane w trakcie procesu uczenia.



Rysunek 1: Model neuronu [4]

Każdy pojedynczy neuron przyjmuje sygnały wejściowe, które są następnie przetwarzane. Każde wejście ma przypisany współczynnik wagowy, który określa jak bardzo wpływa ono na wynik neuronu. Dodatkowo, neuron posiada "bias", czyli dodatkowe wejście, na którym występuje stała wartość. Wszystkie te informacje są sumowane, gdzie jest obliczane łączne pobudzenie neuronu. Następnie wartość pobudzenia przechodzi przez funkcję aktywacji, która określa sygnał wyjściowy neuronu dany wzorem:

$$y = \sum_{j=1}^{L} f(w_j x_j + b)$$

gdzie:

- j – indeks, który przyjmuje wartości od 1 do L,
- y – wyjście neuronu,
- w_j – współczynnik wagowy przypisany do j-tego wejścia,
- x_j – j-ty sygnał wejściowy,
- b – bias [2].

2.2 Funkcja aktywacji

Sam model matematyczny neuronu nie byłby wystarczający do skomplikowanych obliczeń w m.in. sieciach głębokich. Należy wprowadzić funkcje aktywacji, które nadają sieciom neuronowym zdolność modelowania nieliniowych relacji między danymi wejściowymi, a wyjściowymi. Dzięki tej nieliniowej transformacji na wyjściu sztucznego neuronu, możliwe jest wprowadzenie nieliniowości i bardziej skomlipowanych obliczeń. Istnieje wiele funkcji aktywacji.

Szczególnie przydatną w modelach głębokich ze względu na swoją prostotę i skuteczność jest funkcja ReLU, która została przeze mnie wykorzystana jako funkcja aktywacji dla warstw ukrytych. Działa na zasadzie przekazywania wartości dodatnich bez ich zmiany, natomiast dla wartości ujemnych przypisuje zerową wartość. Matematycznie można zdefiniować ją następującym wzorem:

$$f(x) = \max(0, x)$$

Innymi słowy, funkcja ReLU jest zdefiniowana na przedziale:

$$f(x) = \begin{cases} 0, & \text{gdy } x \leq 0, \\ x, & \text{gdy } x > 0. \end{cases}$$

gdzie:

- x – wartość wejściowa, na której zostanie zastosowana funkcja aktywacji.

Istnieje również funkcja aktywacji softmax, która dla danego wektora wyników $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ gdzie n oznacza liczbę klas, funkcja softmax przekształca każdy element z_i na wartość prawdopodobieństwa p_i. Funkcję softmax można zdefiniować w następujący sposób:

$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

gdzie:

- $f(z)_i$ – i-ty element wyniku funkcji softmax dla wektora wyników $\mathbf{z}$,
- e^{z_i} – funkcja wykładniczą podniesiona do potęgi z_i,

- z_i – i-ty element wektora wyników,
- $\sum_{j=1}^{n} e^{z_j}$ – suma funkcji wykładniczych dla wszystkich elementów wektora wyników,
- n – liczba klas.

Ostatnią funkcją aktywacji, którą wykorzystałem jest funkcja sigmoidalna, która tak jak softmax została użyta do aktywacji warstwy wyjściowej. Funkcja sigmoidalna przekształca wartość wejściową na wartość z zakresu (0, 1), co czyni ją przydatną w problemach klasyfikacji binarnej. Matematycznie funkcję sigmoidalną można zdefiniować jako:

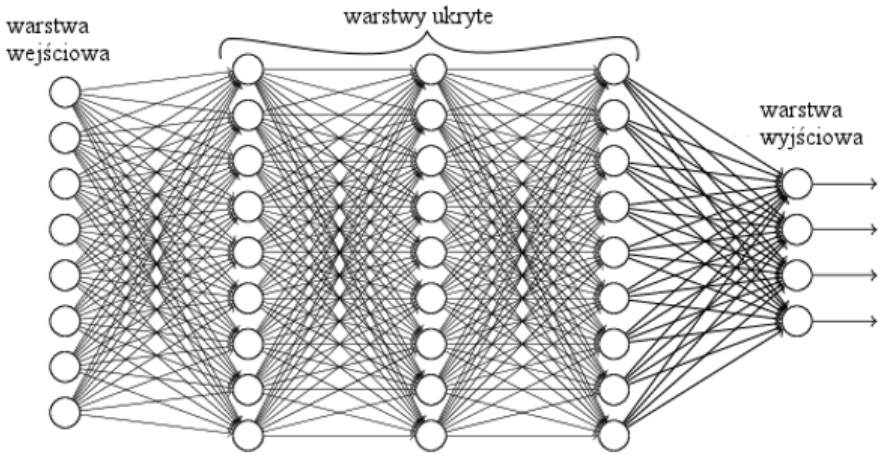
$$f(x) = \frac{1}{1 + e^{-x}}$$

gdzie:

- x – wartość wejściowa do funkcji sigmoidalnej.
- e – podstawa logarytmu naturalnego, przybliżone do wartości 2.71828.
- $f(x)$ – to wartość wyjściowa funkcji sigmoidalnej, przekształcająca x na wartość z przedziału (0, 1).

2.3 Sieć głęboka

Sieci głębokie to rodzaj modeli uczenia maszynowego, które składają się z wielu warstw neuronów, zwanych warstwami głębokimi. Pierwsza warstwa, zwana warstwą wejściową, przyjmuje dane wejściowe. Następnie występują warstwy ukryte, które przetwarzają dane i wyodrębniają cechy. Finalnie, wyniki przekazywane są do warstwy wyjściowej, która generuje ostateczne predykcje.



Rysunek 2: Sieć głęboka [5]

Proces uczenia takiej sieci polega na aktualizowaniu wagi neuronów w celu minimalizacji funkcji kosztu mierzącej błąd predykcji sieci. Zasada ich działania opiera się na propagacji informacji przez kolejne warstwy, nazywaną propagacją w przód (ang. forward propagation) oraz optymalizacji wag w procesie uczenia.

2.4 Funkcja kosztu

Wykorzystaną funkcją kosztu w kodzie jest entropia krzyżowa. Funkcja ta mierzy stopień niezgodności między rzeczywistymi etykietami a przewidywanymi prawdopodobieństwami dla każdej klasy. Im większa niezgodność, tym większa wartość funkcji kosztu. Dąży się do minimalizacji funkcji kosztu, poprzez zmianę wag sieci aby osiągnąć lepsze wyniki uczenia. Entropię krzyżową można zdefiniować następującym wzorem:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k}\log(p_{i,k}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i\log(p_i) + (1-y_i)\log(1-p_i)\right)$$

gdzie:

- N – liczba próbek w zbiorze treningowym,

- θ – parametry sieci, które chcemy uczyć,
- K – liczba klas,
- $y_{i,k}$ – wartość etykiety dla próbki i i klasy k (0 lub 1),
- $p_{i,k}$ – przewidywane prawdopodobieństwo dla próbki i i klasy k.

Aby obliczyć gradient funkcji kosztu, można użyć propagacji wstecznej błędu.

$$\frac{\partial J}{\partial z_i} = p_i - y_i$$

- $\frac{\partial J}{\partial z_i}$ to gradient funkcji kosztu względem wejścia z_i do funkcji aktywacji (np. softmax) w warstwie wyjściowej,
- p_i to przewidywane prawdopodobieństwo dla próbki i,
- y_i to prawdziwa etykieta (0 lub 1) dla próbki i.

Następnie ten gradient jest propagowany wstecz przez warstwy ukryte, aż do warstwy wejściowej, gdzie obliczane są gradienty względem wag sieci. W praktyce, algorytm propagacji wstecznej błędu jest stosowany do obliczenia gradientów dla wszystkich parametrów sieci, a następnie te gradienty są wykorzystywane w procesie optymalizacji, na przykład przy użyciu algorytmu ADAM, aby dostosować wagi sieci i minimalizować funkcję kosztu.

2.5 Algorytm wstecznej propagacji błędu

Algorytm wstecznej propagacji błędu, nazywany również algorytmem największego spadku gradientu, jest jednym z głównych algorytmów stosowanych w procesie uczenia sieci głębokich. Umożliwia on aktualizację wag sieci w celu minimalizacji funkcji kosztu poprzez iterację przez kolejne warstwy sieci. Proces ten można podzielić na dwa kroki: propagację w przód oraz wstecz.

Propagacja w przód polega na przekazywaniu danych wejściowych przez sieć od warstwy wejściowej do warstwy wyjściowej. W każdej warstwie obliczane są aktywacje neuronów na podstawie obecnych wag i danych wejściowych. Wyniki

z danej warstwy przekazywane są do kolejnej aż do warstwy wyjściowej gdzie generowany jest wynik predykcji.

W propagacji wstecznej natomiast porównuje się wynik sieci z oczekiwanym wynikiem, obliczając przy tym błąd. Błąd ten jest następnie propagowany wstecz od zaczynając od warstwy wyjściowej. Obliczany jest gradient funkcji kosztu względem wag sieci, który informuje o kierunku dostosowania wag w celu minimalizacji błędu. Na podstawie gradientu aktualizowane są wagi sieci przy użyciu odpowiedniej metody optymalizacji, takiej jak metoda spadku gradientu czy też jej wariant, np. ADAM.

2.6 Metoda optymalizacji ADAM

Metoda ADAM jest popularna w optymalizacji sieci głębokich, ponieważ łączy zalety adaptacyjnego skalowania kroku uczenia (RMSProp) i momentu, co prowadzi do efektywniejszej optymalizacji i szybszego zbiegania do optymalnych wag sieci.

Wzór na gradient funkcji kosztu w metodzie ADAM można zapisać następująco:

$$g_t = \nabla_\theta J(\theta)$$

gdzie:

- g_t to gradient w kroku czasowym t,
- ∇_θ oznacza gradient po wagach θ,
- $J(\theta)$ to funkcja kosztu.

Gradient jest obliczany za pomocą standardowych metod propagacji wstecznej błędu w sieciach neuronowych.

1. Obliczenie gradientów funkcji kosztu względem wag sieci za pomocą propagacji wstecznej błędu.

2. Aktualizacja momentu gradientu m_t i drugiego momentu gradientu v_t na podstawie obliczonych gradientów.

3. Obliczenie wygładzonych estymacji momentu i drugiego momentu: $\hat{m}_t$ i $\hat{v}_t$.

4. Aktualizacja wag sieci z wykorzystaniem estymacji momentu i drugiego momentu, współczynnika uczenia α oraz małej wartości epsilon ϵ.

W metodzie ADAM, gradient jest używany do aktualizacji wag zgodnie z następującymi wzorami:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

gdzie:

- m_t to estymacja momentu gradientu w kroku czasowym t.
- v_t to estymacja drugiego momentu gradientu w kroku czasowym t.
- β_1 i β_2 to współczynniki z zakresu $[0, 1)$ kontrolujące eksponencjalne wygładzanie momentu i drugiego momentu.
- α to współczynnik uczenia (learning rate).
- ϵ to mała wartość (np. 10^{-8}) używana w celu uniknięcia dzielenia przez zero.
- $\hat{m}_t$ i $\hat{v}_t$ to wygładzone estymacje momentu i drugiego momentu.
- θ_t i θ_{t+1} to wagi w krokach czasowych t i $t + 1$ odpowiednio.

Przy użyciu tych równań, algorytm ADAM aktualizuje wagi sieci neuronowej, wykorzystując estymację momentu i drugiego momentu gradientu. To pozwala na efektywną optymalizację sieci głębokiej, dostosowując krok uczenia (learning rate) dla każdej wagi indywidualnie.

3 Realizacja sieci neuronowej

3.1 Opis skryptu

4 Eksperymenty

4.1 Eksperyment 1

4.2 Eksperyment 2

4.3 Eksperyment 3

5 Wnioski

Literatura

[1] Michael Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[2] Ryszard Tadeusiewicz, Maciej Szaleniec, *Leksykon sieci neuronowych*. Wrocław, 2015.

[3] Nicolas Vandeput, *A Brief History Of Neural Networks*, [dostęp: 17.05.2023].

[4] Zajdel R., *Ćwiczenie 4 Model Neuronu*, Rzeszów, KIiA, PRz.

[5] Paweł Gora, *Głębokie uczenie maszyn*, [dostęp: 17.05.2023].

[6] Zajdel R., *Procedura przygotowania danych dla sieci neuronowych na potrzeby projektu z modułu sztuczna inteligencja - Listing1*, Rzeszów, KIiA, PRz.

[7] Zajdel R., *Procedura przygotowania danych dla sieci neuronowych na potrzeby projektu z modułu sztuczna inteligencja - Listing2*, Rzeszów, KIiA, PRz.