Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Sterowania i Elektroniki Przemysłowej

Praca dyplomowa inżynierska

na kierunku Informatyka Stosowana

w specjalności

Porównanie wydajności rozwiązań do realizacji sieci neuronowych w językach: Matlab, Python, C++.

Piotr Heinzelman
numer albumu 146703

promotor
dr inz. Witold Czajewski

WARSZAWA 2024

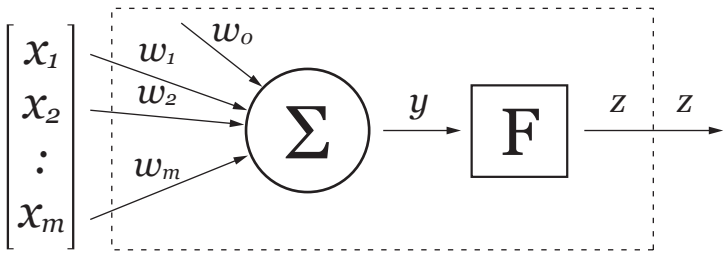
Spis treści

Rozdział 1

Wstęp

1.1 Model klasyczny

Model sztucznego neuronu [1], rys. 1 można rozpatrywać jako specyficzny przetwornik sygnałów, gdzie na wejście podawana jest pewna liczba m sygnałów wejściowych x_1 ... x_m, natomiast na wyjściu pojawia się tylko jeden sygnał wyjściowy z. W modelu klasycznym przetwarzane sygnały nie są sygnałami algebry Boole'a prawda/fałsz w realizacji 0 i 1. Można powiedzieć, że neurony pracują na sygnałach logiki rozmytej (fuzzy logic), ponieważ sygnały przyjmują wartości z określonych zakresów. Odpowiedz sieci jest również wektorem wartości z zakresów, a w szczególności odpowiedz warstwy typu „softmax" – jest wprost wektorem rozkładów prawdopodobieństwa przynależności do klas.



Rysunek 1. Model sztucznego neuronu [1]

W pierwszym kroku przetwarzania dla każdego z sygnałów wejściowych wyznaczana jest jego "ważona" wielkość, czyli iloczyn wielkości sygnału z wartością "wagi" dla tego sygnału. x_i*w_i

W drugim kroku ważone sygnały są sumowane a wynik odpowiadają potencjałowi membranowemu komórki y. $y = \sum_{i=1}^{m} x_i * w_i$ W kroku trzecim wyliczana jest wartość funkcji aktywacji F(y). W większości modeli występuje także energia aktywacji θ, o którą zmniejszona jest wartość wyjściowa zgodnie ze wzorem:

$$y = \sum_{i=1}^{m} x_i * w_i - \theta, z = F(y); \tag{1}$$

gdzie: x_i - i-ty sygnał wejściowy, w_i - i-ta waga w neuronie, θ - energia aktywacji, F - funkcja aktywacji, z - wielkość sygnały wyjściowego neuronu.

Przy podstawieniu: x_0=1 oraz w_0=-θ wzór przyjmuje wygodniejszą postać:

$$y = \sum_{i=0}^{m} x_i * w_i, \tag{2}$$

Neurony zorganizowane są w warstwy w taki sposób, by te należące do jednej warstwy, miały dostęp do tego samego wektora wejściowego X. Sygnał wyjściowy każdego neuronu to pojedyncza wartość z_j, zaś na sygnał wyjściowy warstwy składają się sygnałów wszystkich neuronów należących do tej warstwy i tworzą wektor Z. Ilość neuronów w warstwie oznaczamy n, zatem liczność wektora Z to także n.

$$Z = \begin{bmatrix} z_1, \\ z_2, \\ : \\ z_n \end{bmatrix} \tag{3}$$

W najprostszych układach neurony z jednej warstwy nie komunikują się między sobą.

W zapisie wektorowym mamy:

$$X = \begin{bmatrix} x_0 = 1, \\ x_1 ... \\ : \\ x_m ... \end{bmatrix}, W = \begin{bmatrix} W_1 & W_2 & W_3 \\ \begin{bmatrix} w_0 \\ w_1 \\ w_m \end{bmatrix} & \begin{bmatrix} w_0 \\ w_1 \\ w_m \end{bmatrix} & \begin{bmatrix} w_0 \\ w_1 \\ w_m \end{bmatrix} \end{bmatrix}, \tag{4}$$

$$z_1 = F(W_1 . * X), Z = F(W^T * X); \tag{5}$$

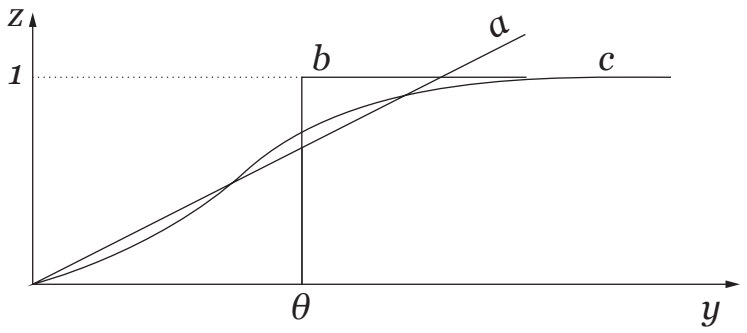
Rozpiszemy mnożenie macierzy przez macierz $W^T * X$:

$$\begin{bmatrix} w_{10} \ , \ w_{11} \ , \ w_{1m} \\ w_{20} \ , \ w_{21} \ , \ w_{2m} \\ w_{n0} \ , \ w_{n1} \ , \ w_{nm} \end{bmatrix} \quad \begin{bmatrix} x_0 \\ x_1 \\ x_m \end{bmatrix} \quad \begin{bmatrix} x_0 * w_{10} + x_1 * w_{11} + x_m * w_{1m} \\ x_0 * w_{20} + x_1 * w_{21} + x_m * w_{2m} \\ x_0 * w_{n0} + x_1 * w_{n1} + x_m * w_{nm} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_m \end{bmatrix} \tag{6}$$

w którym X jest wektorem wejściowym, x_i jest i-tym sygnałem wejściowym, w_i jest i-tą wagą, w_{ji} jest i-tą wagą j-tego neuronu, a. W_j oznacza zapis wag jako wektor, W oznacza macierz wag.

Symbol .* oznacza mnożenie Hadamarda czyli mnożenie pierwszego elementu z pierwszym, drugiego z drugim itd. Zapis X^T oznacza transpozycję macierzy lub wektora. Zapis $W^T * X$ oznacza mnożenie macierzowe X^T przez W. Aby wyliczyć odpowiedz jednej warstwy musimy wykonać $m * n$ mnożeń i $(m-1) * n$ dodawań oraz musimy wyliczyć n razy wartości funkcji $F(y_j)$.

1.1.1 Funkcje aktywacji



Rysunek 2. Funkcje aktywacji

Sygnał y przetwarzany przez blok aktywacji F może być opisany różnymi funkcjami.

- Może być to np. prosta funkcja liniowa (a):

$$z = ky, \text{gdzie } k \text{ jest zadanym stałym współczynnikiem.} \tag{7}$$

- ReLU - funkcja liniowa, która w części dodatniej ma współczynnik k=1, oraz współczynnik k=0 w części ujemnej:

$$z = F(y) = y^+ = max(0, y) = z\begin{cases} x \text{ jeśli } y > 0, \\ 0 \text{ jeśli } y \leq 0, \end{cases} \tag{8}$$

- Funkcja skokowa Heaviside'a, skok jednostkowy (b):

$$y = H(z) = \mathbb{1}(z) = \begin{cases} 1 \text{ jeśli } y > \theta, \\ 0 \text{ jeśli } y \leq \theta, \end{cases} \tag{9}$$

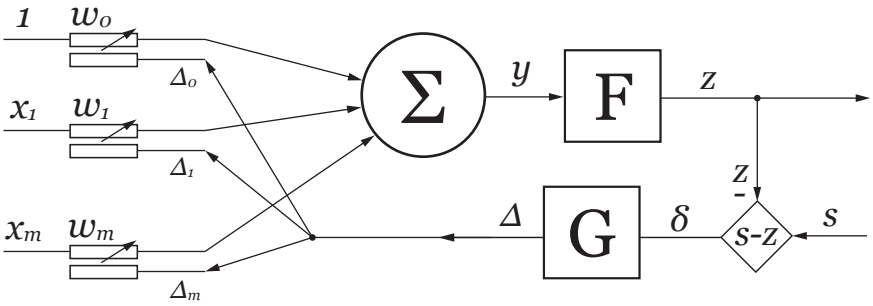
- Funkcja bipolarna:

$$y = sgn(z) = \begin{cases} 1 \text{ jeśli } y > \theta, \\ -1 \text{ jeśli } y \leq \theta, \end{cases} \tag{10}$$

- Funkcja logistyczna (sigmodalna) (c):

$$z = \frac{1}{1 + exp(-\beta y)}, \text{ dla } \beta\text{=1 pochodna: } \frac{\partial z}{\partial y} = z(1 - z) \tag{11}$$

1.1.2 Proces uczenia



Rysunek 3. Proces uczenia elementu perceptronowego

Proces uczenia pojedynczego neuronu polega na:

- obliczeniu wartości sygnału wyjściowego y.
- obliczeniu wartości sygnału wyjściowego $z = F(y)$.
- porównaniu wektora wyjściowego z z wartością oczekiwaną s oraz obliczenie różnicy $\delta = s - z$.
- zależnie od metody uczenia wyznaczenia wartości $\Delta = G(\delta)$;
- wyznaczenie wielkości poprawek dla poszczególnych wag Δ_i.
- aktualizacja wagi: $w(k+1) = w(k) + \Delta w(k)$.

$$w(k+1) = w(k) + G(s - z), \tag{12}$$

Gdzie $w(k+1)$ jest wartością uaktualnioną, $w(k)$ wartością przed aktualizacją. Funkcja $G(\delta)$ jest funkcją, na podstawie której obliczamy wielkości poprawek.

1.1.3 Algorytm propagacji wstecznej

Algorytm ten [1], podaje on przepis na zmianę wag w_{ij} dowolnych połaczeń elementów przetwarzających rozmieszczonych w sąsiednich warstwach sieci jednokierunkowej. Jest on oparty na minimalizacji sumy kwadratów błędów uczenia z wykorzystaniem optymalizacyjnej metody największego spadku (Wit, 1986). Dzięki zastosowaniu specyficznego sposobu propagowania błędów uczenia sieci powstałych na wyjściu, tzn. przesyłania ich do warstwy wyjściowej od wejściowej, algorytm propagacji wstecznej stał się jednym z najskuteczniejszych algorytmów uczenia sieci. Rozważamy sieć jednowarstwową o liniowych elementach przetwarzających. Załóżmy, że mamy P-elementowy zbiór wzorców. Przy prezentacji μ-tego wzorca możemy zdefiniować błąd:

$$\delta_j^\mu = s_j^\mu - z_j^\mu = s^\mu - y_j^\mu = s^\mu - \sum_{i=0}^{m} w_{ij} x_i^\mu, \tag{13}$$

gdzie s_j^μ, y_j^μ oznaczają odpowiednio oczekiwane i aktualne wartości wyjścia j-tego elementu oraz ważoną sumę wejść wyznaczoną w jego sumatorze przy prezentacji μ-tego wzorca. x_i^μ i-ta składowa μ-tego wektora wejściowego, w_{ji} - oznacza wagę połączenia pomiędzy j-tym elementem warstwy wyjściowej a i-tym elementem warstwy wejściowej. m-liczba wejść.
Jako miarę błędu sieci ξ wprowadzimy sumę po wszystkich wzorcach błędów powstałych przy prezentacji każdego z nich:

$$\xi = \sum_{\mu=0}^{P} \xi_\mu = \frac{1}{2} \sum_{\mu=1}^{P} \sum_{j=1}^{n} (s^\mu - y^\mu)^2, \tag{14}$$

gdzie

$$\xi_\mu = \frac{1}{2} \sum_{j=1}^{n} (s^\mu - y^\mu)^2, \tag{15}$$

Problem uczenia sieci to zagadnienie minimalizacji funkcji błędu ξ. Jedną z najprostszych metod minimalizacji jest gradientowa metoda największego spadku [3]. Jest to metoda iteracyjna, która poszukuje kolejnego lepszego punktu w kierunku przeciwnym do gradientu funkcji celu w danym punkcie. Stosując powyższą metodę do uczenia sieci, zmiana Δw_{ji} wagi połączenia winna spełniać relację:

$$\Delta w_{ji} = -\eta \frac{\partial \xi}{\partial w_{ji}} = -\eta \sum_{\mu=1}^{P} \frac{\partial \xi_\mu}{\partial w_{ji}} = -\eta \sum_{\mu=1}^{P} \frac{\partial \xi_\mu}{\partial z_j^\mu} \frac{\partial z_j^\mu}{\partial w_{ji}} \tag{16}$$

gdzie η oznacza współczynnik proporcjonalności. W przypadku elementów liniowych mamy:

$$\frac{\partial \xi_\mu}{\partial z_j^\mu} = -(s_j^\mu - z_j^\mu) = -\delta_j^\mu, \tag{17}$$

$$\frac{\partial z_j^\mu}{\partial w_{ji}} = \frac{\partial y_j^\mu}{\partial w_{ji}} = x_i^\mu \tag{18}$$

stąd otrzymujemy:

$$\Delta w_{ji} = \eta \sum_{\mu=1}^{P} \delta_j^\mu x_j^\mu \tag{19}$$

ostatecznie pełną regułę zapiszemy:

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}, \tag{20}$$

Konsekwentna realizacja metody największego spadku wymaga dokonywania zmian wag dopiero po zaprezentowaniu sieci pełnego zbioru wzorców. W praktyce stosuje się jednak zmiany wag po każdej prezentacji wzorca zgodnie ze wzorem:

$$\Delta^\mu w_{ji} = -\eta \frac{\partial \xi_\mu}{\partial w_{ji}} = \eta \delta_j^\mu x_i^\mu, \tag{21}$$

1.1.4 Uogólniona reguła delty

Rozważmy sieć jednowarstwową z elementami przetwarzającymi o nieliniowej, lecz niemalejącej i różniczkowalnej funkcji aktywacji F wówczas zmianę wag przy prezentacji μ-tego wzorca można opisać równaniem:

$$\Delta w_{ji} = -\eta \frac{\partial \xi}{\partial w_{ji}} = -\eta \frac{\partial \xi}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}} = -\eta \frac{\partial \xi}{\partial z_j} \frac{\partial z_j}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}}, \tag{22}$$

przy czym:

$$\frac{\partial \xi}{\partial z_j} = (s - z_j), \text{ z def. } ((\frac{1}{2}(x-y)^2)' = (x-y), \tag{23}$$

$$\frac{\partial z_j}{\partial y_j} = F'(y_j), \tag{24}$$

$$\frac{\partial y_j}{\partial w_{ji}} = x_i; \tag{25}$$

stąd ostatecznie wzór przyjmuje postać:

$$\Delta w_{ji} = \eta F'(y)(s - z_j)x_i \tag{26}$$

oraz dla warstwy ukrytej:

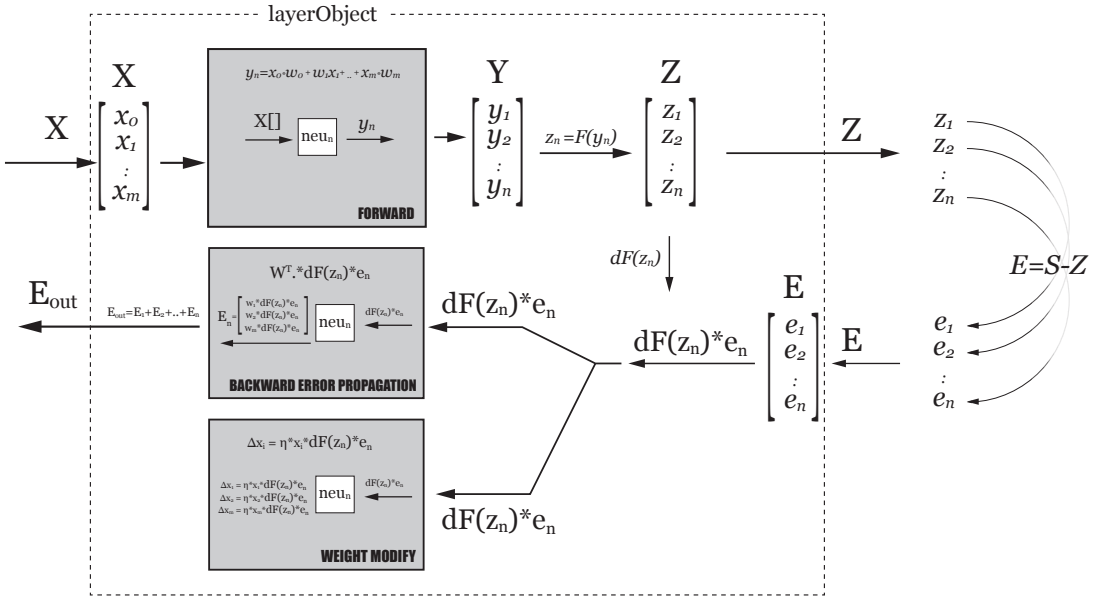
$$\Delta w_{ji} = \eta F'(y_j) \sum_{i=1}^{n_{m+1}} F'(y_j)(s^{m+1} - z_j^{m+1})w_{ji} \tag{27}$$

Rozdział 2

Model obiektowy

Rozbicie matematycznego modelu na dwie prostsze konstrukcje może ułatwić zrozumienie modelu oraz śledzenie zachodzących w nim procesów i obliczeń. Operacje mnożenia macierzy zastępujemy operacjami mnożenia wartości w pętli, a takie operacje są bardziej naturalne dla dziedziny metod numerycznych niż formalne zapisy matematyczne. Realizacja w modelu obiektowego powoduje także że cały proces jest bardzo elastyczny i umożliwia łatwe wprowadzanie dodatkowych obiektów np. obserwatorów.

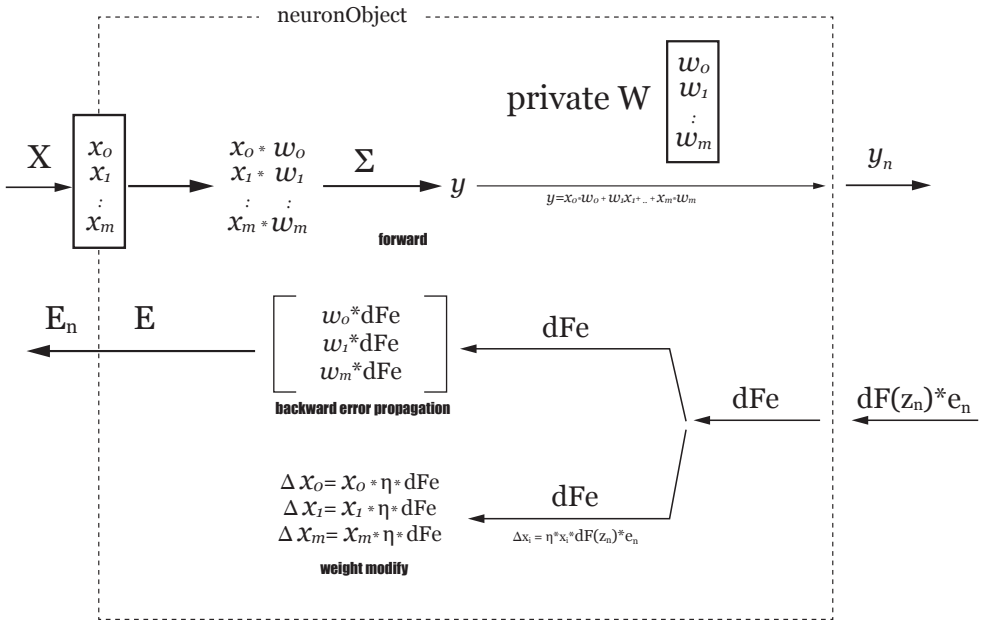
2.1 Obiekt klasy Layer



Rysunek 4. Przepływu sygnałów w obiekcie Layer

Każdy obiekt Layer ma własną kolekcję obiektów klasy Neuron. Dysponuje też polami danych np. X, które są zbiorami wartości. I tak pole danych X jest zbiorem wartości $x_0, x_1, ...x_m$. pole danych Y jest zbiorem wartości $y_0, y_1, ...y_m$. Z matematycznego punktu widzenia, można powiedzieć że wektor X jest wektorem zmiennych niezależnych X = [$x_0, x_1, ...x_m$], skoro tak, to i **pochodne składników tych wektorów będą niezależne względem siebie**. Z informatycznego punktu widzenia pole X jest zbiorem wartości $x_0, x_1, ...x_m$, które może być realizowane przez takie struktury danych jak *zbiór*, *lista*, *tablica* czy *wektor*. Zależnie od języka programowania pewne struktury będą wygodniejsze do wykorzystania od innych. Struktura *tablica* jest najprostsza, kolejne wartości są indeksowane liczbą całkowitą, a sama tablica po utworzeniu nie może zmieniać swojego rozmiaru.

Obiekty klasy Layer wywołuje dla wszystkich neuronów ze swojej kolekcji żądania wykonania obliczeń. Obiekty klasy Neuron mają dostęp do obiektu rodzica, a dzięki temu mają także dostęp do niektórych pól danych obiektu Layer. Neurony nie mają jednak bezpośrednich połączeń między sobą.



Rysunek 5. Przepływu sygnałów w obiekcie Neuron

Obiekty klasy Neuron są bardzo proste i lekkie, realizują kilka operacji matematycznych wywoływanych na żądanie warstwy rodzica. Do obliczeń używają wewnętrznych zmiennych wag zorganizowanych w tablicę $W = [w_0, w_1, ...w_m]$, oraz dostarczonych przez rodzica zmiennych skalarnych oznaczanych małymi lierami np. η, lub tablic skalarów oznaczanych dla rozróżnienia wielkimi literami np. pole X.

2.2 Propagacja sygnału

Pojedynczy neuron odczytuje wartości wektora wejściowego - tablicy X o określonym rozmiarze m, przekazanego przez rodzica. Oblicza iloczyn odpowiednich wag i wartości, a następnie sumuje uzyskane iloczyny. Obliczoną **wartość skalarną zmiennoprzecinkową** - zwraca jako wynik operacji.

```
1  public float forward( float[] X ) {
2      float sum=0;
3      for ( int m=0; m<W.length; m++ ) {
4          yi= X[m]*W[m];
5          sum = sum + yi;
6      }
7      return sum;
8  }
```

obiekt Layer dla otrzymanych wartości $y_1, y_2, ...y_n$ oblicza wielkość funkcji aktywacji $z_i = f(y_i)$. Wielkości te zebrane w tablicę tworzą wartość wyjściową Z. Przy okazji obliczamy także wartość pochodnych funkcji F w punktach y_i i zapisujemy w tablicy dFofZ).

```
1  public void forward(){
2      for ( int n=0; n<neurons.length; n++ ) {
3          Y[n] = neurons[n].forward( X );
4          Z[n] = F ( Y[n] );
5          dFofZ[n] = dF( Z[n] );
6      }
7  }
```

2.3 Wartość funkcji aktywacji $F(y)$

Metoda licząca wielkość funkcji aktywacji zależnie do rodzaju warstwy:

```
1  private float F ( float y ){
2      float z;
3      switch (this.lType) {
4          case sigmod: { return ( 1/(1 + Math.exp( -y ))); }
5          case linear:
6              default: { z=y; break; }
7      }
8      return z;
9  }
```

2.4 Pochodna funkcji aktywacji $dF(y)$

W uczeniu metodą propagacji wstecznej błędów potrzebujemy wielkości pochodnej funkcji aktywacji w punkcie y. Taka metoda uczenia jest najskuteczniejsza. Wielkość zwracana przez tę metodę jest

współczynnikiem proporcjonalności wykorzystywanym do obliczania zmian wartości współczynników wag.
Uwaga: Dla niektórych funkcji, lub dla pewnych wartości funkcji, może nie istnieć wartość pochodnej. Niektóre starsze metody uczenia nie wykorzystywały wartości pochodnej a inne wielkości zależne tylko od czasu, lub od innych parametrów.

```
1    private float dF (float z ){
2        float df;
3        switch (lType) {
4            case sigmod: { df = z*(1-z); break; }
5            case linear:
6            default: { df=1; break; }
7        }
8        return df;
9    }
```

Pierwsze programy napisane przy użyciu modelu wykonują zadania z [6], zadania z -
!! Sprawdzić i uzupełnić kod !! poprawić tekst (zmienne ciągłe) dodać L i dL do rysunków.

Rozdział 5

Jednowymiarowa regresja liniowa

[5] Funkcja liniowa jednej zmiennej to funkcja w postaci $y = w_1x + w_0$; współczynnki w_0 i w_1 możemy traktować jak wagi, i możemy je traktować łącznie jako wektor $\mathbf{W} =< w_0, w_1 >$ a samo przekształcenie można utożsamić z iloczynem skalarnym $y = \mathbf{W} * < 1, x >$. Zadanie dopasowania najlepszej hipotezy hw wiążącej te dwie wielkości nosi nazwę regresji liniowej. Matematycznie dopasowanie to sprowadza się do znalezienia wektora W minimalizującego funkcję straty, zgodnie z teorią Gaussa jako miarę tej straty przyjmuje się sumę miar dla wszystkich przykładów:

$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, hw(x_j)) = \sum_{j=1}^{N} L_2(y_j - hw(x_j))^2 = \sum_{j=1}^{N} L_2(y_j - (w_1x + w_0))^2, \quad (28)$$

Naszym celem jest znalezienie optymalnego wektora W

$$\mathbf{W} = \text{argmin } Loss(h_w) \quad (29)$$

Gdy funkcja ciągła osiąga minimum w danym punkcie, pierwsze pochodne cząstkowe po argumentach tej funkcji zerują się w tym punkcie; w kontekście regresji liniowej nasza funkcja $Loss(h_w)$ jest funkcją dwu zmiennych: w_0 i w_1, których wartości w punkcie minimum określone są przez układ równań:

$$\begin{cases} \frac{\partial}{\partial w_0} \sum_{(} y_j - (w_1x + w_0))^2 = 0, \\ \frac{\partial}{\partial w_1} \sum (y_j - (w_1x + w_0))^2 = 0, \end{cases} \quad (30)$$

Rozwiązaniem takiego układu są wartości:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(x_j^2) - (\sum x_j)^2}, w_0 = \frac{\sum y_j - w_1(\sum x_j)}{N}, \quad (31)$$

Dla dużych N[5] musimy użyć następującej, równoważnej postaci rzeczonych wzorów:

$$w_1 = \frac{\sum(x_j - \overline{x})(y_j - \overline{y})}{\sum(x_j - \overline{x})^2}, w_0 = \overline{y} - w_1\overline{x}, \quad (32)$$

gdzie $\overline{x}$ i $\overline{y}$ są średnimi arytmetycznymi:

$$\overline{x} = \frac{\sum x_j}{N}, \overline{y} = \frac{\sum y_j}{N}, \tag{33}$$

5.0.1 Realizacja obliczeń

Przykłady obliczeń Python i Matlab zostały zaczerpnięte z [ossowski2023]. pełen kod dostępny pod adresem: https://github.com/piotrHeinzelman/inz/tree/main w przypadku Matlab i Python korzystam z dostępnych funkcji, w przypadku Java obliczam wg. wzoru ! 23 !. Obliczenia różnymi metodami dają zbliżone wyniki, więc zakładam że moje implementacje są poprawne.

Matlab:

```
    x=[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ]
    y=[ -1.69 -0.79 5.77 7.80 4.56 14.32 15.47 8.88 7.41 17.26 14.83
   20.47 20.39 27.04 22.53 22.36 29.35 22.86 31.22 28.13 ]

    for i = 1:n
        a = polyfit(x,y,1);
    end
```

Python:

```
for i in range( cycles ):
    a = np.polyfit(x,y,1)
```

Java:

```
for ( int C=0; C<cycles; C++ ) {
        double xsr = 0.0;
        double ysr = 0.0;
        for (int i = 0; i < x.length; i++) {
            xsr += x[i];
            ysr += y[i];
        }
        xsr = xsr / x.length;
        ysr = ysr / y.length;

        w1 = 0.0;
        w0 = 0.0;
        double sumTop = 0.0;
        double sumBottom = 0.0;
        for (int i = 0; i < x.length; i++) {
            sumTop += ((x[i] - xsr) * (y[i] - ysr));
            sumBottom += ((x[i] - xsr) * (x[i] - xsr));
        }
        w1 = sumTop / sumBottom;
        w0 = ysr - w1 * xsr;
```

czasy wykonania kodu polyfit 20 próbek:

Java: 0.042 sek. !!!

Matlab: 5.936 sek.

Python: 26.918 sek.

Bibliografia

[1] Józef Korbicz Andrzej Obuchowicz, D. U., *Sztuczne sieci neuronowe: podstawy i zastosowania*. Akademicka Oficyna wydawnicza PLJ, 1994, ISBN: 83-7101-197-0.

[2] Makowski, Ł., *Szablon prac dyplomowych dla Wydziału Elektrycznego Politechniki Warszawskiej*, <https://github.com/SP5LMA/EE-Dyplom>.

[3] R., W., *Metody programowania nieliniowego*. Warszawa: WNT, 1986.

[4] Stanisław Osowski, R. S., *Matematyczne modele uczenia maszynowego w językach MATLAB i PYTHON*. Oficyna Wydawnicza Politechniki Warszawskiej, 2023, ISBN: 978-83-8156-597-4.

[5] Stuart Russell, P. N., *Artificial Intelligence: A Modern Aproach, 4th Edition*, Grażyński, T. A., red. Pearson Education, Inc., Polish language by Helion S.A. 2023, 2023, ISBN: 978-83-283-7773-8.

[6] Włodzimierz Kasprzak, prof. dr hab. inż., „Metody sztucznej inteligencji (2024L), MSI-C6.pdf," materiał dydaktyczny.

Spis rysunków