Spis treści

1 Model McCullocha-Pittsa **1**
1.1 Przykład . . . 2
1.2 Reprezentacja wektorowa . . . 2

2 Liniowa separowalność **2**
2.1 Przykład . . . 2

3 Associatron **2**
3.1 Liniowy model pamięci . . . 2
3.2 Cel . . . 3
3.3 Macierz wag . . . 3
3.4 Liniowa funkcja asocjacyjna . . . 3
3.5 Nieliniowa funkcja asocjacyjna . . . 3

4 Gradient Descent **4**
4.1 Algorytm . . . 4
4.2 Problemy . . . 4

5 Backpropagation **4**

6 CNN **5**
6.1 Konwolucja . . . 5
6.2 Pooling . . . 5

7 MB **6**

8 Transformer **6**

1 Model McCullocha-Pittsa

Jest to model matematyczny mający naśladować działanie fizjologicznych neuronów. Składa się on z n wejść u_i o wagach w_i i jednego wyjścia y. Neuron aktywuje się, gdy suma iloczynów wejść i wag jest większa od pewnej wartości progowej θ.

$$n_i, y \in \{0.0, 1.0\} \subset \mathbb{R}$$

$$w_i, \theta \in \mathbb{R}$$

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

$$y(\vec{u}, \vec{w}) = f((\sum_{i=1}^{n} w_i u_i) - \theta)$$

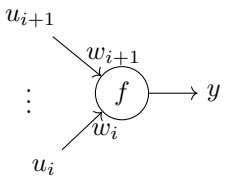

Diagram 1: Wizualizacja modelu McCullocha-Pittsa

u_1	u_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 1: Tabela prawdy dla funkcji logicznej AND

1.1 Przykład

- $u_1 = u_2 = 0 \to y = 0 = f(-\theta) \leftrightarrow \theta \geq 0$
- $u_1 = 0, u_2 = 1 \to y = 0 = f(w_2 - \theta) \leftrightarrow w_2 < \theta$
- $u_1 = 1, u_2 = 0 \to y = 0 = f(w_1 - \theta) \leftrightarrow w_1 < \theta$
- $u_1 = u_2 = 1 \to y = 1 = f(w_1 + w_2 - \theta) \leftrightarrow w_1 + w_2 \geq \theta$

$$\theta = 3, w_1 = 2, w_2 = 2$$

1.2 Reprezentacja wektorowa

$$\vec{u} = (u_1, u_2, \ldots, u_n)$$
$$\vec{w} = (w_1, w_2, \ldots, w_n)$$
$$y(\vec{u}, \vec{w}) = f(\vec{w} \cdot \vec{u} - \theta)$$

2 Liniowa separowalność

$$U_- = \{\vec{u_1}, \ldots, \vec{u_n}\} \subset \mathbb{R}^n$$
$$U_+ = \{\vec{u_{n+1}}, \ldots, \vec{u_{n+m}}\} \subset \mathbb{R}^n$$
$$U_- \cap U_+ = \emptyset$$

Mówimy, że zbiory wektorów (wejść) U_- i U_+ są liniowo separowalne, jeśli istnieje jakikolwiek $\vec{w}$ taki, że: $\vec{w} \cdot \vec{u} < 0 : \vec{u} \in U_-$ oraz $\vec{w} \cdot \vec{u} > 0 : \vec{u} \in U_+$. Innymi słowy jeśli istnieje hiperpłaszczyzna, która dzieli zbiory U_- i U_+.

2.1 Przykład

Dla bramki AND mamy:

$$U_- = \{(0,0), (0,1), (1,0)\}, U_+ = \{(1,1)\}$$

$$\vec{w} = (2,2), \theta = 3$$

3 Associatron

Jest to model pamięci asocjacyjnej, skojarzeniowej, który pozwala na kojarzenie danych. W tym modelu dane reprezentujemy wektorami binarnymi. Jako, że de facto tworzymy macierz, to dowolne dane można asocjować.

3.1 Liniowy model pamięci

$$U = \{\vec{u_1}, \vec{u_2}, \ldots, \vec{u_n}\} \in \mathbb{R}^n$$

$$\vec{u_t} = (u_{t1}, u_{t2}, \ldots, u_{tn}), u_{ti} \in \{0, 1\}$$

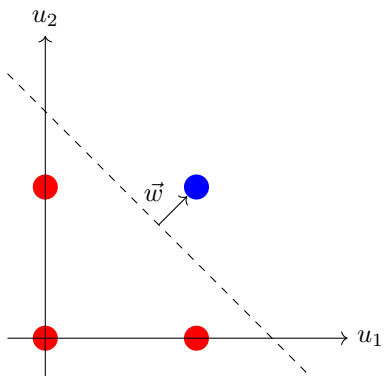

Diagram 2: Liniowa separowalność dla bramki AND

3.2 Cel

Celem modelu jest stworzenie funkcji, która dla danego wektora $\vec{u_t}$ zwróci wektor $\vec{u_s}$, który jest najbardziej podobny do $\vec{u_t}$.

$$\varphi : U \to Y$$
$$U, Y \in \mathbb{R}^n$$

3.3 Macierz wag

$$\vec{y_t} = W \cdot \vec{u_t}$$
$$W = [w_{ij}] = \frac{1}{n} \sum_{t=1}^{N} \vec{y_t} \cdot \vec{u_t}^T$$

Macierz wag jest stała dla danego zbioru wektorów U.

3.4 Liniowa funkcja asocjacyjna

Zakładając, że wszystke wektory w U są ortogonalne ($\vec{u_t} \perp \vec{u_s} \leftrightarrow \vec{u_t} \cdot \vec{u_s} = 0$) to wówczas:

$$\varphi(\vec{u_i}) = W \cdot \vec{u_i} = \frac{1}{n} \sum_{\vec{u_t} \in U} \vec{y_t}(\vec{u_t} \cdot \vec{u_i}) = \frac{1}{n} \cdot \vec{y_i} \cdot \vec{u_i} \cdot \vec{u_i} = \frac{1}{n} \cdot \vec{y_i} \cdot n = \vec{y_i}$$

Czyli dla każdego wektora $\vec{u_i}$ zwracamy wektor $\vec{y_i}$. W istocie w powyższym równaniu szukamy takiego wektora w U, który po pomnożeniu przez wejście, nie będzie ortogonalny, czyli zwróci n.

$$\vec{u_t} \cdot \vec{u_t} = u_t^T u_t = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} = n$$

3.5 Nieliniowa funkcja asocjacyjna

Ograniczenie, że wektory w U są ortogonalne jest bardzo silne. W praktyce nie jesteśmy w stanie tego założyć. Zatem możemy wprowadzić funkcję φ', która pozwala na pewnego rodzaju błąd.

$$\varphi'(\vec{u_t}) = \begin{bmatrix} sgn(x_1) \\ sgn(x_2) \\ \vdots \\ sgn(x_n) \end{bmatrix}, \vec{x} = W \cdot \vec{u_t}$$

$$sgn(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

4 Gradient Descent

Metoda gradient descent pozwala na znalezienie lokalnego minimum funkcji. Gradient funkcji to wektor pochodnych cząstkowych funkcji po każdej zmiennej. Dla funkcji jednej zmiennej gradient to pochodna funkcji po tej zmiennej.

$$\nabla F = \left(\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \ldots, \frac{\partial F}{\partial x_n} \right)$$

W metodzie gradient descent zaczynamy od pewnego punktu (wektora) i iteracyjnie zmieniamy ten wektor w kierunku przeciwnym do gradientu funkcji. Można to sobie wyobrazić jako stanie na górce i chodzenie w kierunku największego spadku.

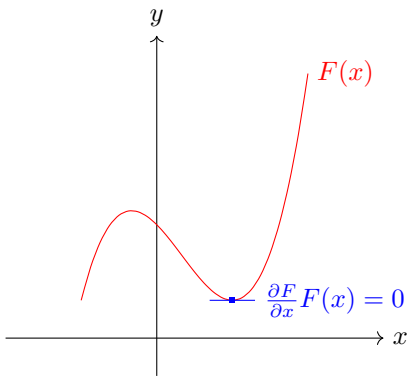

Diagram 3: Przykład działania metody gradient descent

4.1 Algorytm

Instancję algorytmu określa wektor początkowy $\vec{x_0}$, stała c oraz stała ϵ. Dla wektora początkowego $\vec{x} = (x_1, x_2, \ldots x_n)$, oraz wyjściowego po jednym kroku $\vec{x'}$ tak długo jak $\max_{1 \geq i \geq n} |x'_i - x_i| > \epsilon$ wykonujemy następującą operację:

$$\vec{x'} = \vec{x} - c\nabla F(\vec{x})$$

lub równoważnie:

$$x'_i = x_i - c\frac{\partial F}{\partial x_i} F(x)$$

4.2 Problemy

Największym problemem tej metody, jest określenie odpowiednich wartości stałych c oraz ϵ. Wartość c nie może być zbyt mała, bo wtedy algorytm będzie działał bardzo wolno, ale też nie może być zbyt duża, bo wtedy algorytm może nie zbiegać. Wartość ϵ określa dokładność, z jaką chcemy znaleźć minimum. Im mniejsza wartość, tym dokładniejsze minimum, ale też dłuższy czas działania algorytmu. Z kolei wartość $\vec{x_0}$ jest bardzo ważna, bo od niej zależy, czy algorytm zbiegnie do minimum lokalnego czy globalnego.

5 Backpropagation

Backpropagation (backprop) to metoda trenowania sieci neuronowych. Jest to w istocie zastosowanie optymalizacji gradientowej do determinowania wag sieci neuronowej.

Najpierw obliczamy wynik sieci neuronowej. Dla wektora wejściowego $\vec{x}$, czynnika bias b oraz macierzy początkowej wag $\vec{W_0}$ wykonujemy następujące operacje:

$$\vec{y} = \sigma(\vec{x} \cdot \vec{W_0} + b)$$

gdzie σ to funkcja aktywacji. $W_0 \in \mathbb{R}^{n \times m}$ gdzie n to liczba wejść, a m to liczba neuronów w warstwie.

Następnie określamy metrykę błędu F. Klasycznym błędem jest MSE ($\frac{1}{n}\sum_{i=1}^{n}(y_i - t_i)^2$). Następnie mając określoną metrykę, dla każdej warstwy obliczamy gradient błędu i to jak każda waga się przczyniła do błędu.

Zakładając, optymalizację SGD, obliczamy gradient błędu dla każdej warstwy i aktualizujemy wagi za pomocą wzoru:

$$W_{i+1} = W_i - c\frac{\partial F}{\partial W_i} = W_i - c\frac{\partial F}{\partial \hat{y}} \cdot \sigma'(W_i x_i + b_i)$$

gdzie c to współczynnik uczenia.

$$\frac{\partial MSE}{\partial \hat{y}} = \hat{y} - y$$

1. Forward pass
   (a) $a = \sigma(W_1 x + b_1)$ - pierwsza warstwa
   (b) $\hat{y} = \sigma(W_2 a + b_2)$ - druga warstwa
   (c) $\mathcal{L} = MSE(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$ - obliczamy błąd
2. Backprop
   (a) $\frac{\partial \mathcal{L}}{\partial b_2} = \delta_2 = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \sigma'(W_2 a + b_2)$
   (b) $\frac{\partial \mathcal{L}}{\partial W_2} = \delta_2 \cdot a^T$
   (c) $\frac{\partial \mathcal{L}}{\partial b_1} = \delta_1 = (W_2^T \delta_2) \cdot \sigma'(W_1 x + b_1)$
   (d) $\frac{\partial \mathcal{L}}{\partial W_1} = \delta_1 \cdot x^T$
3. Descent
   (a) $W_1 := W_1 - c\frac{\partial \mathcal{L}}{\partial W_1}$
   (b) $W_2 := W_2 - c\frac{\partial \mathcal{L}}{\partial W_2}$
   (c) $b_1 := b_1 - c\delta_1$
   (d) $b_2 := b_2 - c\delta_2$

6 CNN

6.1 Konwolucja

Wejściem jest macierz $u_{i,j}$ o wymiarach $N \times N$. Rdzeniem jest macierz $w_{i,j}$ o wymiarach $2H+1 \times 2H+1$. Operacja konwolucji zwraca nam macierz $x_{i,j}$ o wymiarach $N \times N$. W konwolucji stosuje się funkcję aktywacji f, tutaj:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$x_{i,j} = f(\sum_{k=-H}^{H}\sum_{l=-H}^{H} u_{i+k,j+l} w_{k,l} - \theta)$$

6.2 Pooling

Wejściem jest zkonwolucjonowana macierz $x_{i,j}$ o wymiarach $N \times N$. Wyjściem jest macierz $y_{i,j}$ o identycznych wymiarach.

$$y_{i,j} = f(\frac{1}{(2K+1)^2}\sum_{k=-K}^{K}\sum_{l=-K}^{K} x_{i+k,j+l} - \theta)$$

7 MB

Mamy wektor wejściowy x o wymiarach N. Wektor jest inicjalizowany losowo.

$$u_i(t) = \sum_{j=1}^{n} w_{i,j} x_j(t) - \theta_i$$

$$f(u_i(t)) = \frac{1}{1 + e^{-u_i(t)/T}}$$

$$x_i(t+1) = \begin{cases} 1 & \text{if } f(u_i(t)) \geq U(0,1) \\ 0 & \text{otherwise} \end{cases}$$

8 Transformer

Dla każdego słowa mamy wektor wejściowy x_1, x_2, …, x_n. Tworzymy trzy projekcje wektorów wejściowych:

$$q_i = W_q x_i k_i = W_k x_i v_i = W_v x_i$$

$$y_i = \sum_{j=1} \frac{q_i \cdot k_j}{\sqrt{d_k}} \cdot v_j$$