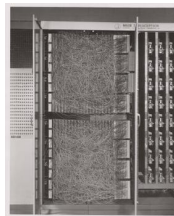
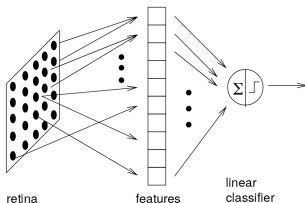


Perceptron

Reguły uczenia

Perceptron Rosenblatta (1958)

Klasyfikator neuronowy Mark I do rozpoznawania znaków alfanumerycznych, wzorowany na biologicznej percepcji (układ wzrokowy)



Trzy warstwy:

- **S-units:** wejście, siatkówka oka, np. fotokomórki 20 x 20
- **A-units:** asocjacyjne, zbierające dane z większych obszarów (obliczanie cech), 512 jednostek
- **R-units:** liniowy klasyfikator uczący, 8 wyjść

Rys: [wikipedia.org](https://www.wikipedia.org)

Martin Riedmiller, *Machine Learning (lectures)*

- $S_i = \{-1, +1\}$ sygnał docierający do elementów sensorycznych
- połączenia $c_{ij} = \{-1, 0, +1\}$ elementów S_j i A_i , przypadkowo rozrzucone w pewnym obszarze, nie ulegają zmianom, realizują wstępne przetwarzanie (ekstrakcja cech)

$$A_i = \begin{cases} +1, & \text{dla } \sum_j c_{ij} S_j \geq \Theta_i \\ -1, & \text{dla } \sum_j c_{ij} S_j < \Theta_i \end{cases}$$

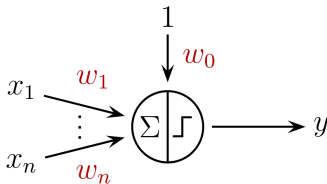
- sygnał wyjściowy

$$R_i = \begin{cases} +1 & \sum_j w_{ij} A_j > N\kappa \\ -1 & \sum_j w_{ij} A_j < -N\kappa \\ 0 & \text{w pozostałych przypadkach} \end{cases}$$

- wagi w_{ij} realizowane przez potencjometry
- proces uczenia - mechaniczna regulacja ich wartości

Perceptron współczesny

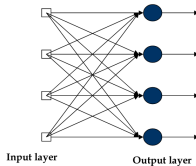
- Zwykle przez „perceptron” rozumie się teraz jeden neuron z wieloma wejściami (bez jednostek S, bo tu nie ma adaptacji).
- **Perceptron prosty**



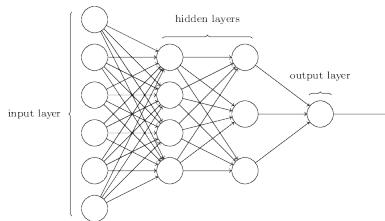
$$y = f \left(\sum_i w_i x_i + w_0 \right)$$

Rys: Martin Riedmiller, Machine Learning (lectures)

Single layer perceptron



Multilayer perceptron (MLP)

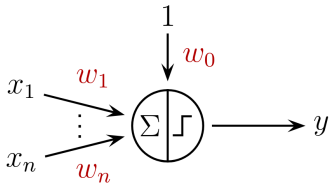


Rys: M. Bennamoun, Neural Computation (lecture)
computersciencewiki.org

Neuron binarny

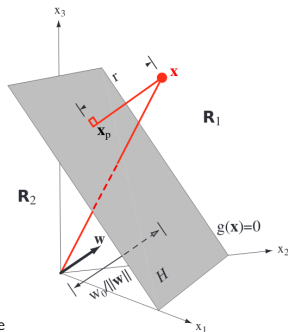
Perceptron prosty (binarny)

$$y = \begin{cases} 1, & \text{gdy } \sum_i w_i x_i + w_0 \geq 0 \\ 0, & \text{gdy } \sum_i w_i x_i + w_0 < 0 \end{cases}$$



Interpretacja geometryczna

- $x_1, \dots, x_n$ to współrzędne punktu w n -wymiarowej przestrzeni ($\mathbf{x} \in \mathbb{R}^n$)
- równanie $\sum x_i w_i + w_0 = 0$ definiuje płaszczyznę (hiperpłaszczyznę) rozdzielającą przestrzeń wejściową na dwie części R_1 i R_2
- perceptron prosty dzieli wektory x_i na te leżące poniżej płaszczyzny decyzyjnej ($y = 0$) oraz leżące powyżej płaszczyzny ($y = 1$)



Rys: Duda and Hart, Pattern Classification

Zadanie klasyfikacji binarnej

- **Klasyfikacja binarna** - przypisanie obiektów do jednej z 2 klas
- **Dane treningowe**: zbiór n przypadków $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, każdy przypisany do jednego z dwóch zbiorów $\mathbb{P}$ lub $\mathbb{N}$
- **Uczenie**: procedura doboru wag $\mathbf{w}$ i wartości progowej w_0 tak aby perceptron zwracał wartość 1 dla wszystkich $\mathbf{x}_i$ ze zbioru $\mathbb{P}$, zaś wartość 0 dla wszystkich $\mathbf{x}_i \in \mathbb{N}$
- założenie: dane są spójne, tzn. $\mathbb{P} \cap \mathbb{N} = \emptyset$

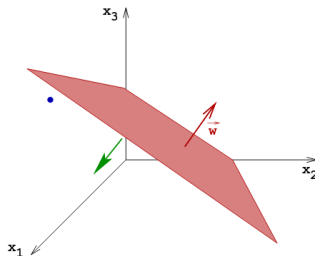
Uczenie perceptronu - idea

Niech $\mathbf{x}$ należy do zbioru $\mathbb{P}$. Jeżeli perceptron popełnia błąd to

$$\sum w_i x_i + w_0 < 0$$

Jak zmienić $\mathbf{w}$ i w_0 aby zniwelować błąd?

- zwiększyć w_0



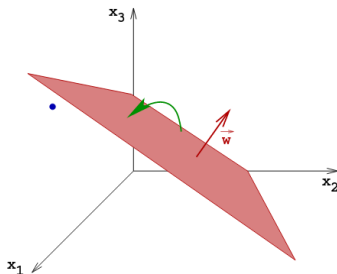
Uczenie perceptronu - idea

Niech $\mathbf{x}$ należy do zbioru $\mathbb{P}$. Jeżeli perceptron popełnia błąd to

$$\sum w_i x_i + w_0 < 0$$

Jak zmienić $\mathbf{w}$ i w_0 aby zniwelować błąd?

- zwiększyć w_0
- jeśli $x_i > 0$ to zwiększyć w_i
- jeżeli $x_i < 0$ to zmniejszyć w_i



Rys: Riedmiller, Machine Learning (lectures)

obrót płaszczyzny ze zmianą $\mathbf{w}$

Algorytm uczenia perceptronu

Algorytm 1 Algorytm uczenia perceptronu

Input: zbiór wektorów należących do jednego z dwóch zbiorów $\mathbb{P}$ i $\mathbb{N}$

Output: perceptron klasyfikujący wszystkie przypadki (jeżeli istnieje)

- 1: zainicjuj wagi $\mathbf{w}$ oraz wartość progową w_0 (np. małe losowe wartości w okolicy 0)
- 2: **while** istnieje błędnie klasyfikowany $\mathbf{x}$ **do**
- 3: **if** $\mathbf{x} \in \mathbb{P}$ **then**
- 4: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$
- 5: $w_0 \leftarrow w_0 + 1$
- 6: **else**
- 7: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}$
- 8: $w_0 \leftarrow w_0 - 1$
- 9: **return** $\mathbf{w}, w_0$

Algorytm uczenia perceptronu

- Jeżeli istnieje rozwiązanie (tj. problem jest **liniowo separowany**) to algorytm je odnajdzie w skończonej liczbie kroków
- Możliwe cykle, gdy wektor wag się powtórzy w sekwencji uczenia to problem jest nierozwiązywalny
- Liczba błędów nie maleje monotonicznie - kolejna modyfikacja może popsuć klasyfikację poprzednio nauczonych przypadków
- Jak znaleźć najlepsze możliwe rozwiązanie nawet gdy problem nie jest liniowo separowalny?
- **Algorytm kieszeniowy** - uczenie perceptronu z zapamiętaniem wag dla których popełniono najmniej błędów

Algorytm 2 Algorytm kieszonkowy

Input: zbiór wektorów należących do jednego z dwóch zbiorów $\mathbb{P}$ i $\mathbb{N}$

Output: perceptron klasyfikujący przypadki do dwóch klas

- 1: zainicjuj losowo wagi $\mathbf{w}$ i w_0
 - 2: $t \leftarrow 0$, $t' \leftarrow t$, $\mathbf{w}' \leftarrow \mathbf{w}$, $w'_0 \leftarrow w_0$
 - 3: **for** losowo wybranego $\mathbf{x} \in \mathbb{P} \cup \mathbb{N}$ **do**
 - 4: **if** $\mathbf{x}$ poprawnie klasyfikowany **then**
 - 5: $t \leftarrow t + 1$
 - 6: **else**
 - 7: **if** $t > t'$ **then**
 - 8: $t' \leftarrow t$, $\mathbf{w}' \leftarrow \mathbf{w}$, $w'_0 \leftarrow w_0$
 - 9: $t \leftarrow 0$
 - 10: wykonaj aktualizację wag perceptronu
 - 11: **return** $\mathbf{w}'$, w'_0
-

Algorytm kieszonkowy

- Algorytm kieszonkowy ma za zadanie znaleźć najlepsze możliwe rozwiązanie nawet w przypadku problemów, które nie są liniowo separowalne
- Zapamiętuje tylko wagi ostatniego poprawnego przypadku, więc istnieje możliwość zignorowania wcześniejszego, lepszego rozwiązania. Przeciwdziała temu alg. z zapadką, jednak wymaga on więcej nakładów obliczeniowych.
- **Algorytm kieszeniowy z zapadką** - modyfikacja algorytmu kieszeniowego, gdzie zapamiętywany jest zwycięzca tylko wtedy, gdy klasyfikuje poprawnie więcej przypadków treningowych

Algorytm 3 Algorytm kieszonkowy z zapadką

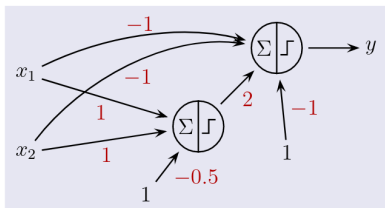
Input: zbiór wektorów należących do jednego z dwóch zbiorów $\mathbb{P}$ i $\mathbb{N}$

Output: perceptron klasyfikujący przypadki do dwóch klas

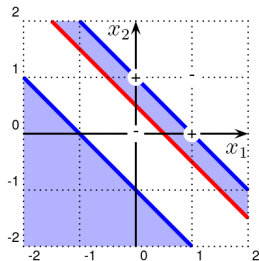
- 1: zainicjuj losowo wagi $\mathbf{w}$ i w_0
- 2: $t \leftarrow 0$, $t' \leftarrow t$, $\mathbf{w}' \leftarrow \mathbf{w}$, $w'_0 \leftarrow w_0$
- 3: **for** losowo wybranego $\mathbf{x} \in \mathbb{P} \cup \mathbb{N}$ **do**
- 4: **if** $\mathbf{x}$ poprawnie klasyfikowany **then**
- 5: $t \leftarrow t + 1$
- 6: **else**
- 7: **if** $t > t'$ **then**
- 8: **if** $\mathbf{w}$ i w_0 klasyfikują poprawnie więcej przypadków niż $\mathbf{w}'$ i w'_0 **then**
- 9: $t' \leftarrow t$, $\mathbf{w}' \leftarrow \mathbf{w}$, $w'_0 \leftarrow w_0$
- 10: $t \leftarrow 0$
- 11: wykonaj aktualizację wag perceptronu
- 12: **return** $\mathbf{w}'$, w'_0

Czego może się nauczyć perceptron?

- Perceptron binarny potrafi rozwiązać wyłącznie problemy liniowo separowalne
- Przykład XOR - nie jest liniowo separowalny
- Sieci połączonych neuronów mają większe możliwości
- Problem XOR można rozwiązać za pomocą 2 perceptronów prostych



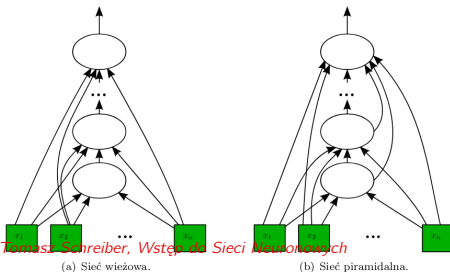
Rys: Riedmiller, Machine Learning (lectures)



Algorytm wieżowy i piramidalny

Przykład algorytmów rozrastających się

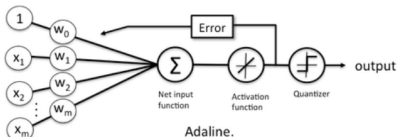
1. Wytrenuj pojedynczy perceptron (np. algorytmem kieszonkowym) na danych treningowych
2. Jeżeli nie uzyskano zadowalającego rezultatu to dodaj perceptron, którego wejściem będzie wyjście poprzedniego neuronu (alg. **wieżowy**) lub wyjścia wszystkich poprzednich neuronów (alg. **piramidalny**). Wróć do punktu 1.



Maja Czoków, Jarosław Piersa, Thomas Schreiber, Wstęp do Sieci Neuronowych

Adaline - Adaptive Linear Element

Adeline (Widrow, 1959) - jednowarstwowa sieć składająca się z perceptronów prostych. Realizacja sprzętowa z użyciem memistorów.



Reguła uczenia Widrowa-Hoffa

$$\Delta w_k = \lambda \left(y - \sum_i w_i x_i \right) x_k$$

gdzie y jest oczekiwanym sygnałem wyjściowym

Rys: <https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html>

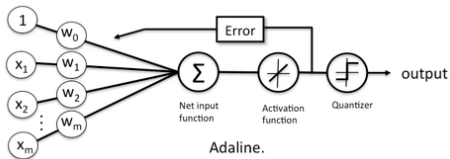
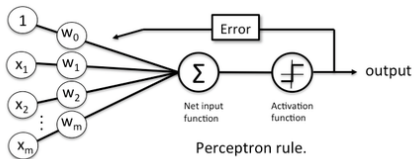
Reguła uczenia

- Reguła uczenia Widrowa-Hoffa dąży do minimalizacji błędu kwadratowego aktywacji neuronu $\mathbf{w}^T \mathbf{x}$ względem pożądanego sygnału wyjściowego y

$$E_{MSE}(\mathbf{x}, y) = \frac{1}{2} \left(y - \sum_i w_i x_i \right)^2$$

- funkcja kosztu E jest ciągła, więc możliwy jest trening metodą spadku gradientu
- dla neuronów liniowych reguła jest równoważna regule delta a Adeline rozwiązuje problem aproksymacji, gdzie $y_i \in \mathbb{R}$
- dla neuronów z wyjściem progowym Adeline staje się klasyfikatorem, gdzie $y_i = \{-1, +1\}$
- Madeline - wielowarstwowa wersja Adeline

Adeline vs. Perceptron rule



Rys: <https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html>

Reguła delta

- **Reguła delta** - uogólniona reguła uczenia perceptronu dla ciągłych (różniczkowalnych) funkcji aktywacji $f(x)$

$$\Delta w_i = \lambda (y - f(\mathbf{x})) f'(\mathbf{x}) x_i$$

$\lambda > 0$ współczynnik uczenia

y to pożądany sygnał dla wejścia $\mathbf{x}$

$y \in \mathbf{R}$ dla problemu aproksymacji,

$y = \{-1, +1\}$ lub $y = \{0, 1\}$ dla klasyfikacji binarnej

- minimalizacja błędu kwadratowego E w kierunku największego spadku gradientu

$$\Delta w = -\lambda \nabla_w E$$

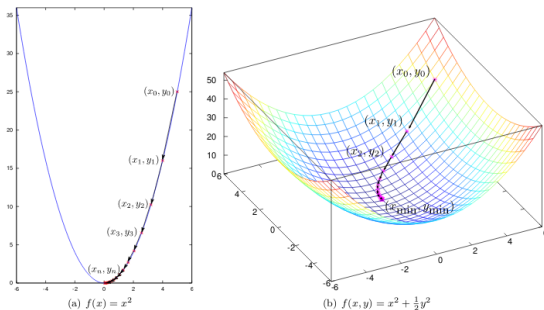
Dla błędu MSE

$$\Delta w_i = -\lambda \frac{\partial}{\partial w_i} E_{MSE} = -\lambda \frac{\partial}{\partial w_i} \frac{1}{2} (y - f(\mathbf{x}))^2 = \lambda (y - f(\mathbf{x})) f'(\mathbf{x}) x_i$$

Algorytm spadku gradientu

1. ustaw punkt startu $\mathbf{w}_0$ oraz stałą uczenia $\lambda > 0$
2. dopóki nie spełnione kryterium stopu wykonuj

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \nabla E(\mathbf{w})$$



Rys: Maja Czoków, Jarosław Piersa, Tomasz Schreiber, Wstęp do Sieci Neuronowych

Aspekty uczenia spadkiem gradientu

- Jeżeli istnieją minima lokalne to istnieje ryzyko utknięcia algorytmu w tym minimum
- Powtórzenie kilkakrotne uczenia z różnymi punktami startowymi może pozwolić uniknąć minimów lokalnych
- Trajektoria zależy od punktu startowego
- Kryterium stopu algorytmu:
 - ilość kroków uczenia lub inne ograniczenie czasowe
 - osiągnięcie zadowalającego poziomu dokładności $E < \epsilon$
 - niewielkie zmiany $\|\Delta w\| < \epsilon$
- Dobór stałej uczenia λ . Gdy za duża to rozwiązanie może być pominięte (przeskoczone) a nawet proces może być rozbieżny, gdy za mała to uczenie będzie powolne. Nie musi być wartością stałą, np. może być zmniejszana w czasie treningu lub dobierana zależnie od wypukłości funkcji

Funkcje aktywacji

- **identyczność**

$$f(x) = x \quad f'(x) = 1$$

- funkcja **liniowa** - nieograniczona $f(x) \in \mathbb{R}$

$$f(x) = ax + b \quad f'(x) = a$$

- **progowa** unipolarna - nieciągła, nieróżniczkowalna

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

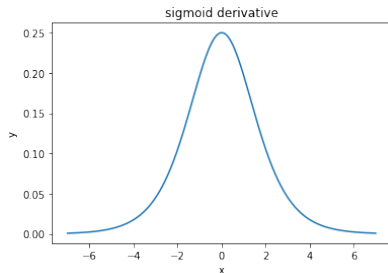
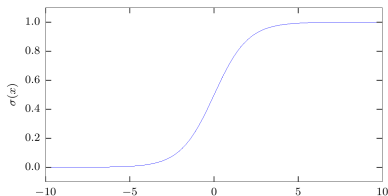
- progowa bipolarna

$$f(x) = \begin{cases} 1, & \text{gdy } x \geq a \\ -1, & \text{gdy } x < a \end{cases}$$

Funkcje aktywacji

- **sigmoidalna** (unipolarna) - ograniczona $f(x) \in (0, 1)$

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

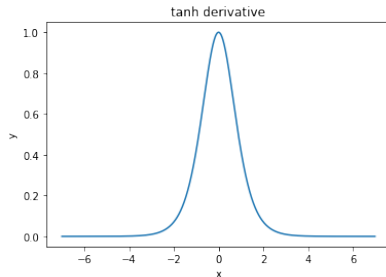
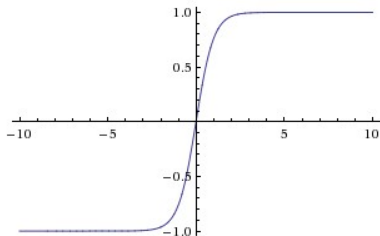


Rys: <http://cs231n.github.io/>

Funkcje aktywacji

- **tangens hiperboliczny** (bipolarna) - ograniczona
 $f(x) \in (-1, +1)$

$$f(x) = \tanh x = \frac{1 - e^{-x}}{1 + e^{-x}} \quad f'(x) = 1 - f^2(x)$$



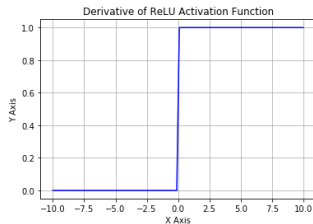
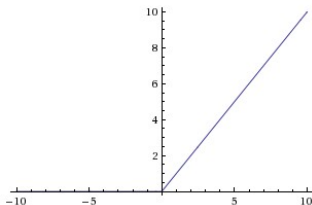
Rys: <http://cs231n.github.io/>,
<https://www.kaggle.com/moriano/a-showcase-of-how-relus-can-speed-up-the-learning>

Funkcje aktywacji

- **ReLU** (Rectified Linear Unit) - fragmentami ciągła, brak pochodnej w 0

$$f(x) = \max(0, x) = \begin{cases} x, & \text{gdy } x \geq 0 \\ 0, & \text{gdy } x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & \text{gdy } x > 0 \\ 0, & \text{gdy } x < 0 \end{cases}$$

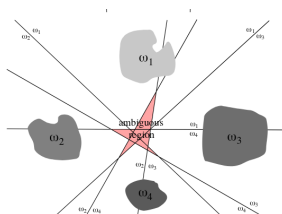
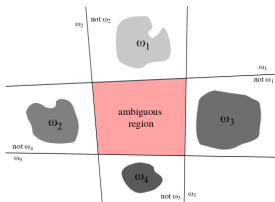


Rys: <http://cs231n.github.io/>,
<https://learnopencv.com/understanding-activation-functions-in-deep-learning/>

Klasyfikacja wielu klas

W przypadku wieloklasowym, gdy $y_i = 1, 2, 3, \dots, k$

- k klasyfikatorów binarnych po jednym na klasę
Pojedynczy perceptron prosty może być nauczony aby separować przypadki z pojedynczej klasy od pozostałych (,
- $k(k - 1)/2$ klasyfikatorów binarnych dla każdej pary klas

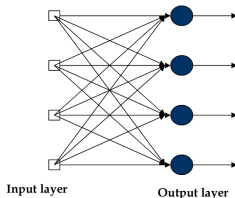


Rys: Duda and Hart, Pattern Recognition

Maszyna liniowa

Jednowarstwowa sieć liniowa klasyfikująca do k grup

- sieć jednowarstwowa - liczba wyjść równa liczbie klas



- funkcja dyskryminująca

$$f_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

- klasyfikacja: przypisanie wektorowi $\mathbf{x}$ klasy i odpowiadającej wyjściu f_i o największej wartości

$$f_i(\mathbf{x}) > f_k(\mathbf{x}) \quad \text{dla każdego} \quad i \neq k$$

Algorytm 4

Uczenie maszyny liniowej

Input: zbiór wektorów należących do jednej z k klas

Output: klasyfikator separujący wektory treningowe

- 1: zainicjuj losowo wagi $\mathbf{w}_i$
 - 2: **for** losowo wybranego $\mathbf{x}$ **do**
 - 3: **if** $\mathbf{x}$ należący do klasy i jest niepoprawnie przypisany do klasy j **then**
 - 4: $\mathbf{w}_i \leftarrow \mathbf{w}_i + \mathbf{x}$
 - 5: $\mathbf{w}_j \leftarrow \mathbf{w}_j - \mathbf{x}$
 - 6: **return** $\mathbf{w}, w_0$
-