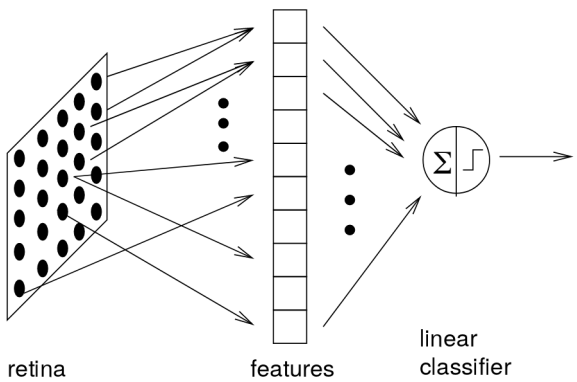
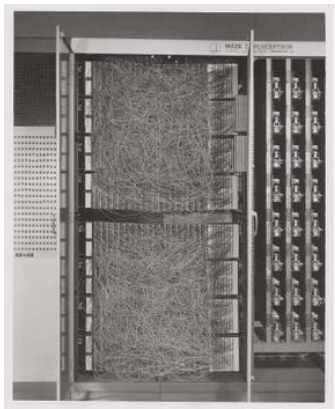
Perceptron

Reguły uczenia

Perceptron Rosenblatta (1958)

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





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
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)

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

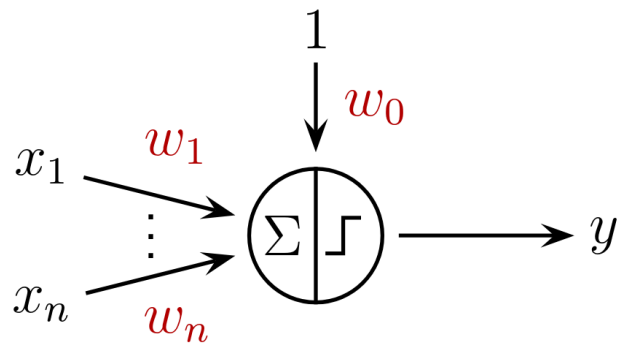
- sygnał wyjściowy

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

- wagi w_{ij} - potencjometry, uczenie - mechaniczna regulacja ich wartości

Perceptron współczesny

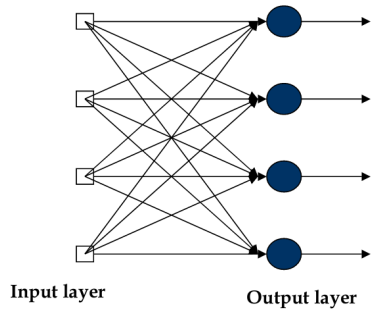
- Zwykle przez „perceptron” rozumie się teraz jeden neuron z wieloma wejściami (bez jednostek S, bo tu nie ma adaptacji).
- **Perceptron prosty**



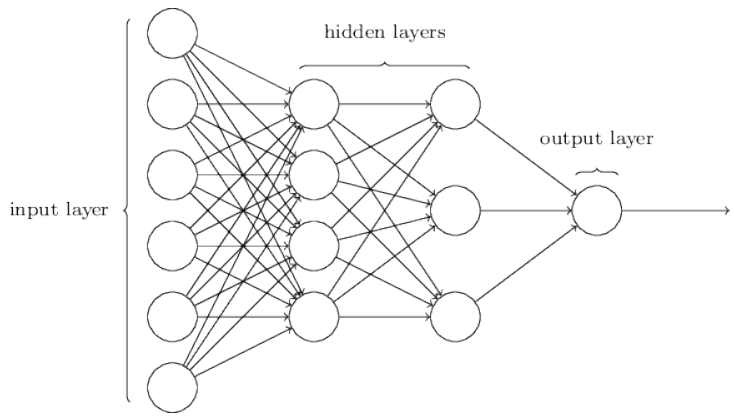
$$y = f\left(\sum_i w_i x_i + w_0\right)$$

Rys: Martin Riedmiller, Machine Learning (lectures)

Single layer perceptron



Multilayer perceptron (MLP)

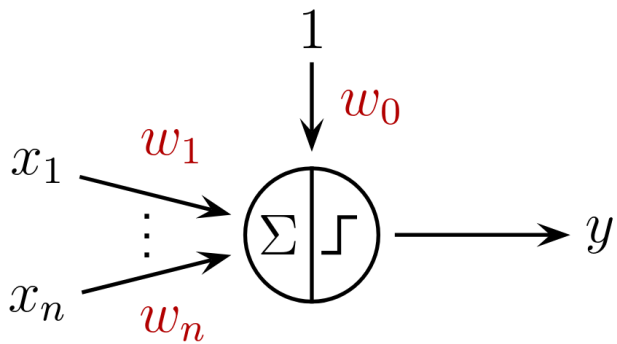


Rys: M. Bennamoun, Neural Computation (lecture)
computersciencewiki.org

Neuron binarny

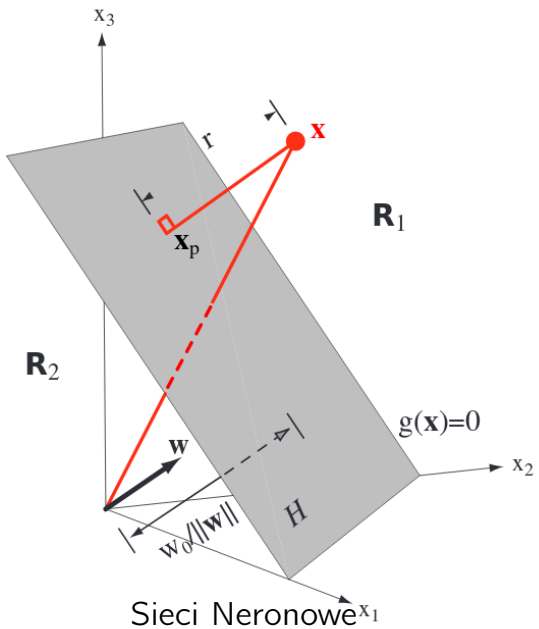
Perceptron prosty (binarny)

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



Interpretacja geometryczna

- $x_1, \ldots, x_n$ to punkty 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 dzieli wektory x_i na te leżące poniżej płaszczyzny decyzyjnej ($y < 0$) oraz leżące powyżej płaszczyzny ($y \geq 0$)



Rys: Duda and Hart, Pattern Classification

Zadanie klasyfikacji

- **Klasyfikacja binarna** - przypisanie obiektów do jednej z 2 klas
- **Dane treningowe**: zbiór n przypadków $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, każdy przypisany do jednego z dwóch zbiorów $\mathbb{P}$ lub $\mathbb{N}$
- **Uczenie**: dobrać wagi $\mathbf{w}$ i wartość progową 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. $\mathbb{P} \cap \mathbb{N} = \emptyset$

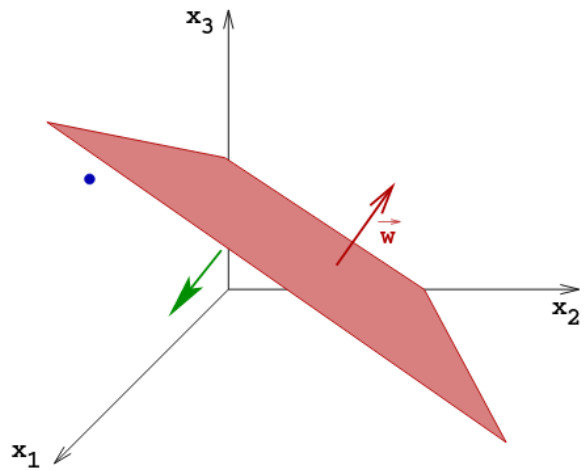
Uczenie perceptronu - idea

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

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

Jak zmienić **w** i w_0 aby zniwelować błąd?

- zwiększyć w_0



przesunięcie płaszczyzny ze zwiększeniem w_0

Rys: Riedmiller, Machine Learning (lectures)

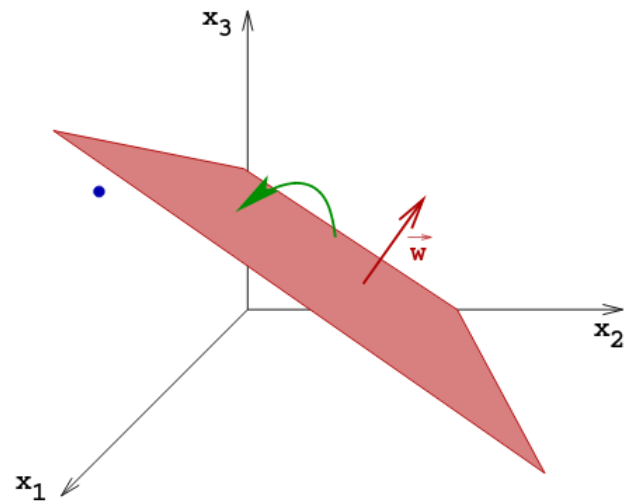
Uczenie perceptronu - idea

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

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

Jak zmienić **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



obrót płaszczyzny ze zmianą **w**

Rys: Riedmiller, Machine Learning (lectures)

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 (problem jest **liniowo separowany**) to algorytm je odnajdzie w skończonej liczbie kroków
- Możliwe cykle: gdy wektor wag się powtórzy to problem jest nierozwiązywalny
- Liczba błędów nie maleje monotoniczne - 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: $t \leftarrow 0, \quad t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
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: $t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
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 rozwiązanie. Przeciwdziała temu alg. z zapadką, jednak wymaka 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

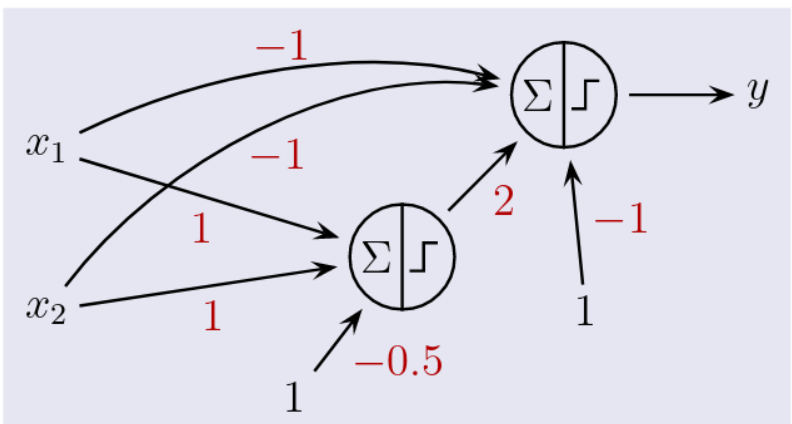
Input: zbiór wektorów należących do jednego z dwóch zbiorów $\mathbb{P}$ i $\mathbb{N}$

Output: perceptron klasyfikujący przypadki do dwóch klas

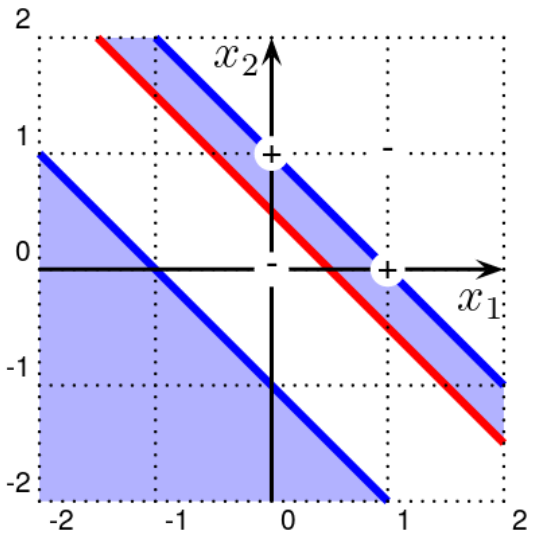
1: zainicjuj losowo wagi $\mathbf{w}$ i w_0
2: $t \leftarrow 0, \quad t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
3: **for** losowo wybranego $\mathbf{x} \in \mathbb{P} \cup \mathbb{N}$ **do**
4: **if** $\mathbf{x}$ poprawnie klasyfikowany **then**
5: $t \leftarrow t + 1$
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: $t' \leftarrow t, \quad \mathbf{w}' \leftarrow \mathbf{w}, \quad w_0' \leftarrow w_0$
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



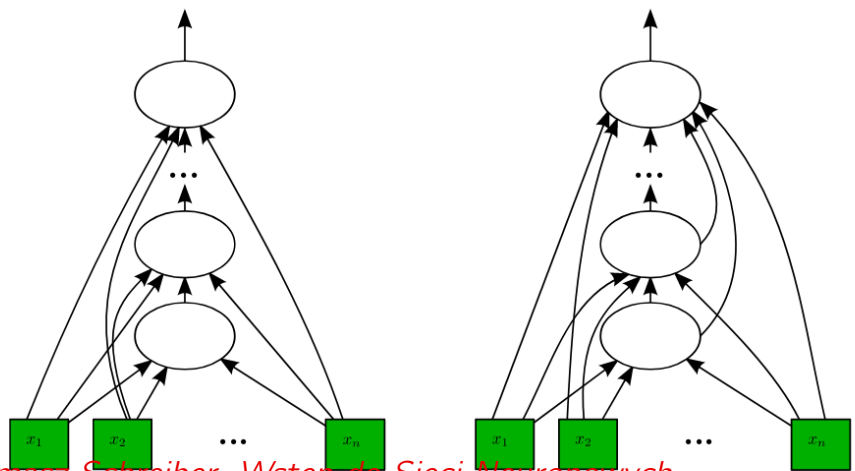
Rys: Riedmiller, Machine Learning (lectures)



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.

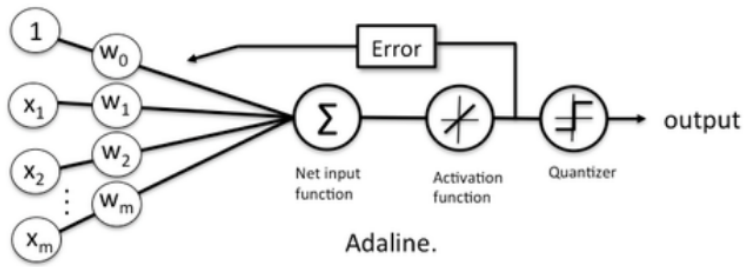


(a) Sieć wieżowa. (b) Sieć piramidalna.

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

Adaline - Adaptive Linear Element

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



Adaline.

Reguła uczenia Widrowa-Hoffa

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

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

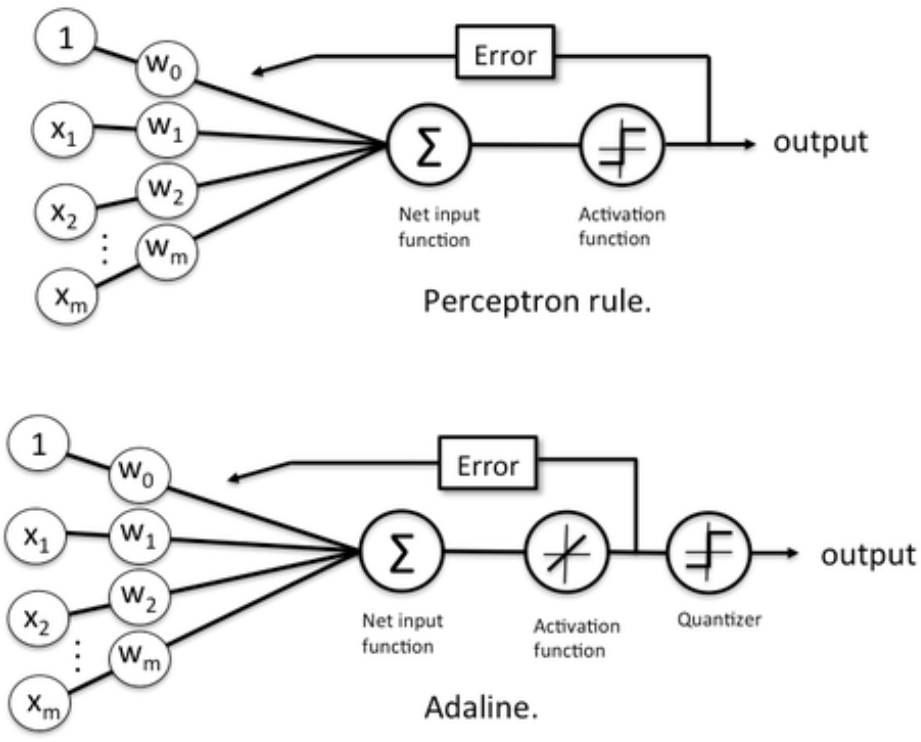
Reguła uczenia

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

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

- Ponieważ funkcja celu jest ciągła to możliwy jest trening za pomocą metody największego 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 $f(x)$

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

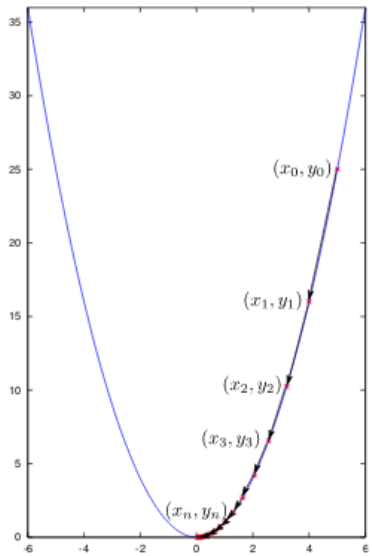
  $\lambda > 0$ współczynnik uczenia
  y to pożądany sygnał, $y \in \mathbf{R}$ dla aproksymacji,
  $y = \{-1, +1\}$ lub $y = \{0, 1\}$ dla klasyfikacji binarnej

- minimalizacja błędu kwadratowego E w kierunku największego spadku gradientu

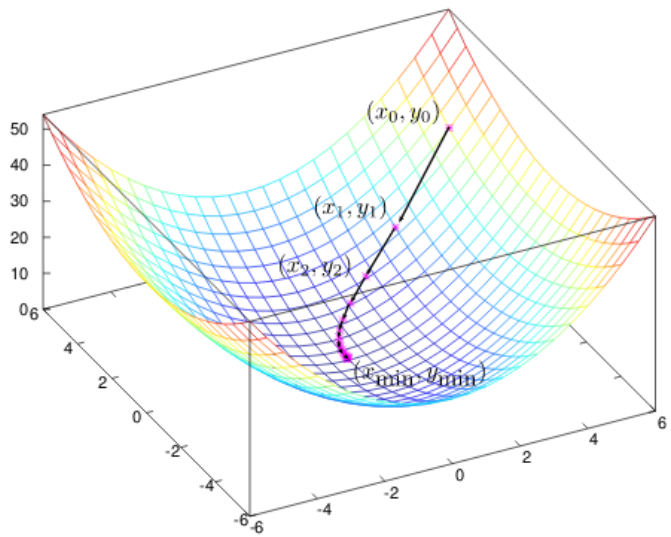
$$\Delta w = -\lambda \nabla_w E$$

Algorytm spadku gradientu

$$w \leftarrow w - \lambda \nabla E(w)$$



(a) $f(x) = x^2$



(b) $f(x, y) = x^2 + \frac{1}{2}y^2$

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

Aspekty uczenia

- Jeżeli istnieją minima lokalne to istnieje ryzyko utknięcia algorytmu w tym minimum
- Powtórzenie kilkukrotne 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 $||\Delta w|| < \epsilon$
- wartość stałej uczenia λ, gdy za duża to rozwiązanie może być pominięte, gdy za mała to uczenie będzie powolne. Nie musi być wartością stałą, np. może być zmniejszana w czasie treningu.

Funkcje aktywacji I

- **identyczność**

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

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

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

- **progowa** unipolarna - nieciągła, nieróżniczkowalna

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

- progowa bipolarna

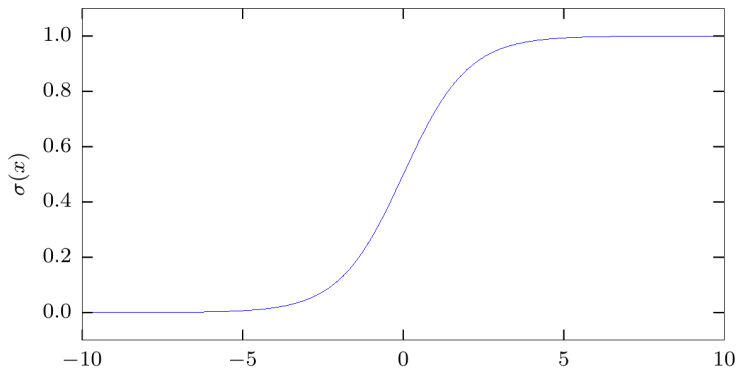
$$f(x) = \begin{cases} 1, & \text{gdy} \quad x \geq a \\ -1, & \text{gdy} \quad x < a \end{cases}$$

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

Funkcje aktywacji II

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

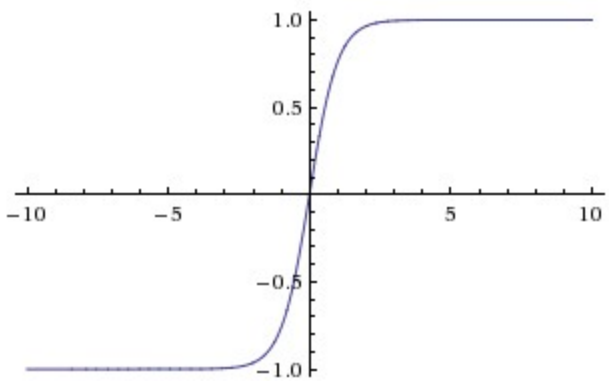
$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x)(1 - f(x))$$



Funkcje aktywacji III

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

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

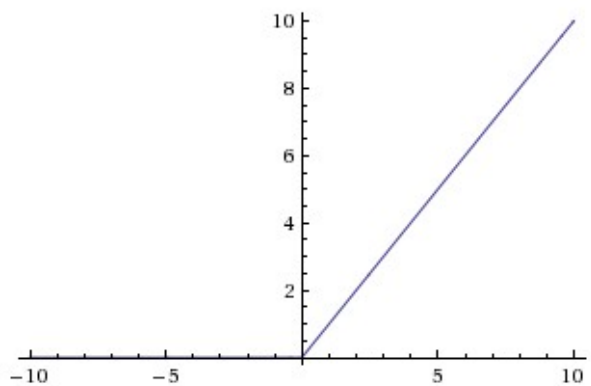


Funkcje aktywacji IV

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

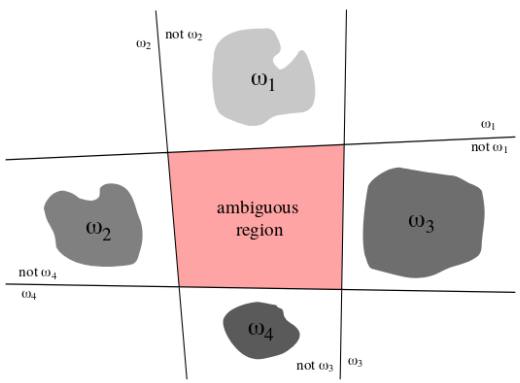
$$f(x) = \max(0, x) = \begin{cases} x, & \text{gdy} \quad x \geq 0 \\ 0, & \text{gdy} \quad x < 0 \end{cases}$$

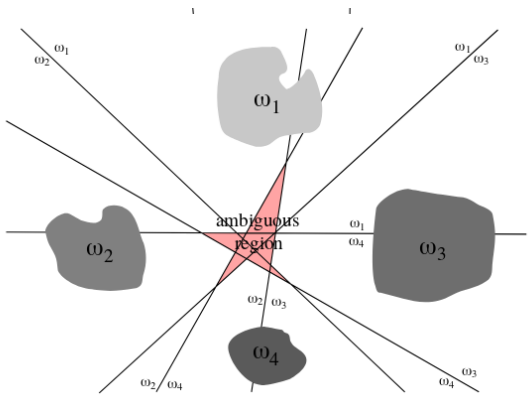
$$f'(x) = \begin{cases} 1, & \text{gdy} \quad x > 0 \\ 0, & \text{gdy} \quad x < 0 \end{cases}$$



Klasyfikacja wielu klas

- W przypadku wieloklasowym $y_i = 1, 2, 3, ..., k$
- Pojedynczy perceptron prosty może być nauczony aby separować przypadki z pojedynczej klasy od pozostałych (po jednym perceptronie na klasę), k klasyfikatorów binarnych
- Bardziej wymagające podejście $k(k-1)/2$ klasyfikatorów binarnych dla każdej pary klas

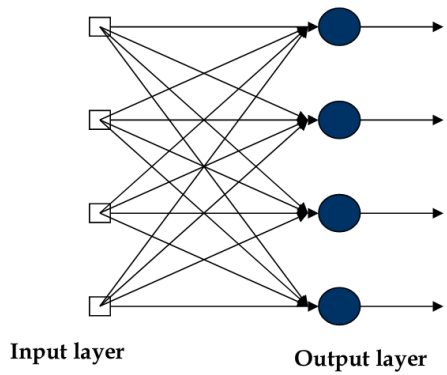




Rys: Duda and Hart, Pattern Recognition

Maszyna liniowa

- 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 **x do**
3: **if** **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** $\mathbf{w}, w_0$