O uczeniu maszynowym i sieciach neuronowych

Paweł Rychlikowski

Instytut Informatyki UWr

18 maja 2023

Przykładowe dane uczące

Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 19.2 Examples for the restaurant domain.

Przykładowe drzewo

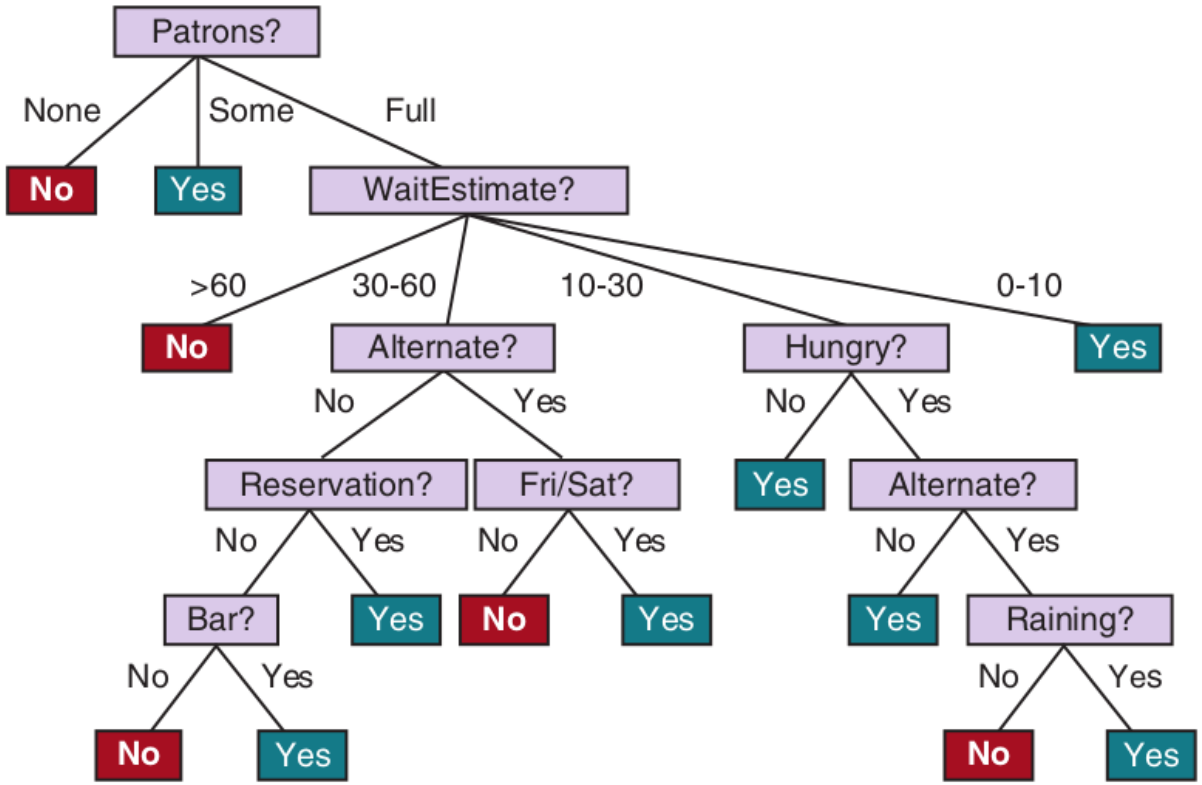


Figure 19.3 A decision tree for deciding whether to wait for a table.

Cel: indukcja drzew decyzyjnych

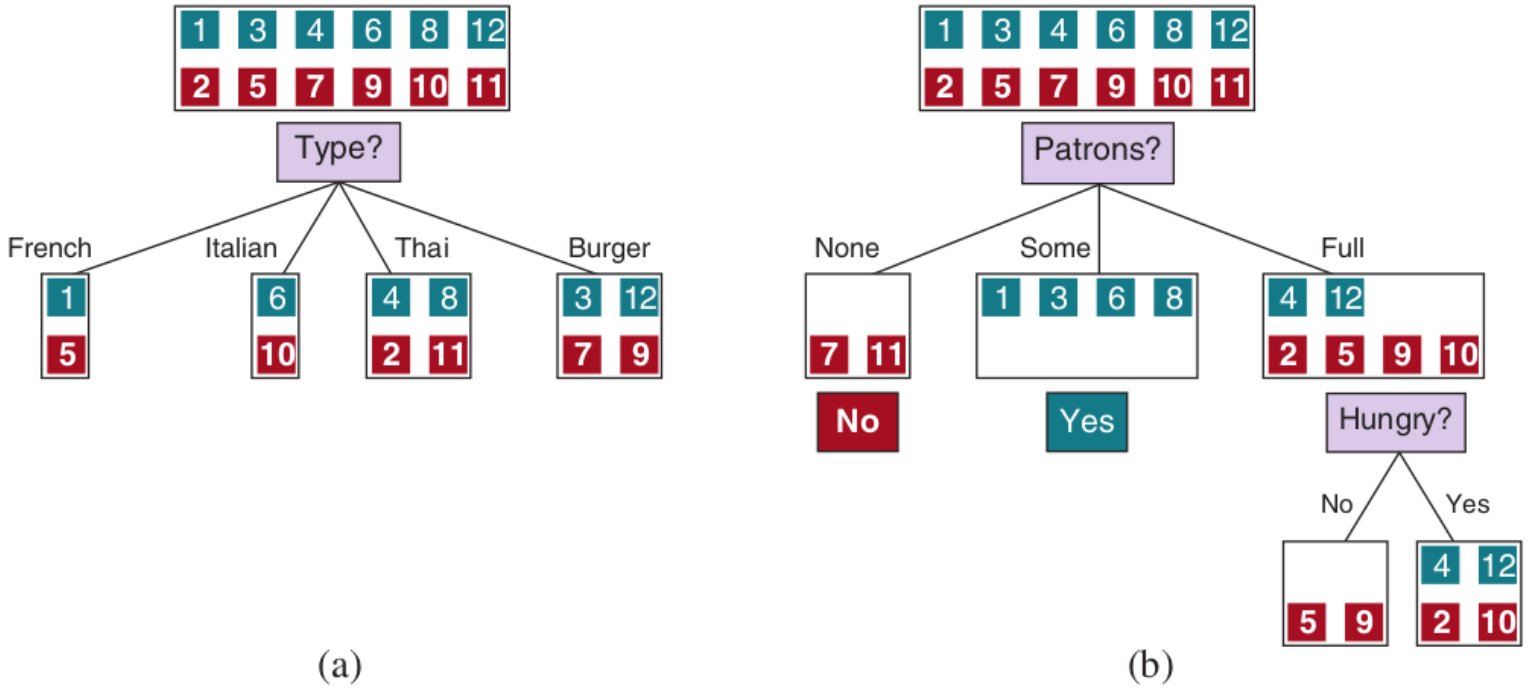


Figure 19.4 Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining. (a) Splitting on *Type* brings us no nearer to distinguishing between positive and negative examples. (b) Splitting on *Patrons* does a good job of separating positive and negative examples. After splitting on *Patrons*, *Hungry* is a fairly good second test.

Szkic algorytmu

Algorytm rekurencyjny, cztery przypadki:

1. Wszystkie przypadki należą do tej samej klasy: utwórz liść z nazwą klasy
2. Są przypadki pozytywne i negatywne, wybierz atrybut, utwórz rekurencyjnie poddrzewa
3. Nie ma żadnego elementu z danych uczących w tym węźle. Wówczas tworzymy liść z wartością z węzła z poziomu wyżej (najczęstsza klasa w tym węźle)
4. Nie ma już atrybutów do sprawdzenia, a mamy przypadki pozytywne i negatywne. Tworzymy liść z odpowiedzią (najczęstsza klasa)

Indukcja drzew decyzyjnych

function LEARN-DECISION-TREE($examples$, $attributes$, $parent\_examples$) **returns** a tree

if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
  $tree \leftarrow$ a new decision tree with root test A
  for each value v of A **do**
    $exs \leftarrow \{e : e \in examples \text{ and } e.A = v\}$
    $subtree \leftarrow$ LEARN-DECISION-TREE(exs, $attributes - A$, $examples$)
    add a branch to $tree$ with label $(A = v)$ and subtree $subtree$
  return $tree$

Figure 19.5 The decision tree learning algorithm. The function IMPORTANCE is described in Section **??**. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

Jak oceniać wybór atrybutu

Załóżmy, że atrybut A dzieli zbiór przykładów E na rozłączne podzbiory $E_1, \ldots, E_i \ldots, E_d$, w każdym z nich mamy p_i pozytywnych, i n_i negatywnych przykładów.
Ponadto: $p = p_1 + \cdots + p_d$ oraz $n = n_1 + \cdots + n_d$
Zdefiniujemy $B(p, n)$ jako **bałagan**, związany z niejednorodnością zbioru (są przykłady pozytywne i negatywne).

Information gain

Zysk to $B(p, n) - \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B(p_k, n_k)$

Entropia

Oczywiście *bałagan* z poprzedniego slajdu, to entropia.
Dla zmiennej losowej X zwracającej dwie wartości, mamy:

$$H(X) = -(q \log_2(q) + (1-q) \log_2(1-q)),$$

gdzie $q = \frac{p}{p+n}$

Nauczone drzewo decyzyjne

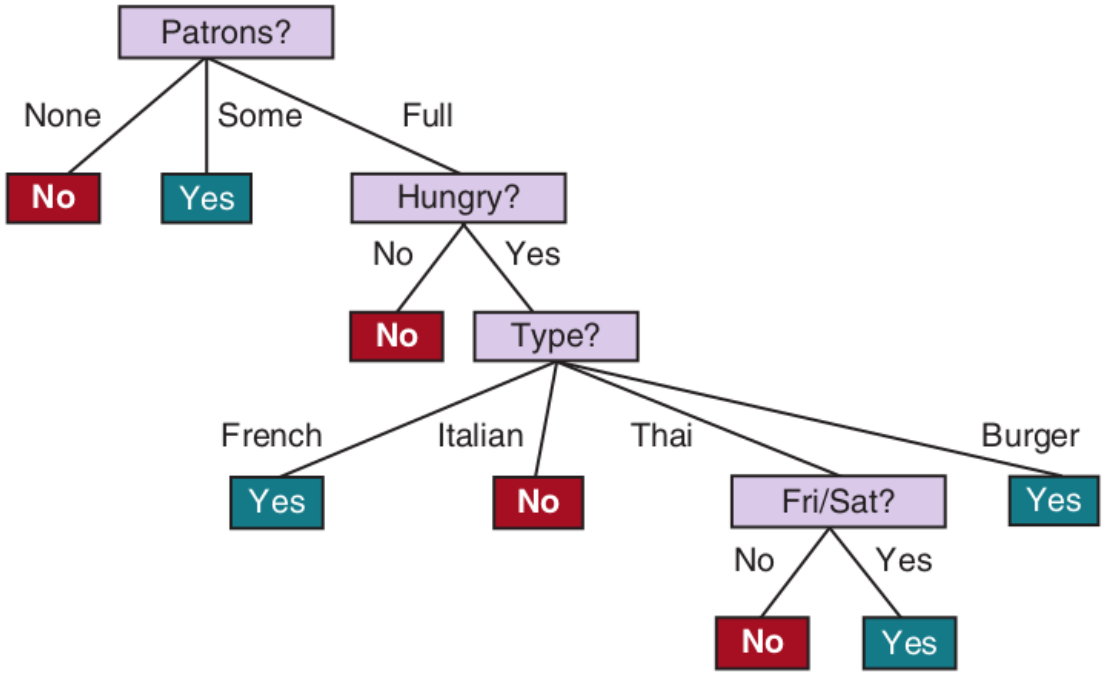


Figure 19.6 The decision tree induced from the 12-example training set.

Dlaczego mówimy o drzewach decyzyjnych?

Zła wiadomość

Obecnie drzewa decyzyjne w różnych tekstach łączą się z frazą słaby klasyfikator

Dobra wiadomość

Teksty, o których mowa, opisują nowoczesne metody, które wykorzystują wiele niezależnych *słabych klasyfikatorów* (w tej roli najczęściej są nasze drzewa)

Mądrość tłumu. Wół Galtona

Metody zespołowe

Łączą wiele *słabych* klasyfikatorów/regresorów

Bagging

- **uczenie**
  - Tworzymy K zbiorów uczących, każdy z nich składa się z N elementów (losowanych ze zwracaniem z oryginalnego zbioru uczącego o wielkości N).
  - Dla każdego zbioru trenujemy osobny klasyfikator
- **klasyfikacja / regresja**
  - Dla każdego elementu liczymy wyniki wszystkich klasyfikatorów.
  - Uśredniamy te wyniki (regresja) lub głosujemy na najlepszą klasę (klasyfikacja)

Metody zespołowe (2)

Lasy losowe. Przybliżenie pierwsze

Jak wyżej, ale oprócz tego losujemy zbiór atrybutów, o które pytamy (dla każdego drzewa inny). Albo tylko losujemy atrybuty

Lasy losowe (random forests)

Każde drzewo trenujemy na tych samych danych, ale przed wyborem atrybutu (za każdym razem jak rozwijamy węzeł), losujemy podzbiór dozwolonych atrybutów i z niego wybieramy.

Propozycja: $\sqrt{n}$ atrybutów dla klasyfikacji, $\frac{n}{3}$ atrybutów dla regresji (gdzie n jest liczbą dostępnych atrybutów)

Uczenie funkcji liniowej

- Klasyfikacji (czy regresji) możemy dokonywać na bazie wartości:

$$\sum_{i=0}^{N} w_i \phi_i(x)$$

- Cechami (dla MNIST-a) są po prostu wartości kolejnych pikseli
- Jedna funkcja „wystarcza” dla binarnego zadania typu: czy cyfra jest piątką? (i jak bardzo)
- Do rozwiązywania MNIST-a potrzebujemy 10 takich funkcji, wybieramy cyfrę dla tej funkcji, która zwraca największą wartość.

Funkcja liniowa i Reversi

- Możemy zdefiniować zadanie uczenia (regresji) dla Reversi:
  Widząc sytuację na planszy w ruchu 20 postaraj się przewidzieć zakończenie gry.
- Cechy: binarne cechy mówiące o zajętości pola.
- Przykładowo:
  Czy pole (4,5) jest czarne?
  Czy pole (1,1) jest białe?

Uwaga

Taki mechanizm byłby użyteczną funkcją heurystyczną, do użycia np. w algorytmie MiniMax.

Funkcja kosztu

Definicja

Funkcja kosztu (loss) opisuje, jak bardzo niezadowoleni jesteśmy z działania naszego mechanizmu (klasyfikatora, przewidywacza wartości).

Funkcja kosztu jest określona na: danych uczących (x,y) oraz wagach (parametrach klasyfikatora). Przy czym dane uczące traktujemy jako parametr, a wagi – jako właściwe argumenty.

Przykładowa funkcja kosztu: błąd średniokwadratowy

Funkcja kosztu (2)

Średniokwadratowa funkcja straty

$$\text{TrainLoss}(\mathrm{w}) = \frac{1}{|D_{\text{train}}|} \sum_{(x,y)\in D_{\text{train}}} (f_{\mathrm{w}}(x) - y)^2$$

Wariant liniowy:

$$f_{\mathrm{w}}(x) = \sum_{i=0}^{N} w_i \phi_i(x)$$

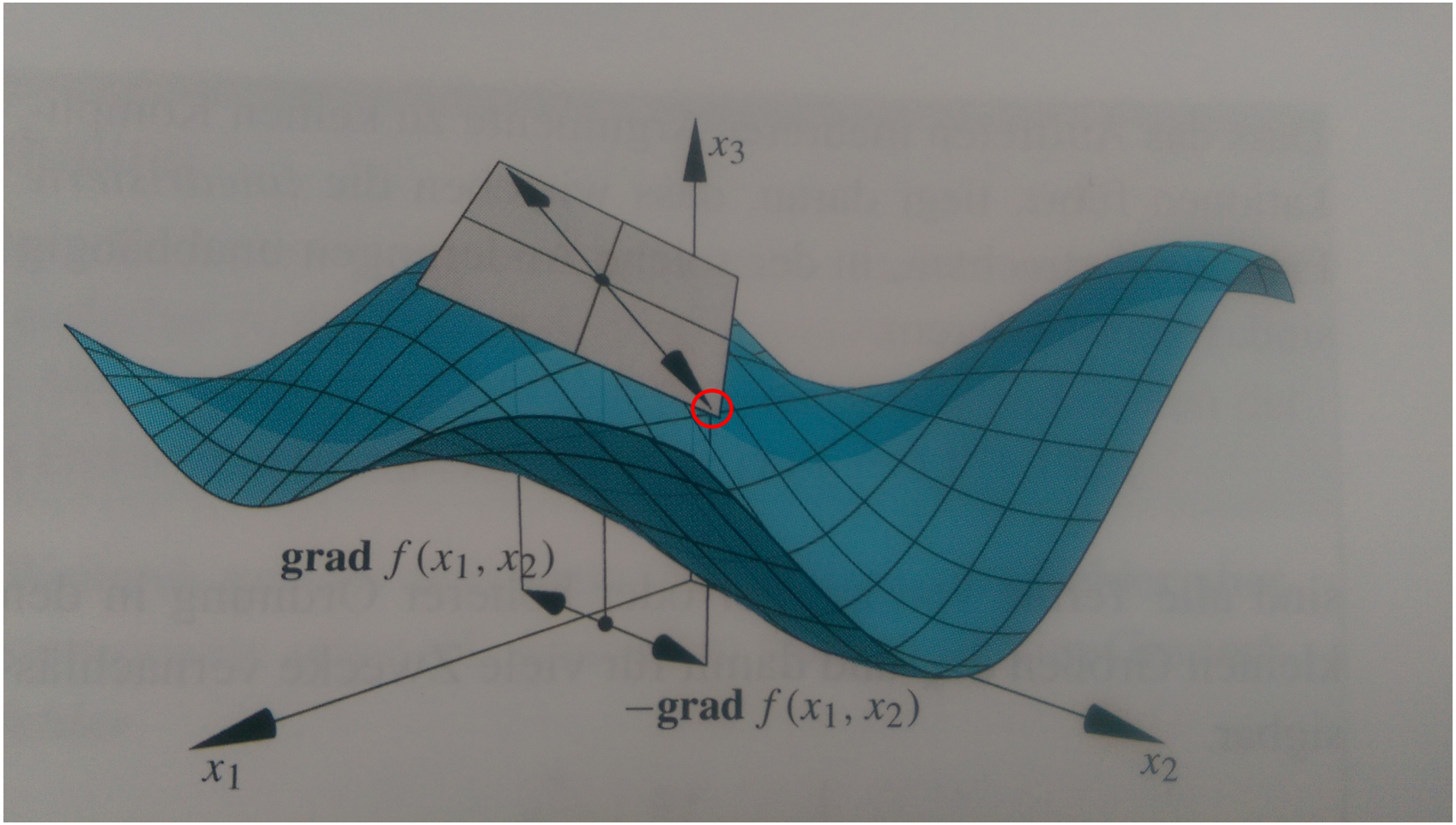
Gradient

- Gradient jest wektorem pochodnych cząstkowych.
- Wskazuje kierunek największego wzrostu funkcji.

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

Gradient

Wizualizacja gradientu w dla funkcji $\mathcal{R}^2 \rightarrow \mathcal{R}$



Jeszcze o gradiencie

Wzór

Pochodna po w_i (dla liniowej funkcji f_w

$$\frac{1}{|D_{\text{train}}|} \sum_{(x,y) \in D_{\text{train}}} 2 \cdot (f_w(x) - y) \cdot w_i \phi_i(x)$$

- Obliczenie gradientu wymaga przejścia przez cały zbiór uczący
- Maksymalizacja funkcji: dodawanie gradientu przemnożonego przez małą stałą (minimalizacja: odejmowanie)

Stochastic Gradient Descent

Obliczamy nie cały gradient, tylko jego składnik, związany z jednym egzemplarzem danych uczących i jego dodajemy (przemnożonego przez stałą)

Sieci neuronowe w paru prostych slajdach

- Wybierzemy absolutne minimum tego, co należy wiedzieć o sieciach neuronowych
- Oczywiście nie będzie to w pełni kompletna wiedza.

Slajd 1. Neuron

- Neuron to funkcja $f : \mathcal{R}^n \to \mathcal{R}$

$$f(x_1 \ldots x_n) = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$

- σ jest jakąś ustaloną funkcją nieliniową, raczej rosnącą, raczej różniczkowalną, na przykład: $\max(0, v)$, albo $\tanh(v)$
- Wygodna (jak za chwilę zobaczymy) jest notacja wektorowo-macierzowa, w niej mamy:

$$f(\mathrm{x}) = \sigma(\mathrm{w}^T \cdot \mathrm{x} + b)$$

Slajd 2. Prosta sieć neuronowa

- Warstwa to funkcja $\mathcal{R}^n \to \mathcal{R}^m$.
- Najbardziej typowa warstwa wyraża się wzorem:

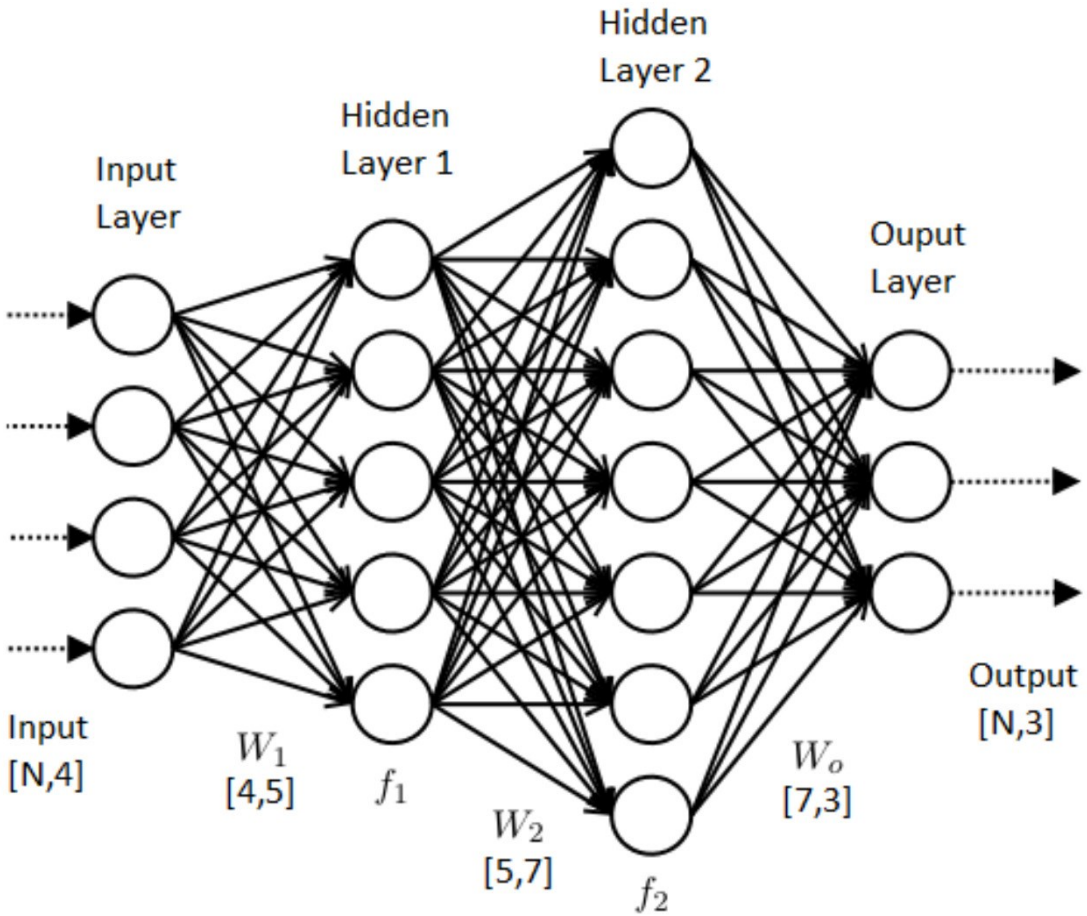
$$L(\mathrm{x}) = \sigma(\mathrm{W}\mathrm{x} + \mathrm{b})$$

- **Uwaga:** W jest macierzą wag (złożoną z wektorów wag), a $\sigma(y_1 \dots y_m) = (\sigma(y_1) \dots \sigma(y_m))$

Definicja

Sieć neuronowa typu MLP jest złożeniem warstw (z różnymi macierzami wag dla każdej warstwy).

Slajd 2b. Prosta sieć neuronowa



Źródło: VIASAT (https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4)

Slajd 3. Uczenie sieci

Zadanie

Danymi jest ciąg (x_i, y_i) opisujący porządane zachowanie sieci S oraz architektura tejże sieci (liczba warstw, ich wymiary, funkcja/funkcje σ).
Chcemy tak dobrać parametry (W_k oraz b_k) żeby dla każdego i

$$S(x_i) \approx y_i$$

Funkcja kosztu

Powyższe zadanie formalizujemy jako zadanie znalezienia takich parametrów, że koszt błędów jest jak najmniejszy. Przykładowo, jeżeli wyjściem jest liczba, to możemy wybrać:

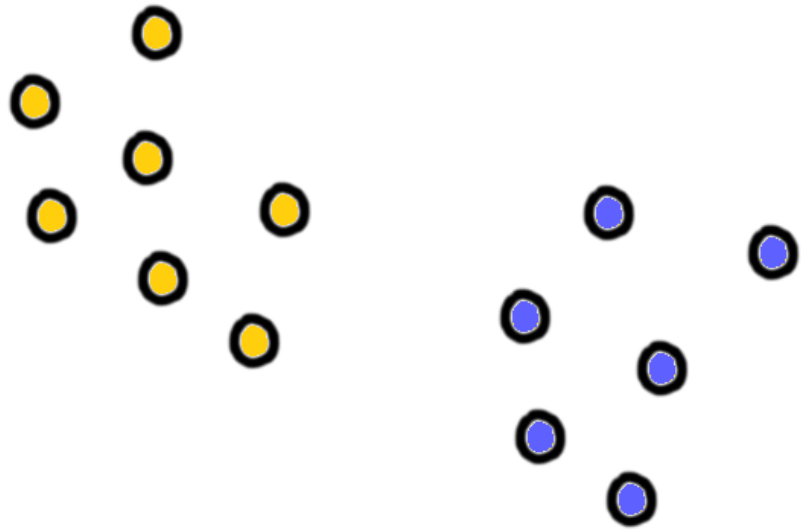
$$\text{Loss}(\theta) = \sum_{i}^{n} (S_\theta(x_i) - y_i)^2$$

Sieci neuronowe w praktyce. Klasyfikacja

Ogólne założenia (przypadek dwóch klas)

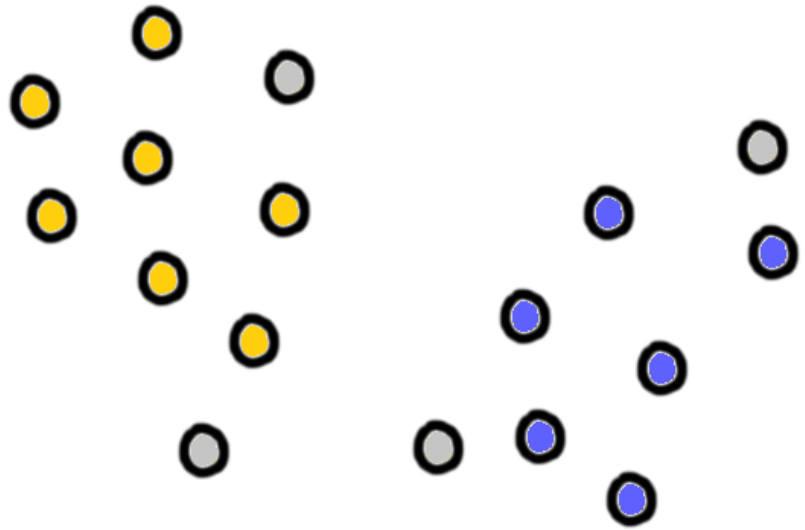
Mamy jakiś zbiór przykładów pozytywnych i negatywnych, interesuje nas mechanizm, który będzie poprawnie klasyfikował nieznane przykłady.

Klasyfikacja w $\mathcal{R}^2$



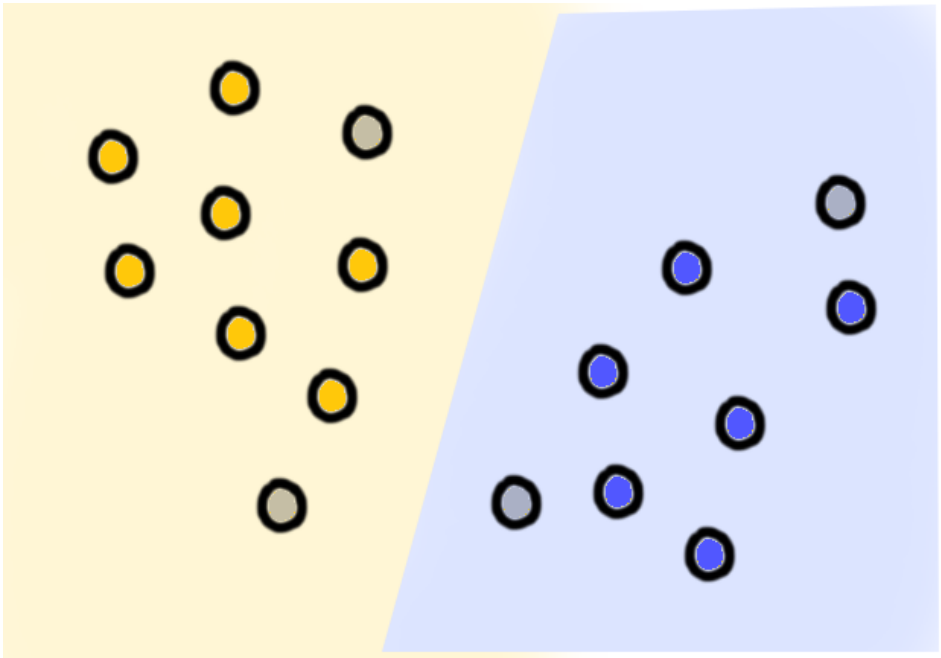
- Dane punkty wraz z informacją o kolorze.
- Mechanizm powinien umieć określać kolor nieznanych punktów.
- Możemy o tym myśleć, jako o „kolorowaniu płaszczyzny”

Klasyfikacja w $\mathcal{R}^2$



- Dane punkty wraz z informacją o kolorze.
- Mechanizm powinien umieć określać kolor nieznanych punktów.
- Możemy o tym myśleć, jako o „kolorowaniu płaszczyzny”

Klasyfikacja w $\mathcal{R}^2$



- Dane punkty wraz z informacją o kolorze.
- Mechanizm powinien umieć określać kolor nieznanych punktów.
- Możemy o tym myśleć, jako o „kolorowaniu płaszczyzny”

Spróbujmy poeksperymentować chwilę z Tensorflow Playground