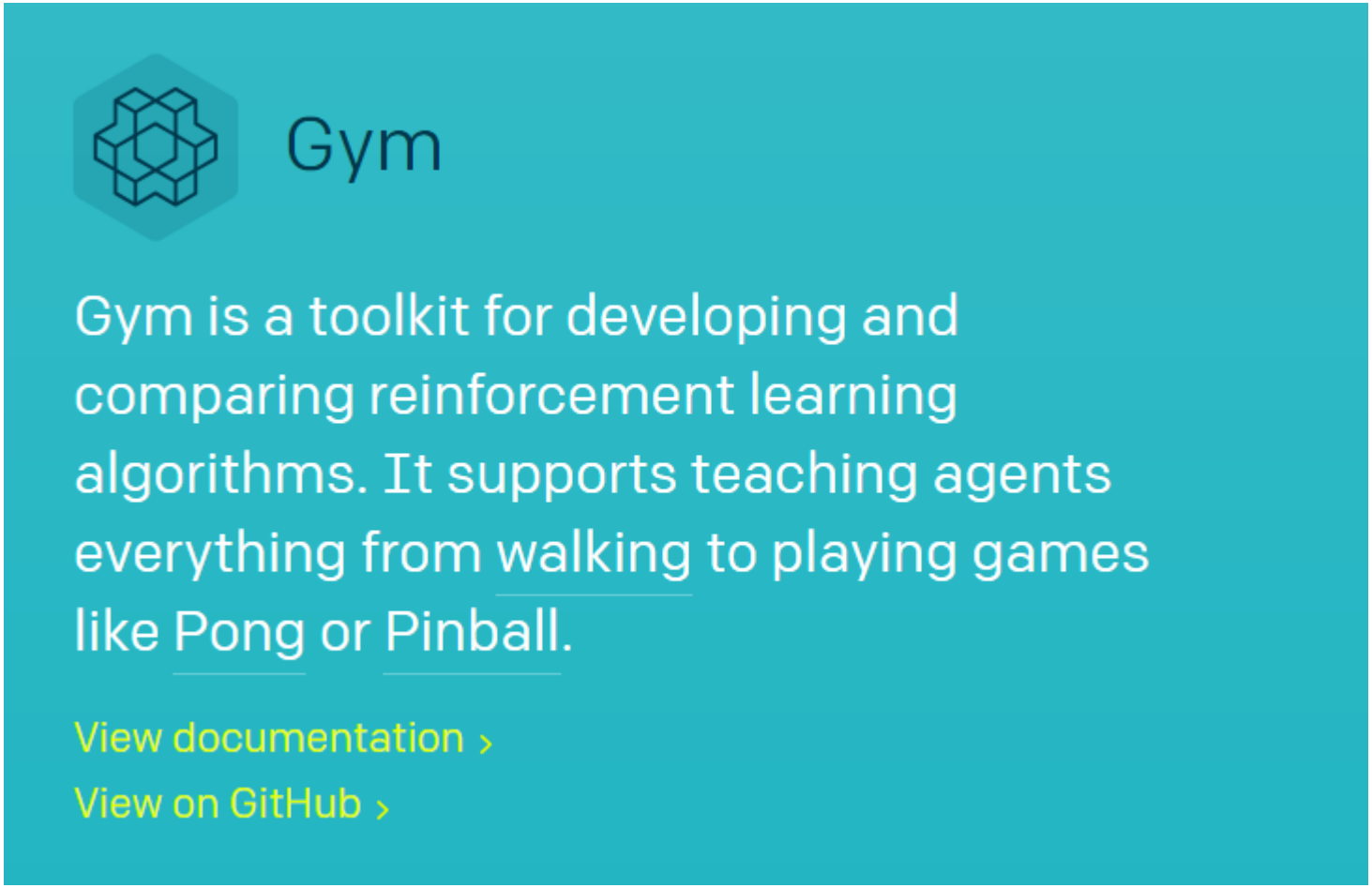
Wprowadzenie do uczenia ze wzmocnieniem

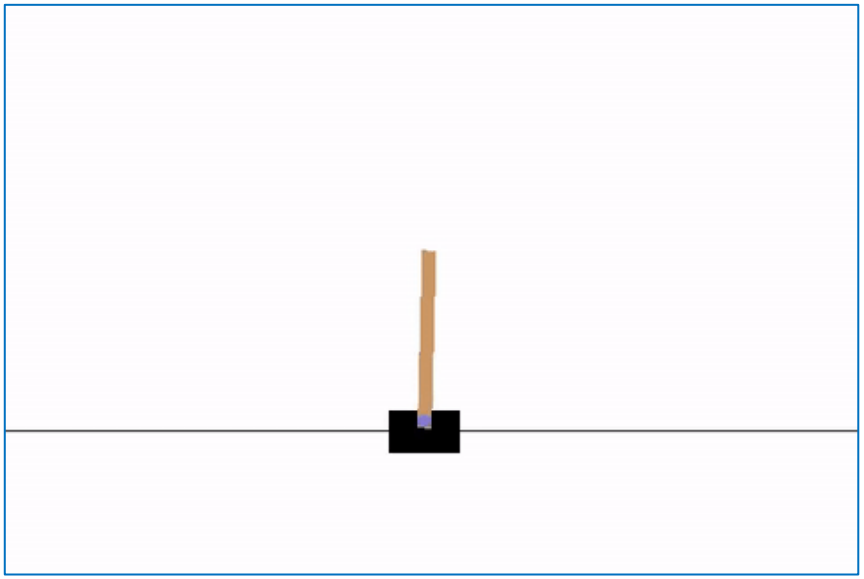
część 5

https://gym.openai.com



CartPole-v0

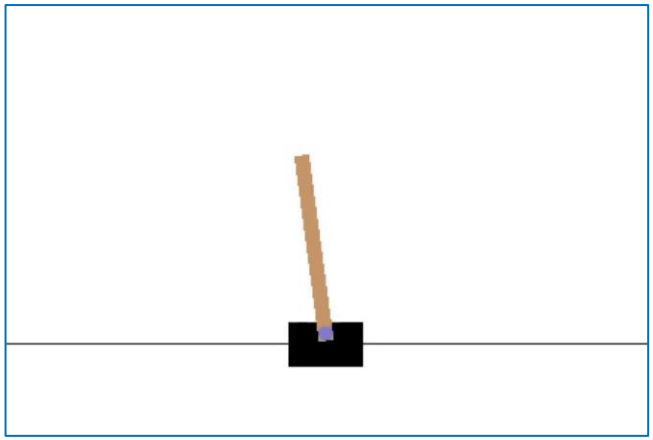
Ruchome wahadło jest przymocowane do wózka, który porusza się po torze bez tarcia.



Początkowo wahadło jest umieszczone pionowo, a celem jest zapobieżenie przewróceniu go przez zwiększenie i zmniejszenie prędkości wózka.

CartPole-v0

```
Actions:
    Type: Discrete(2)
    Num Action
    0   Push cart to the left
    1   Push cart to the right
```



```
Observation:
    Type: Box(4)
    Num Observation                Min         Max
    0   Cart Position             -4.8         4.8
    1   Cart Velocity             -Inf         Inf
    2   Pole Angle                 -24 deg     24 deg
    3   Pole Velocity At Tip      -Inf         Inf
```

Zbiór stanów jest nieskończony!!! Potrzebna jest dyskretyzacja.

CartPole-v0

```
Reward:
    Reward is 1 for every step taken, including the termination step

Starting State:
    All observations are assigned a uniform random value in
    [-0.05..0.05]

Episode Termination:
    Pole Angle is more than 12 degrees
    Cart Position is more than 2.4 (center of the cart reaches the edge
        of the display)
    Episode length is greater than 200
    Solved Requirements
    Considered solved when the average reward is greater than or equal
        to 195.0 over
    100 consecutive trials.
```

DQN

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

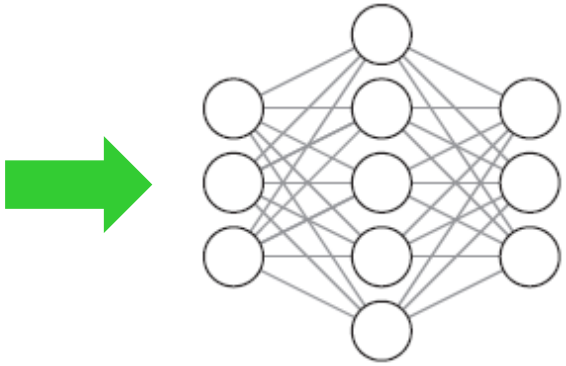
https://arxiv.org/abs/1312.5602

DQN – nasza wersja podstawowa

Sieć neuronowa aproksymuje nam Q:

S_t

```
[-0.01852746,
 -0.02263666,
 -0.0256044 ,
 -0.02643432]
```



A_t	Q
0	0.34
1	-0.11

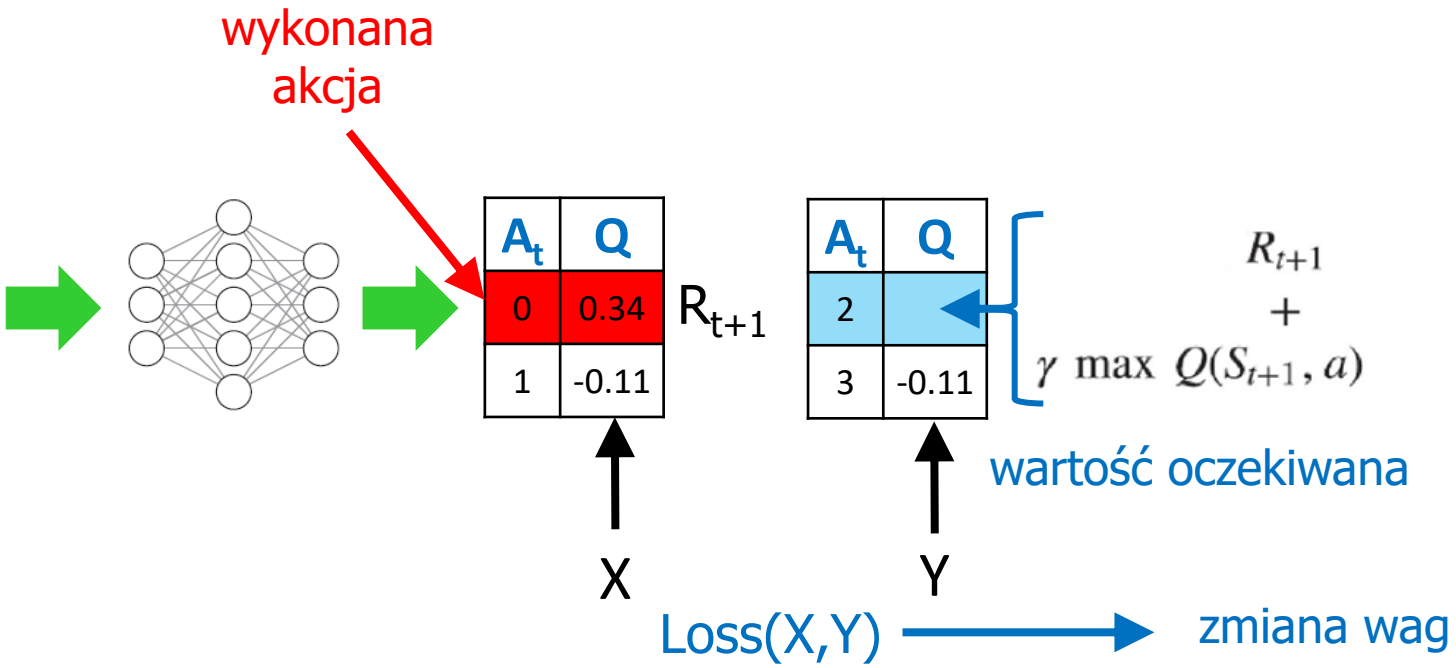
Sieć neuronowa zwraca szacowane wartości zwrotu dla każdej akcji (w środowisku CartPole akcje są dwie).

DQN – nasza wersja podstawowa

Uczenie:

S_t

```
[-0.01852746,
 -0.02263666,
 -0.0256044 ,
 -0.02643432]
```



Na wejście sieci neuronowej podajemy stan S_t dla którego sieć zwraca X, a wartością oczekiwaną jest Y.

Do modyfikacji parametrów sieci wykorzytujemy **jeden stan** S_t i jedną wartością oczekiwaną Y.

Sieci neuronowe

Regresja liniowa

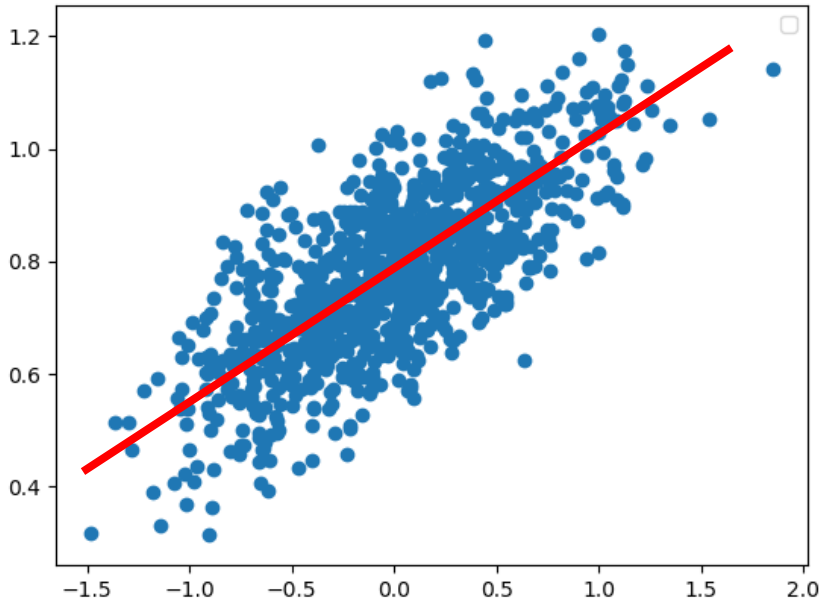
Regresja liniowa – metoda estymowania wartości oczekiwanej zmiennej **y** przy znanych wartościach innej zmiennej lub zmiennych **x**.

Zmienna **y** jest nazywana zmienną objaśnianą lub zależną. Inne zmienne **x** nazywane są zmiennymi objaśniającymi lub niezależnymi.

Zarówno zmienne objaśniane i objaśniające mogą być wielkościami skalarnymi lub tensorami.

Regresja liniowa

Regresja liniowa jest nazywana liniową, gdyż zakładanym modelem zależności między zmiennymi zależnymi a niezależnymi, jest funkcja liniowa bądź przekształcenie liniowe (afiniczne) reprezentowane przez macierz (tensor!) w przypadku wielowymiarowym.



Jaka prosta?

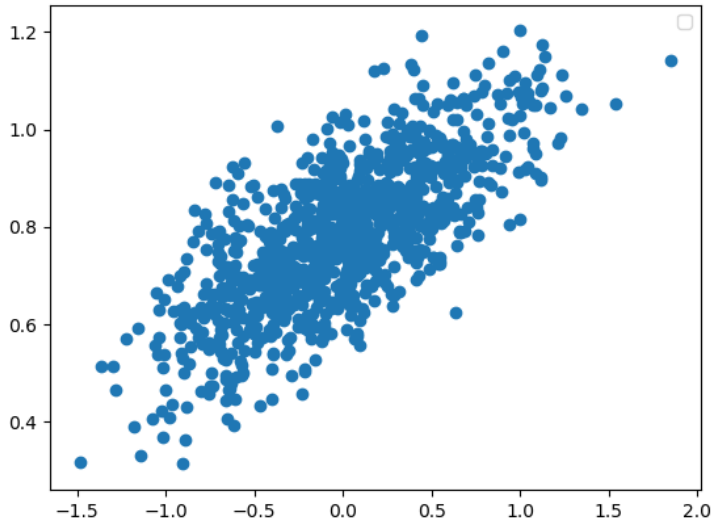
$y=ax+b$

$a = ?$ $b = ?$

Regresja liniowa

Dla danych $\{(x_1,y_1), \ldots, (x_N,y_N)\}$ zdefiniujmy błąd:

$$E(a,b) = \sum_{n=1}^{N} (y_n - (ax_n + b))^2$$

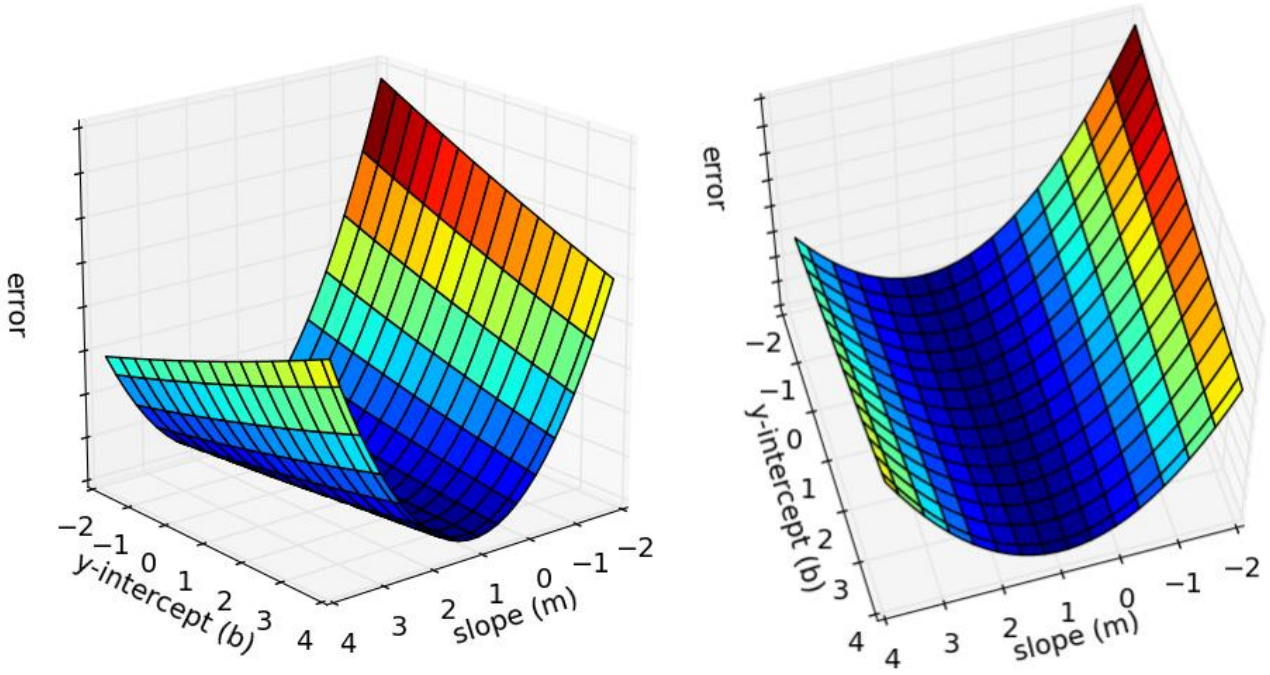


Nasz cel to znalezienie wartości ***a*** i ***b*** dla których błąd jest najmniejszy.

Regresja liniowa

$$E(a,b) = \sum_{n=1}^{N} (y_n - (ax_n + b))^2$$

Jak wygląda powierzchnia błędu *E(a,b)*?



Metoda najmniejszych kwadratów

Nasz cel to znalezienie wartości *a* i *b* dla których błąd jest najmniejszy.

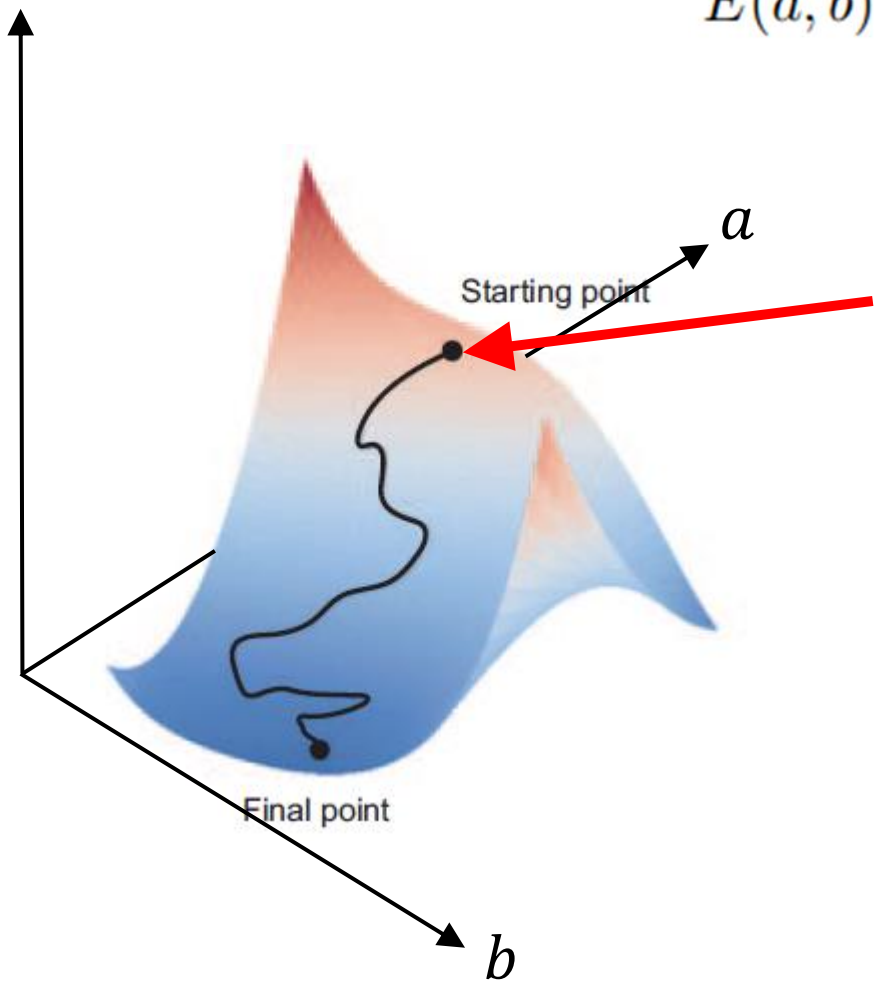
Interesuje nas minimum funkcji *E(a,b)*

W metodzie najmniejszych kwadratów szukamy wartości *(a, b)* takich, że:

$$\frac{\partial E}{\partial a} = 0, \quad \frac{\partial E}{\partial b} = 0.$$

Inne rozwiązanie?

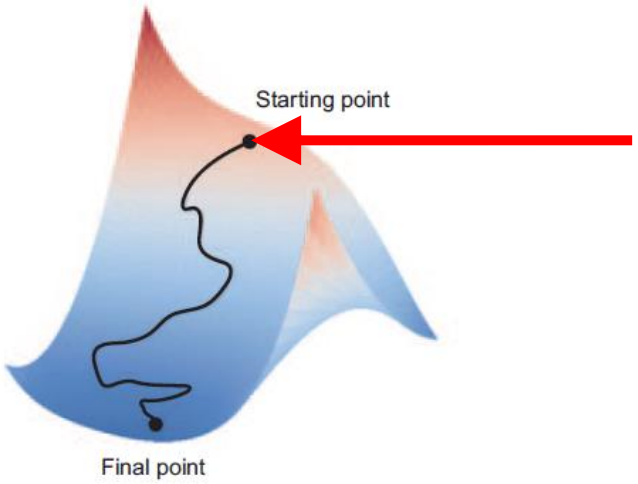
$$E(a,b) = \sum_{n=1}^{N} (y_n - (ax_n + b))^2$$

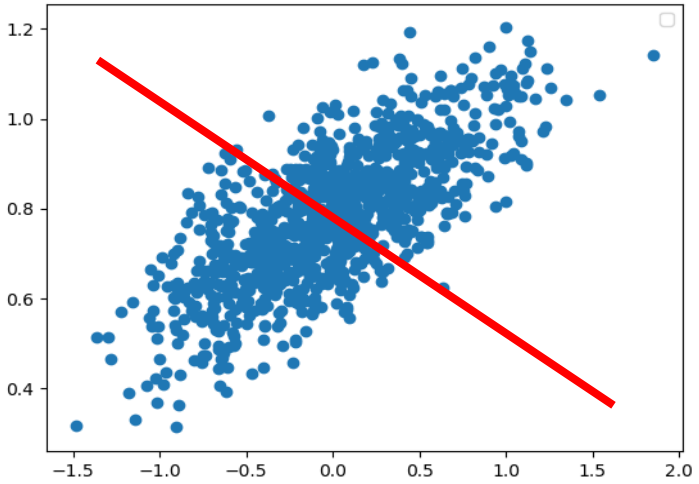


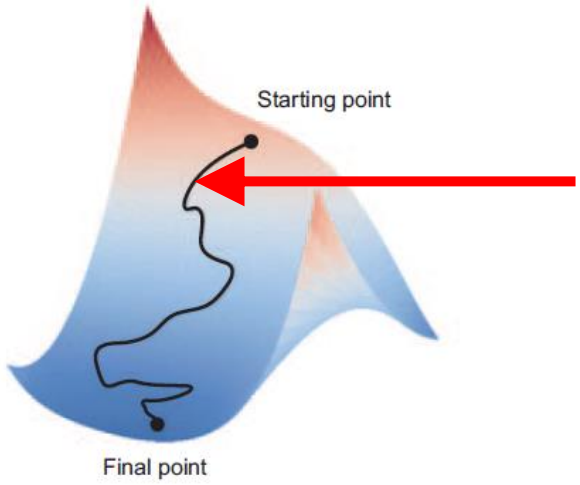
Wartość błędu $E(a,b)$ dla początkowych wartości parametrów a i b.

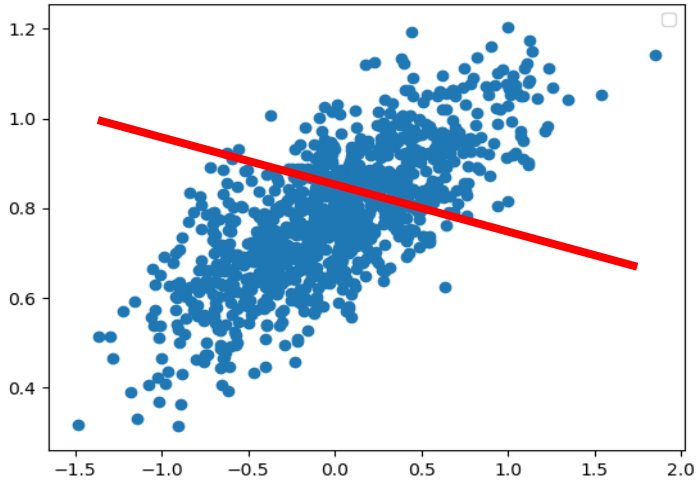
Czy możemy modyfikować parametry a i b w taki sposób, że wartość błędu będzie się przesuwała w kierunku minimum funkcji $E(a,b)$?

Inne rozwiązanie?

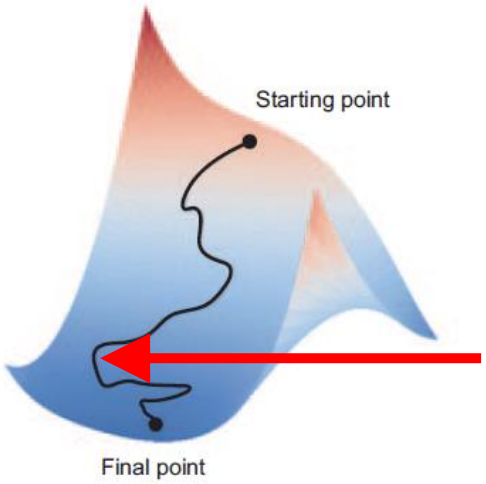


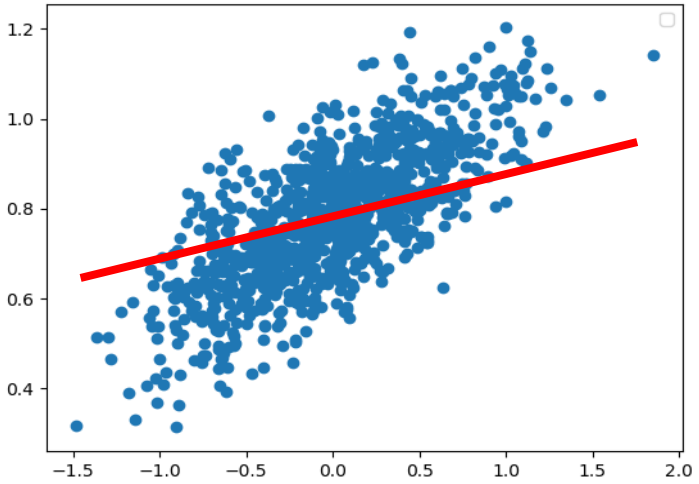


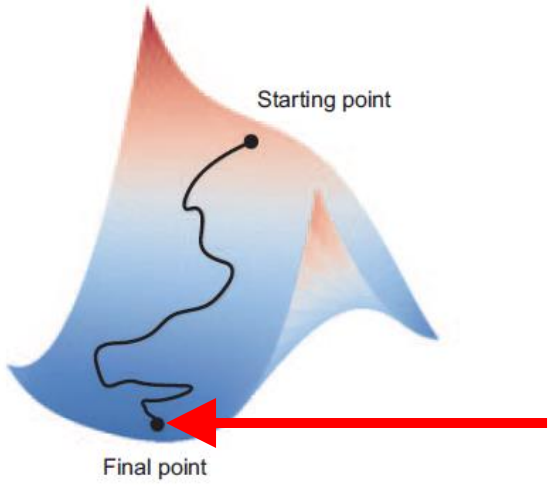


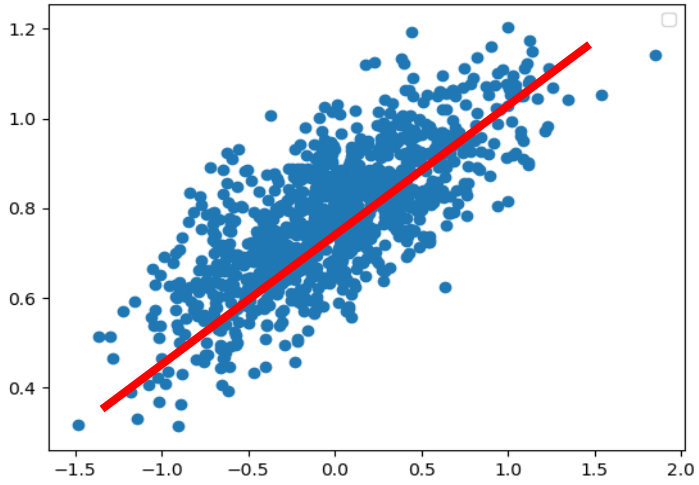


Inne rozwiązanie?







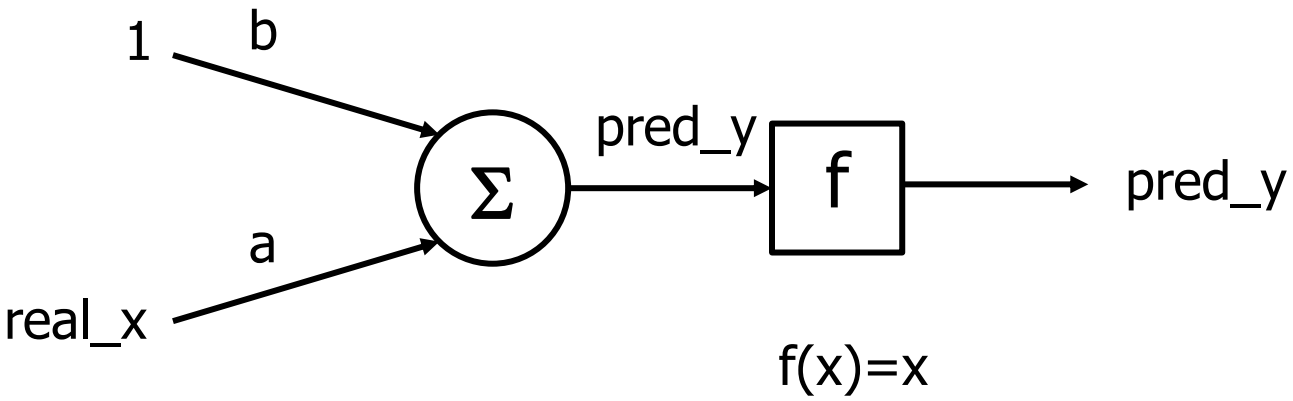


Sieć neuronowa?

Rozwiązujemy problem za pomocą jednego neuronu o jednym wejściu.

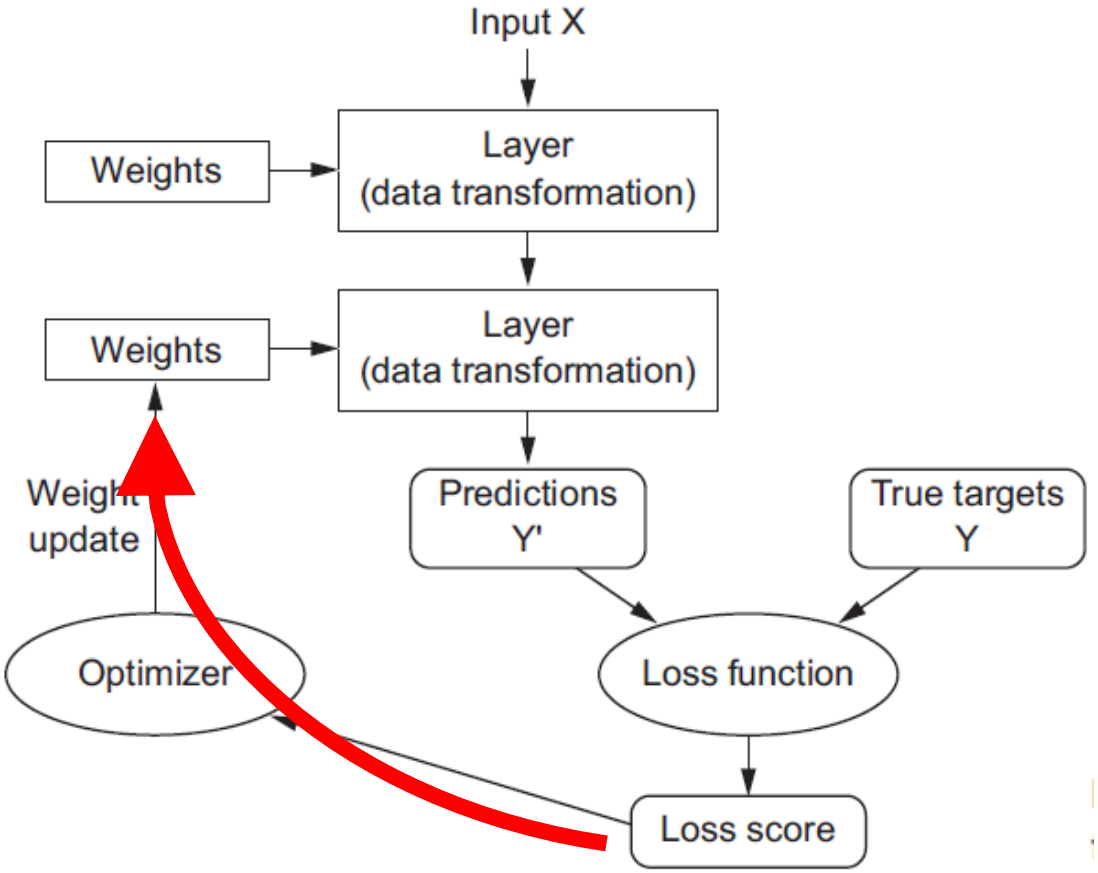
```
pred_y = a * real_x + b
```

Wagami neuronu są parametry a i b.



ML - idea

W oparciu o wartość policzonego błędu modyfikowane są wagi.



ML - idea

Uczenie odbywa się w następującej pętli treningowej:

1. Wybierz partię próbek x i odpowiednich celów y.
2. Podaj na wejście sieci x (krok nazywany *forward pass*), aby uzyskać prognozy y_{pred}.
3. Oblicz stratę sieci czyli błąd między y_{pred} i y.
4. Zaktualizuj wszystkie wagi sieci w sposób, który (nieznacznie) zmniejsza błąd.
5. Jeżeli to konieczne (błąd jest nadal duży) – wróć do punktu 1.

ML - idea

Zauważmy, że:

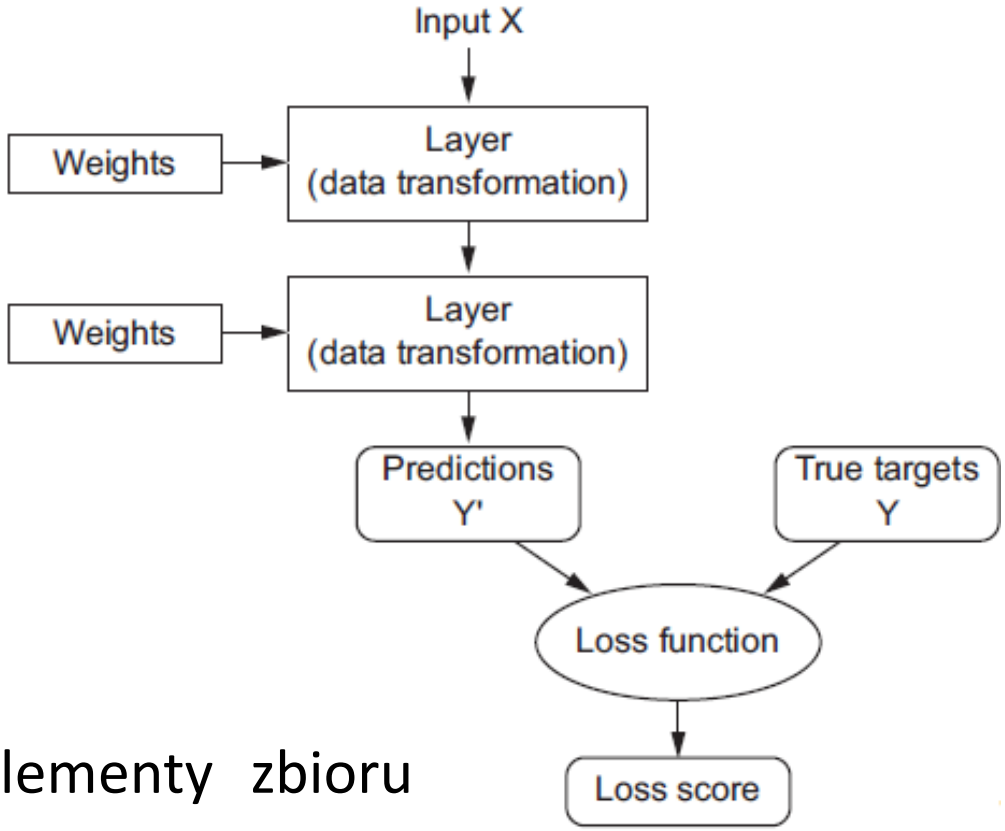
$y_{pred} = f_1(W,x)$

$loss = f_2(y_{pred}, y)$

W jest tensorem wag.

Czyli:

$loss = f_2(f_1(W,x), y)$



Ponieważ x i y jako elementy zbioru treningowego są stałe zatem:

loss = f(W)

$$E(a,b) = \sum_{n=1}^{N} (y_n - (ax_n + b))^2$$

Gradient funkcji

Przypomnijmy sobie zatem definicję gradientu funkcji:

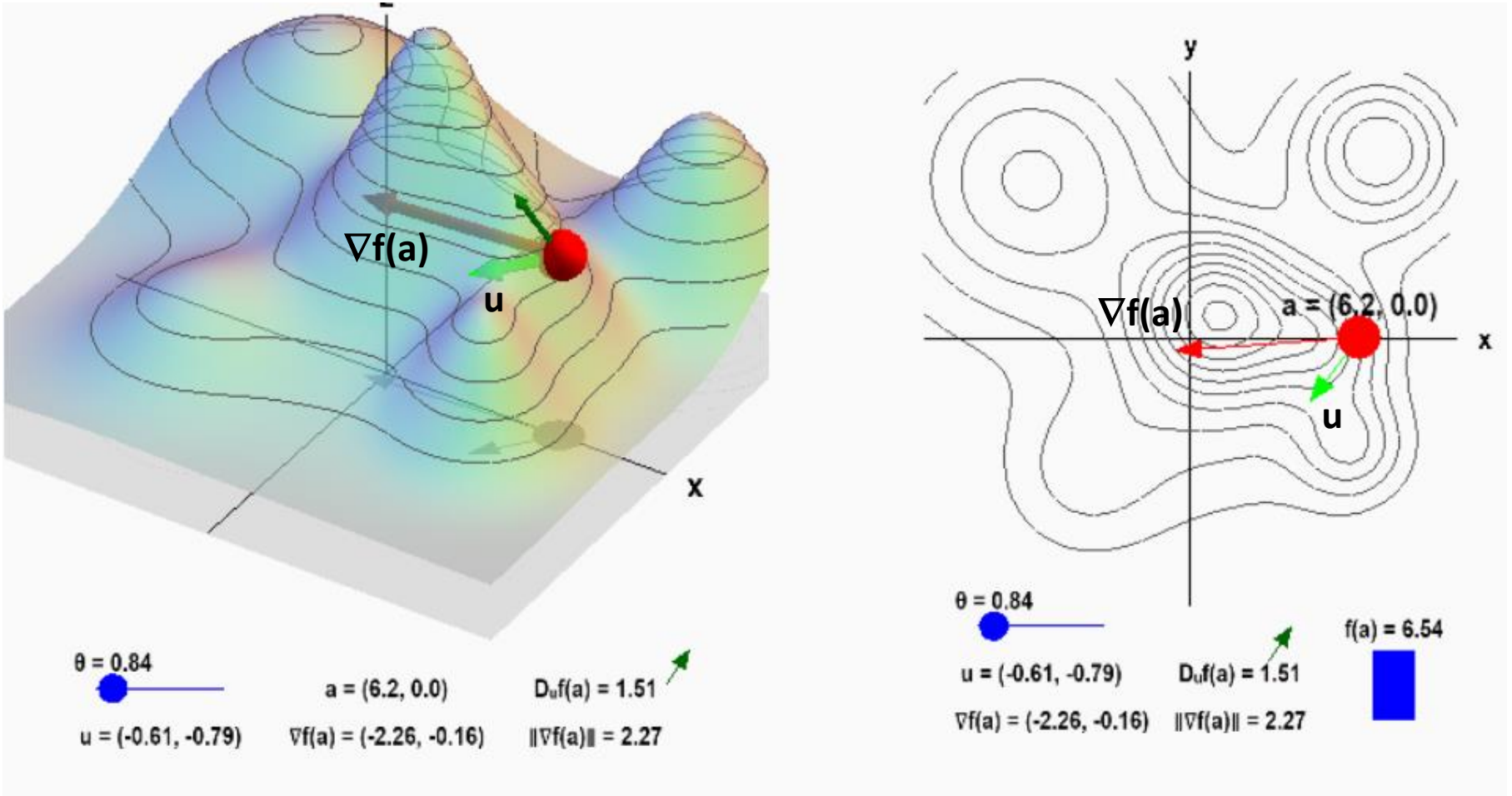
Gradient (lub gradientowe pole wektorowe) funkcji skalarnej $f(x_1, \ldots, x_n)$ oznaczany ∇f, gdzie ∇ (nabla) to wektorowy operator różniczkowy nazywany nabla. Innym oznaczeniem gradientu f jest $\operatorname{grad} f$.

W układzie współrzędnych kartezjańskich gradient jest wektorem, którego składowe są pochodnymi cząstkowymi funkcji f. Gradient definiuje się jako pewne pole wektorowe. W układzie współrzędnych kartezjańskich składowe gradientu funkcji f są pochodnymi cząstkowymi tej funkcji tzn.

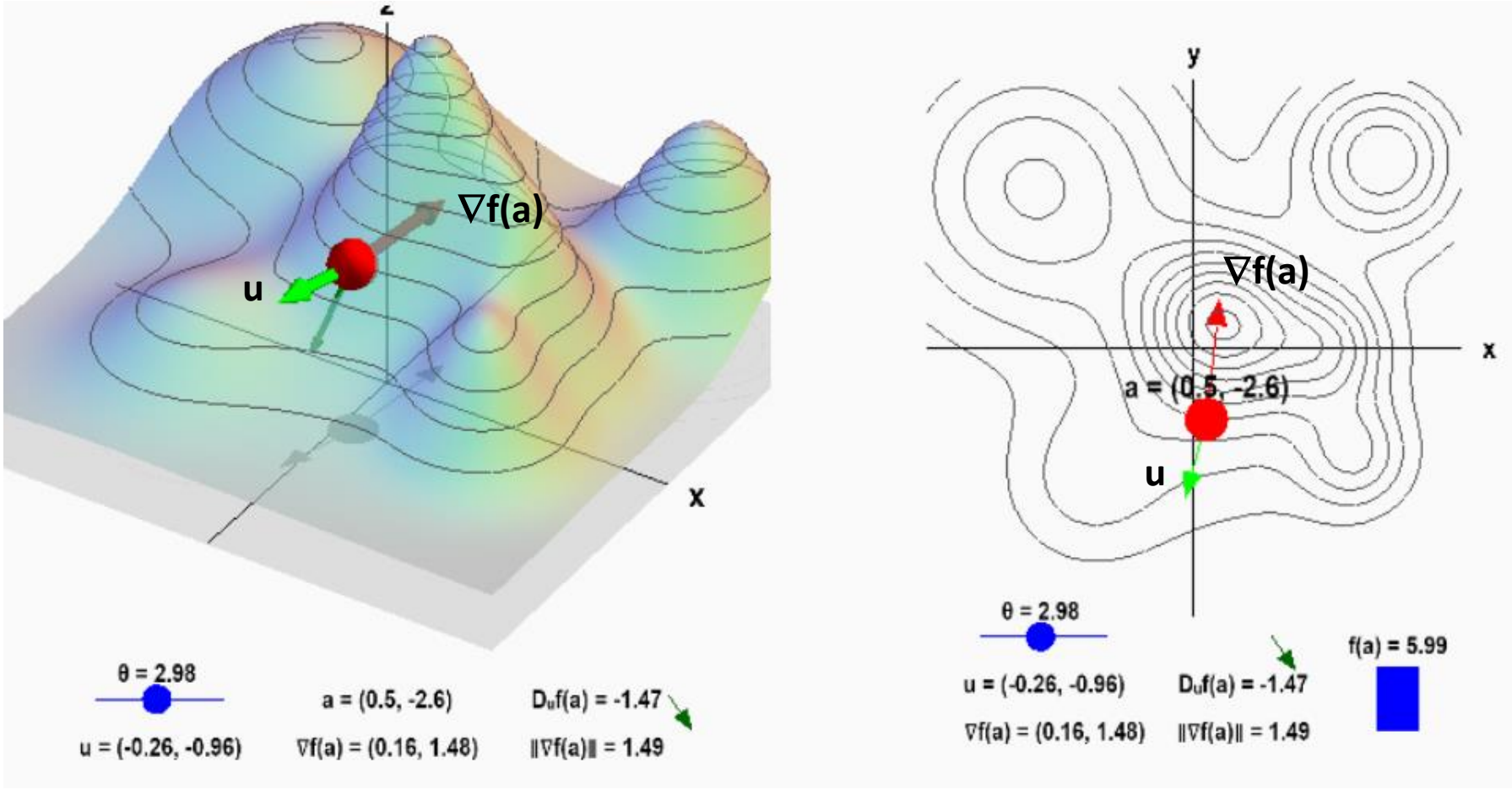
$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right].$$

Wektor gradientu pokazuje kierunek największego wzrostu funkcji w danym punkcie!

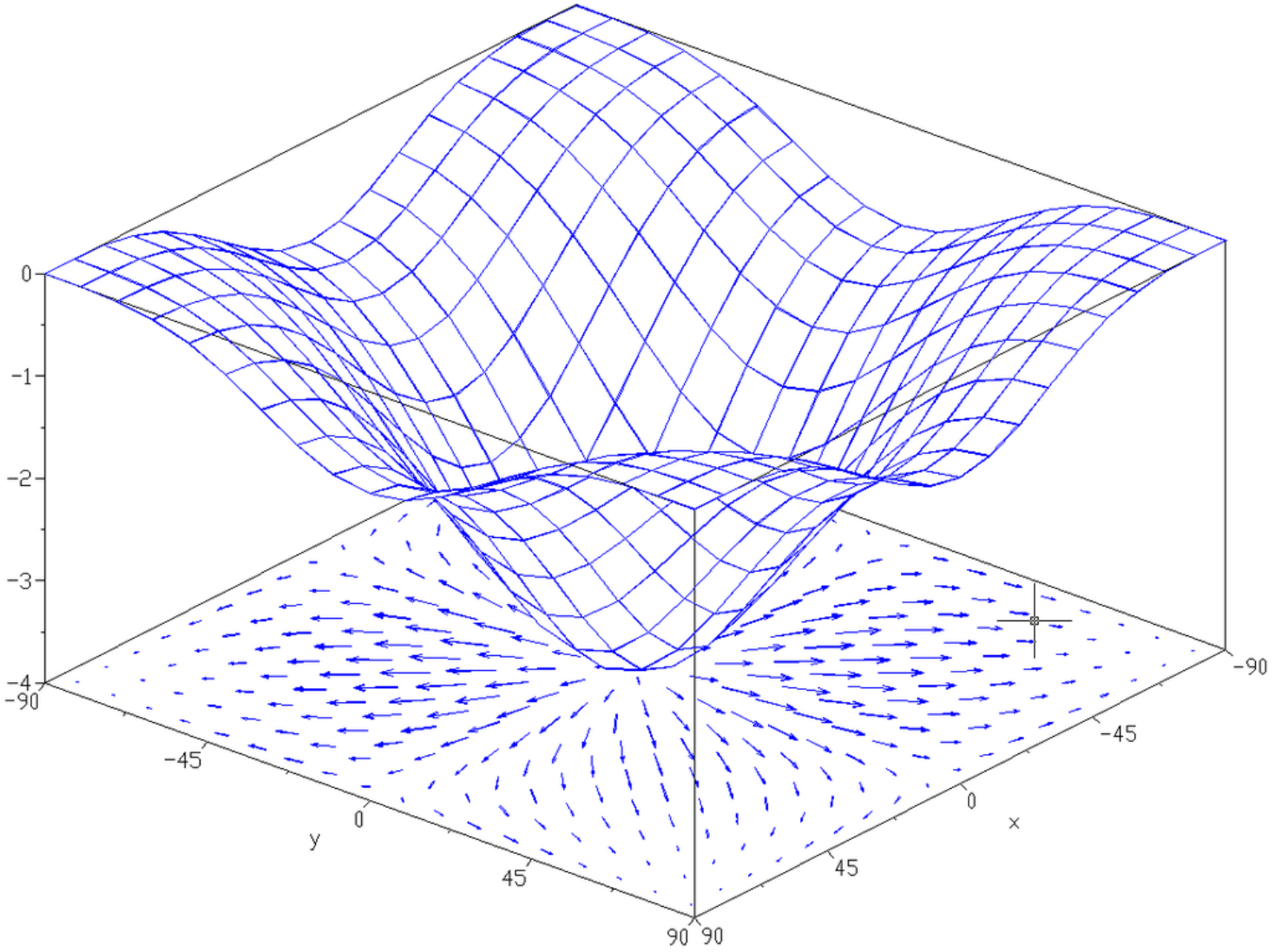
Gradient funkcji



Gradient funkcji



Gradient funkcji



Gradient funkcji

Przykład

Funkcja: $f(x, y) = x^2 y.$

Policzmy gradient: $\nabla f(3, 2)$

Z definicji: $\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right]$

Pochodne cząstkowe:

$$\frac{\partial f}{\partial x}(x, y) = 2xy \qquad \frac{\partial f}{\partial y}(x, y) = x^2$$

$$\frac{\partial f}{\partial x}(3, 2) = 12 \qquad \frac{\partial f}{\partial y}(3, 2) = 9$$

Optymalizacja gradientowa

Wróćmy do funkcji błędu:

$$\text{loss} = f(W) \qquad (*)$$

Funkcja ta jest zwykle funkcją bardzo wielu zmiennych (bo sieć ma bardzo dużo parametrów).

$$\text{loss} = f(w_1, w_2, \ldots, w_n)$$

UWAGA: 'W' w powyższej formule (*) to pewien tensor.

Optymalizacja gradientowa

Funkcji błędu:

$$\text{loss} = f(W)$$

Przyjmijmy, że aktualna wartość W wynosi W_0.

W_0 jest tensorem zawierającym parametry sieci.

UWAGA: Gradient funkcji f w punkcie W_0 czyli:

$$\nabla f(W_0)$$

ma ten sam kształt co W_0.

Optymalizacja gradientowa

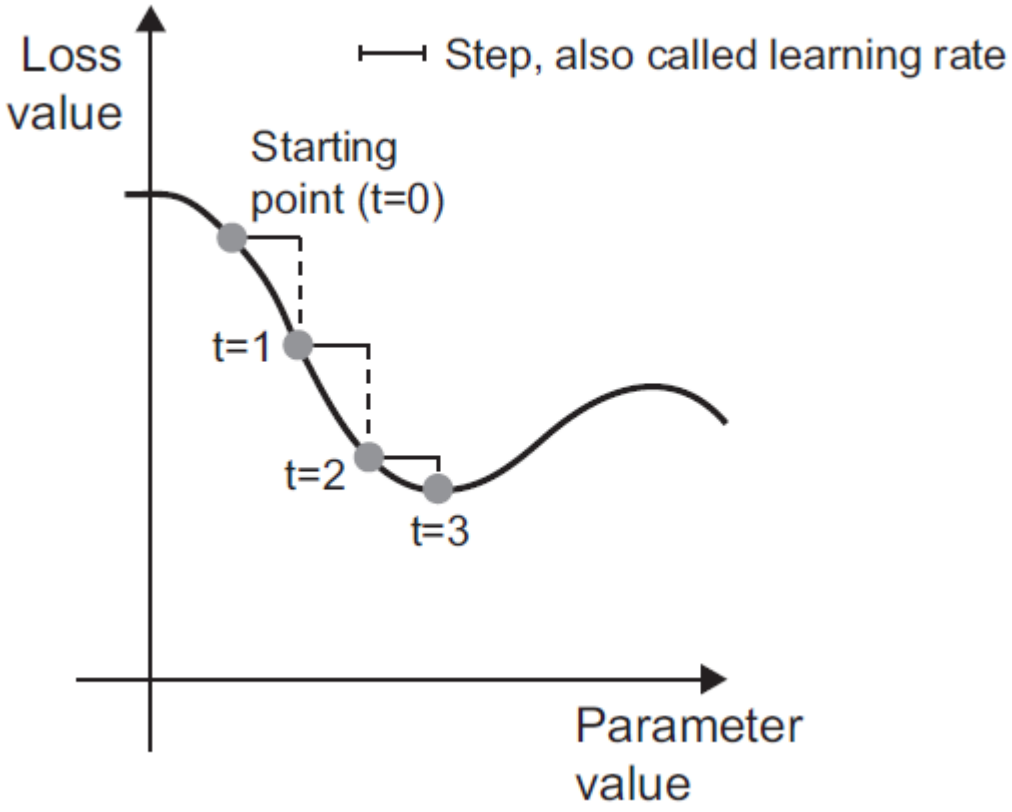
Ponieważ wektor gradientu pokazuje kierunek największego wzrostu funkcji w danym punkcie zatem wartość funkcji f(W) możemy zmniejszyć przesuwając się w kierunku przeciwnym do gradientu np:

$$W_1 = W_0 - \alpha \cdot \nabla f(W_0)$$

Przy czym α jest małym współczynnikiem uczenia. Jest on konieczny bo nie chcemy odejść zbyt mocno od W_0.

Optymalizacja gradientowa

W przypadku jednego parametru (wagi):



Optymalizacja gradientowa

Uczenie odbywa się zatem w następującej pętli treningowej:

1. Wybierz partię próbek x i odpowiednich celów y.
2. Podaj na wejście sieci x (krok nazywany *forward pass*)), aby uzyskać prognozy y_{pred}.
3. Oblicz stratę sieci czyli błąd między y_{pred} i y.
4. Zmodyfikuj wszystkie wagi sieci w sposób, który nieznacznie zmniejsza błąd.
5. Jeżeli to konieczne (błąd jest nadal duży) – wróć do punktu 1.

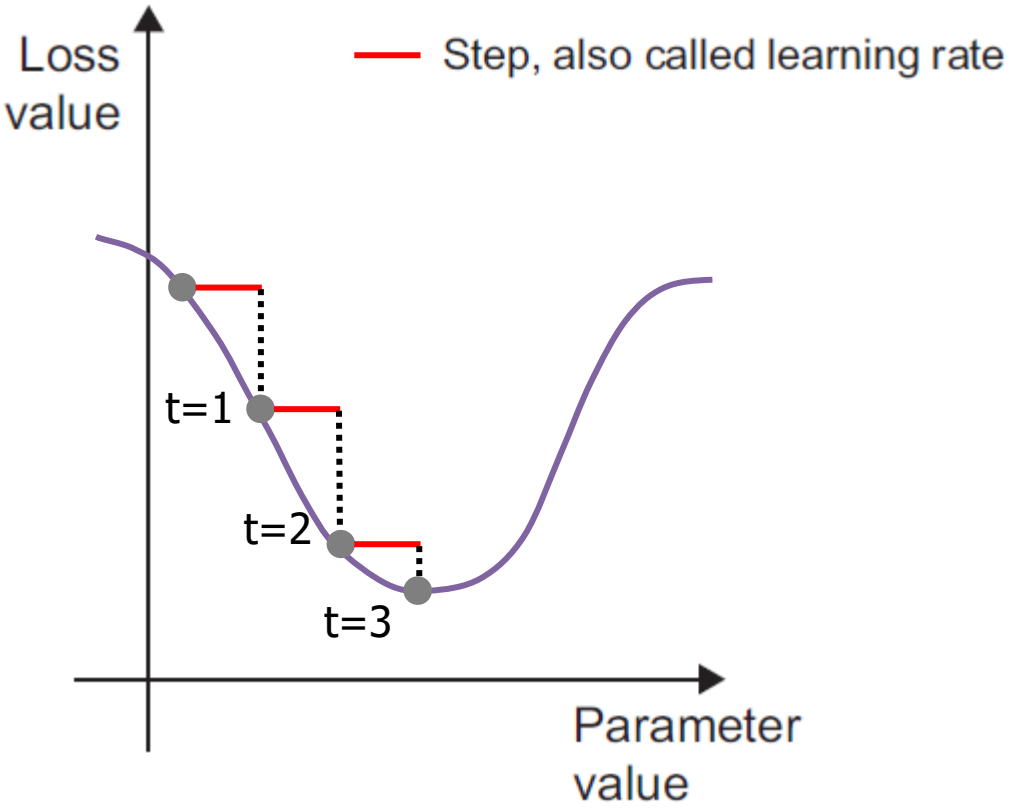
Optymalizacja gradientowa

Uczenie odbywa się zatem w następującej pętli treningowej:

1. Wybierz partię próbek x i odpowiednich celów y.
2. Podaj na wejście sieci x (krok nazywany *forward pass*)), aby uzyskać prognozy y_{pred}.
3. Oblicz stratę sieci czyli błąd między y_{pred} i y.
4. Oblicz gradient funkcji błędu f(W) i zmodyfikuj wszystkie wagi: $W_1 = W_0 - \alpha \cdot \nabla f(W_0)$
5. Jeżeli to konieczne (błąd jest nadal duży) – wróć do punktu 1.

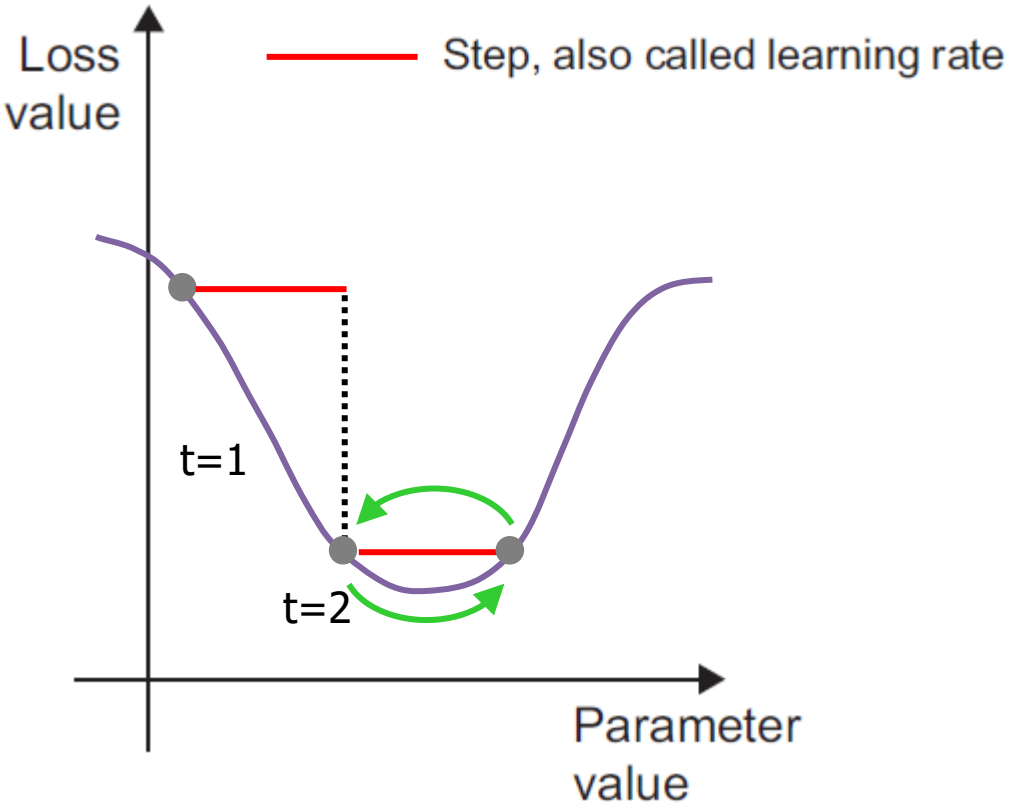
Optymalizacja gradientowa

W przypadku jednego parametru (wagi):



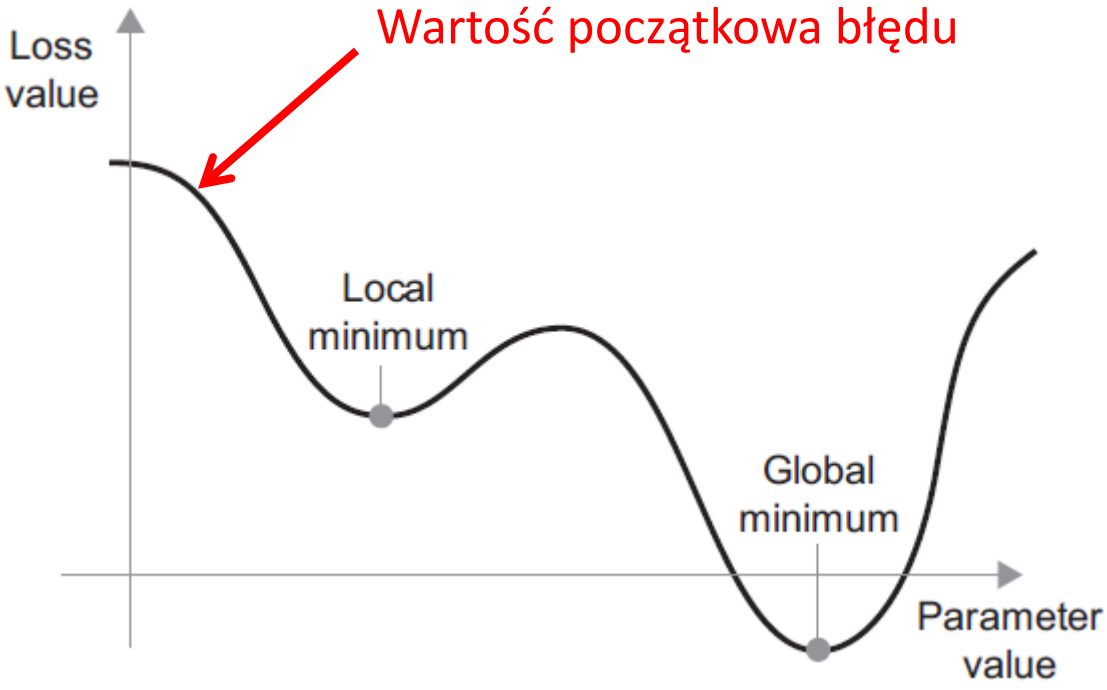
Optymalizacja gradientowa

W przypadku jednego parametru (wagi):



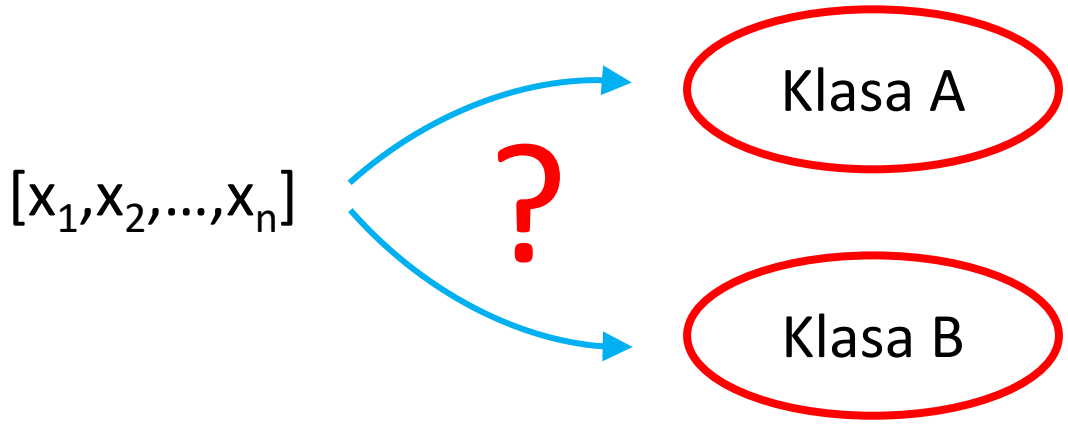
Optymalizacja gradientowa

A co w takiej sytuacji?



Regresja logistyczna

Rozważmy klasyfikator binarny

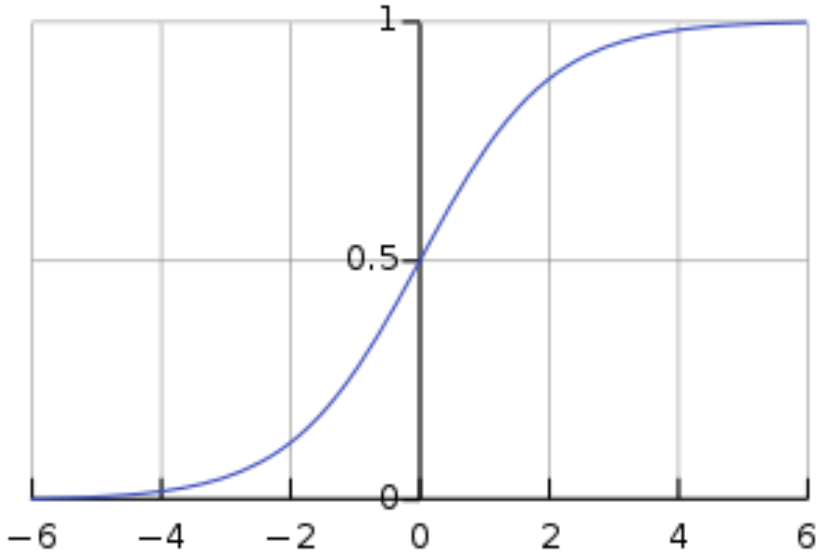


Przykład: Dowolne pytanie na które odpowiedź jest TAK/NIE
np. czy dany email jest spamem?

Regresja logistyczna

Funkcja logistyczna

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



Funkcja zwraca wartość z przedziału (0,1) czyli prawdopodobieństwo odpowiedzi TAK.

Regresja logistyczna

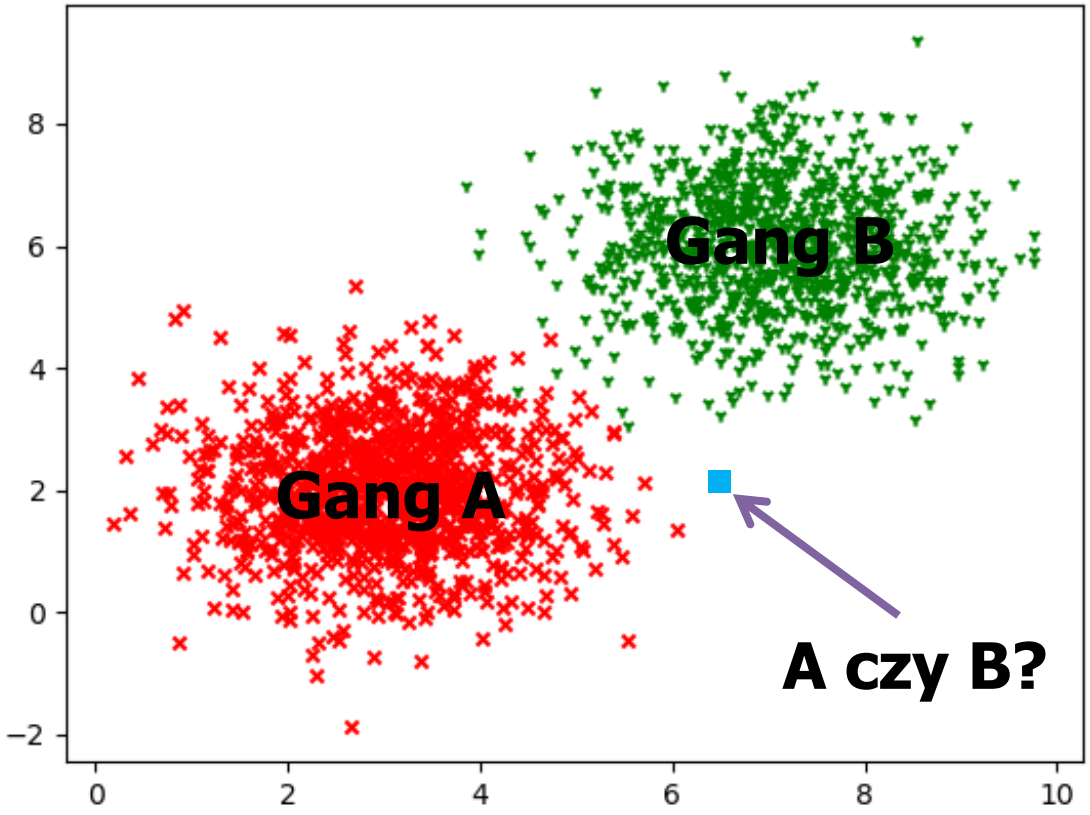
Do obliczenia błędu wykorzystujemy entropię krzyżową:

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

Rozkład prawdopodobieństwa y_i jest rozkładem oczekiwanym. Rozkład $\hat{y}_i$ jest zwracany przez model.

Regresja logistyczna

Rozważmy teraz zbiór danych:



Regresja logistyczna

```
pred_y = tf.sigmoid(w[2] * y + w[1] * x + w[0])
```

Neuron: