

# Inteligencja obliczeniowa - Sieci neuronowe

Grzegorz Madejski

# Kilka słów wstępu

- Sztuczne sieci neuronowe bazują na biologicznych odpowiednikach: neuronach i mózgach.
- Pierwsza praca na ten temat: Warren McCulloch, Walter Pitts (1943).

## **A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY**

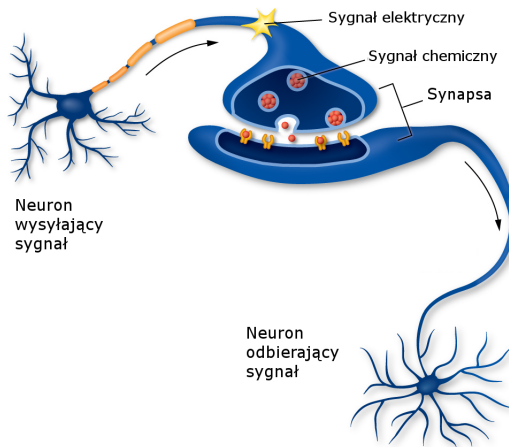
WARREN S. MCCULLOCH and WALTER H. PITTS

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

# Kilka słów wstępu

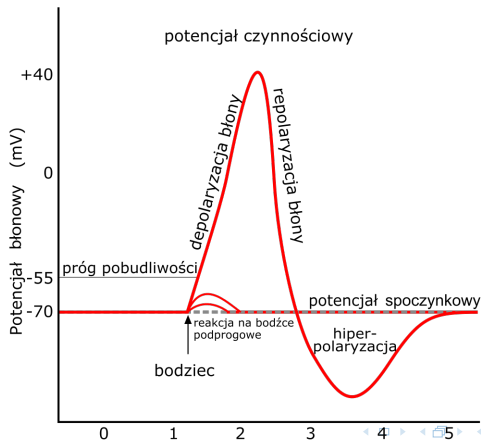
- Lata 1950: pierwsze kalkulatory przeliczające proste sieci neuronowe.
- Lata 1960: wielowarstwowe sieci neuronowe.
- Początek lat 1970: stagnacja w rozwoju sieci.
- 1975: Algorytm wstecznej propagacji (John Werbos).  
Ożywienie badań.
- Czasy współczesne: głębokie sieci neuronowe.

# Budowa neuronu



# Przesyłanie sygnału

- Neuron otrzymuje dostatecznie silny bodziec na dendrytach.
- Dochodzi do wytworzenia potencjału czynnościowego (polaryzacja błony neuronu)



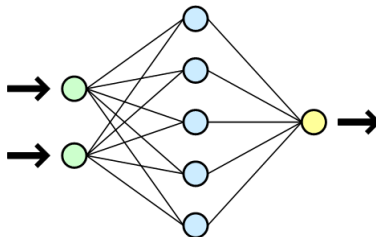
# Przesyłanie sygnału

- Neuron otrzymuje dostatecznie silny bodziec na dendrytach.
- Dochodzi do wytworzenia potencjału czynnościowego (polaryzacja błony neuronu)
- Potencjał czynnościowy jest stały dla neuronu (nie może wytwarzać silniejszych potencjałów).
- Impuls dociera do zakończeń aksonu.
- Sygnał elektryczny pobudza wydzielanie neuroprzekazników w synapsie. One wywołują sygnał elektryczny w kolejnym neuronie.

# Kilka filmów i zdjęć...

- "Neurons firing in hippocampus" Garret Stuber  
[<https://www.youtube.com/watch?v=j7aOwjGLOq0>]
- "Dendritic Imaging with GCaMP6s in Awake Monkey" Li et al. (2017)  
[<https://www.youtube.com/watch?v=Jtg83WKAi2g>]

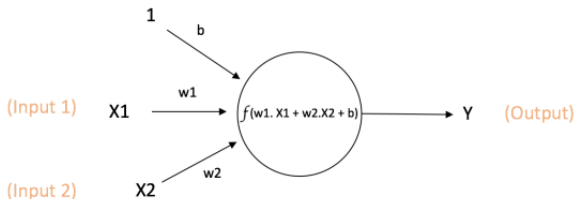
# Definicja: sieć neuronowa



- *Sztuczna sieć neuronowa*, w ujęciu matematyczno-informatycznym, to graf skierowany, z wagami na krawędziach i o specyficznej warstwowej strukturze.
- Warstwa wejściowa (Input layer) zawiera neurony, które pobierają sygnały/bodźce/dane z zewnątrz.
- Warstwy ukryte (hidden layers) przetwarzają te sygnały.
- Warstwa wyjściowa (output layer) zwraca dane wyjściowe/wyniki.



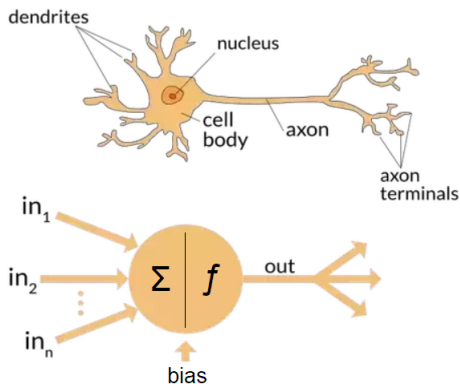
# Definicja: neuron



$$\text{Output of neuron} = Y = f(w1.X1 + w2.X2 + b)$$

- Neuron dostaje dane liczbowe (imitacja sygnałów elektrycznych), mnoży przez wagi na krawędziach i sumuje.
- Następnie suma przetwarzana jest przez tzw. funkcję aktywacji  $f(n)$
- Tak obliczona liczba jest przekazywana do kolejnej warstwy (propagowanie sygnału, forward propagation)



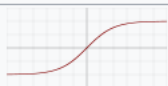
# Definicja: neuron



Czy widać podobieństwa między modelem biologicznym i matematycznym?

# Definicja: funkcja aktywacji

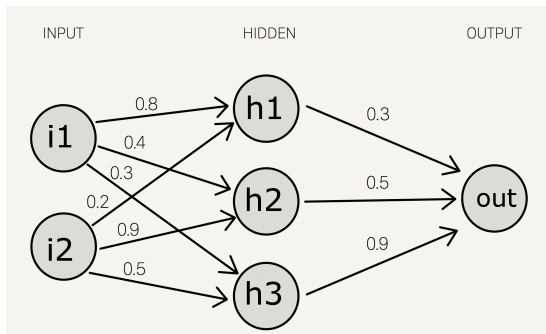
- Przypomnienie: potencjał czynnościowy neuronu biologicznego jest ograniczony przez pewien zakres.
- Funkcja aktywacji  $f(n)$  to "ogranicznik" dla modelu matematycznego. Jej wartości są z wąskiego zakresu, często  $[-1, 1]$  lub  $[0, 1]$ .
- Przykładowe funkcje:

funkcja progowa		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
funkcja logistyczna, sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$
tangens hiperboliczny		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$

# Przykład działania

Policz wyjście dla podanej sieci dla danych wejściowych  $(0, 0)$ ,  $(0, 1)$ ,  $(1, 0)$ ,  $(1, 1)$  oraz podanej funkcji aktywacji.

$$f(x) = \begin{cases} -1 & \text{dla } x < 1 \\ x & \text{dla } x \in [-1, 1] \\ 1 & \text{dla } x > 1 \end{cases}$$



# Definicja: schemat działania

Sieci neuronowe w praktyce same konstruuja potrzebne użytkownikowi modele, ponieważ automatycznie uczą się na podanych przez niego przykładach [źród: prof. E.Richter-Wąs].

- użytkownik sieci gromadzi reprezentatywne dane
- uruchamia algorytm uczenia, który ma na celu wytworzenie w pamięci sieci potrzebnej struktury (modelu)
- wyuczona sieć realizuje wszystkie potrzebne funkcje związane z eksploatacją wytworzonego modelu.

# Definicja: dane wejściowe i wyjściowe

- Przed włożeniem danych wejściowych do sieci warto je przygotować i obrobić np. normalizacją.
- Dane wyjściowe należy umieć poprawnie zinterpretować. Jeśli oczekujemy od sieci odpowiedzi tak/nie, to powinna ona zwracać liczby bliskie 0 i 1. Jeśli bliżej 1 to odpowiedź tak, jeśli 0 to nie.
- Bywa, że neurony warstwy wejściowej i wyjściowej nie są traktowane jako "pełnoprawne neurony" i nie stosuje się na nich funkcji aktywacji.

# Zalety sieci neuronowych

Zalety sieci neuronowych [źród: prof. E.Richter-Wąs].

- Potrafią odpowiadać w warunkach informacji niepełnej
- Nie wymagają znajomości algorytmu rozwiązania zadania (automatyczne uczenie)
- Przetwarzają informację w sposób wysoce równoległy
- Potrafią generalizować (uogólniać na przypadki nieznane)
- Są odporne na częściowe uszkodzenia
- Potrafią realizować pamięć asocjacyjną (skojarzeniową – podobnie jak działa pamięć u ludzi) w przeciwieństwie do pamięci adresowanej (charakterystycznej dla klasycznych komputerów)
- Duża moc (nieliniowych charakter)

# Różne typy sieci

Jest mnóstwo typów sieci neuronowych.

- Najprostszą sieć, z jednym neuronem, nazywamy *perceptronem*. Może klasyfikować zbiory liniowo separowalne. Dlaczego?
- Nasza definicja dot. klasycznej sieci jednokierunkowej (feedforward network). Istnieją też sieci rekurencyjne, w których sygnały mogą wracać do poprzednich warstw. Sieci modułarne, składające się z kilku mniejszych sieci.
- W obrębie każdego typu jest wiele podtypów.



# Czas na zabawę!

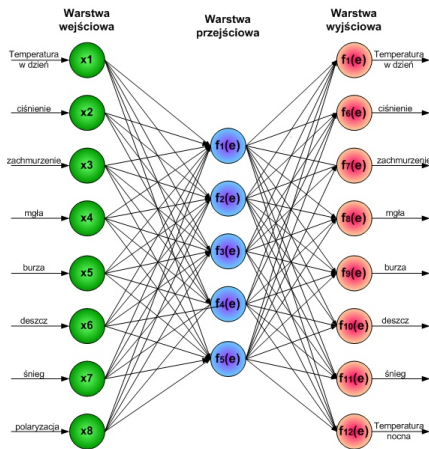
- Na stronie <https://playground.tensorflow.org/> znajduje się narzędzie do symulacji sieci neuronowych.
- Stwórzmy perceptron z wejściami  $X_1$  i  $X_2$  (podstawowe dane). Jak będzie klasyfikował różne datasety? Czy radzi sobie ze zbiorami, które nie są liniowo separowalne?
- Jak zmieni się wynik, gdy dodamy drugi neuron ukryty?
- Rozbuduj sieć i sprawdź czy uda ci się dobrze klasyfikować każdy z datasetów.

# Zastosowania

- Rozpoznawanie twarzy, obiektów (rozpoznawanie wzorców).
- Dane medyczne: stawianie diagnozy.
- Prognozowanie: giełda, sprzedaż.
- Aproksymacje funkcji.
- Rozpoznawanie pisma (OCR), mowy, gestów.
- Gry i rozwiązywanie problemów (szachy).

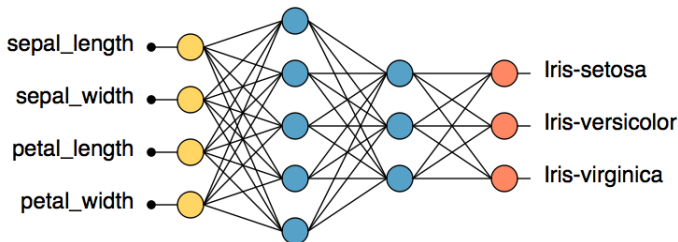
# Zastosowania: System neuronowy prognozowania pogody

System neuronowy prognozowania pogody (AGH, 2005). Na podstawie danych pogodowych przewidujemy dane na kolejny dzień.



# Zastosowania: Przewidywanie gatunku irysa

Sieć przetwarza parametry płatków irysów. Warstwa wyjściowa ma trzy neurony. Każdy zwraca liczbę, którą należy przybliżyć do 0 (nie ten gatunek) lub 1 (to ten gatunek).

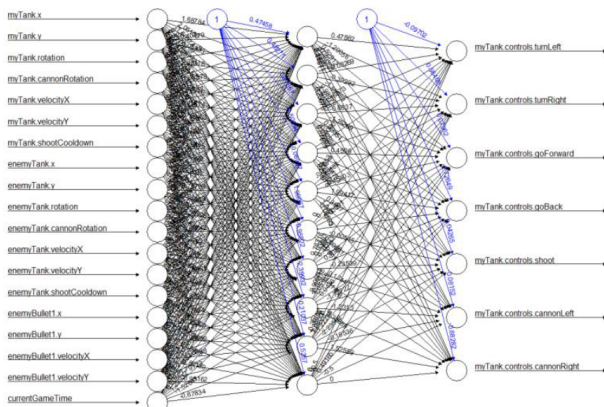


Jaki irys otrzymamy dla wyjścia (`Iris-setosa`=0.92, `Iris-versicolor`=0.76, `Iris-virginica`=0.14)?

Sieć radzi sobie równie dobrze jak kNN i Naive Bayes.

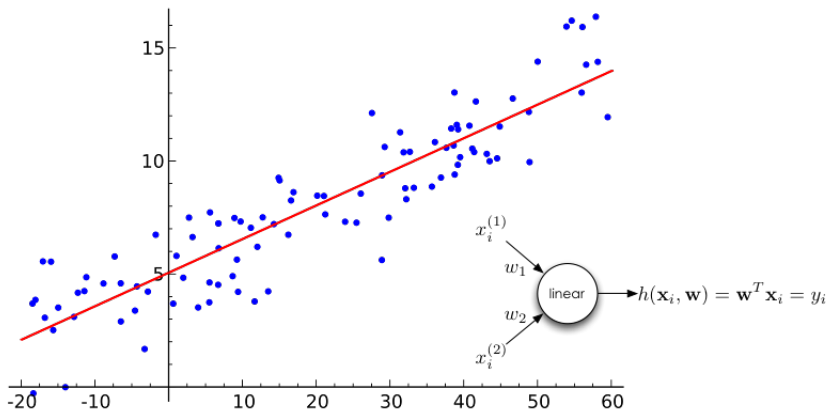
# Zastosowania: Sterowanie czołgiem

Konkurs wśród studentów UG (styczeń 2019) na czołg z najlepiej sterującą nim siecią neuronową. Niestety problem złożony, sterowanie wyszło bardzo marne (film).



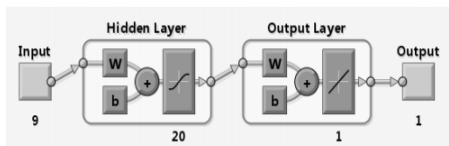
# Zastosowania: Regresja liniowa

Sieci neuronowe (nawet proste) mogą dokonać regresji liniowej dla danych tzn. narysować linię wyznaczającą trend w danych.



# Zastosowania: Przewidywanie bankructwa

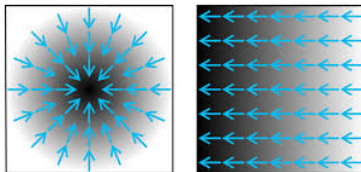
Badanie banków hiszpańskich w obliczu kryzysu 1977-1985. Sieć poradziła sobie w 92% przypadków [“Neural Networks in Bank Insolvency Prediction”, Qeethara Al-Shayea et al.]



Spanish banks data	
S. No.	Predictor Variable Name
1	Current assets/total assets
2	Current assets-cash/total assets
3	Current assets/loans
4	Reserves/loans
5	Net income/total assets
6	Net income/total equity capital
7	Net income/loans
8	Cost of sales/sales
9	Cash flow/loans

# Gradient

*Gradient* to pole wektorowe dla funkcji liczbowej. W każdym punkcie dziedziny funkcji zaczepiamy strzałkę wskazującą, w którą stronę funkcja rośnie/maleje i jak szybko się to dzieje. Matematycznie można go rozumieć jako zestaw pochodnych cząstkowych.



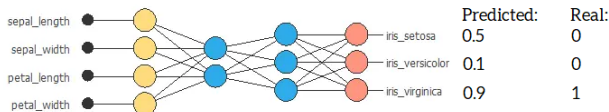
Przykład: gradientem dla funkcji  $f(x, y, z) = 2x - 3y^2 - \sin z$  jest:

$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right] = [2, -6y, -\cos z]$$



# Funkcja straty

W przypadku sieci, funkcja  $f$ , którą chcemy zminimalizować (patrzac na jej gradient), to *funkcja błędu / straty* (ang. *loss function*). Zwykle wybiera się funkcję *MSE* (Mean Squared Error) lub *Cross Entropy*.



Policzmy MSE dla przykładu powyżej:

$$\begin{aligned}
 MSE &= \frac{1}{3} \cdot ((0 - 0.5)^2 + (0 - 0.1)^2 + (1 - 0.9)^2) = \frac{1}{3} \cdot (0.25 + 0.01 + 0.01) = \\
 &= \frac{1}{3} \cdot 0.27 = 0.09
 \end{aligned}$$

# Funkcja straty

Funkcja MSE porównuje przewidywany output z prawdziwym outputem dla jednej próbki (te dwa wektory są jej argumentami).

$$MSE(output_{pred}, output_{real}) = \sum_i (output_{pred}^i - output_{real}^i)^2 / n$$

W trenowaniu sieci poprawiamy jednak wagi  $W$  w sieci. Należy zdefiniować więc funkcję straty następująco, z innym argumentem:

$$Loss_{input}(W) = MSE(ForwardProp_W(input), output_{real})$$

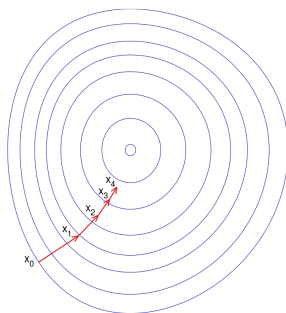
dla pewnej próbki  $input$ .

W praktyce bierzemy też często większy zestaw próbek (tzw. batch, ale o tym później) i obliczamy średnią funkcję:

$$Loss_{batch}(W) = \sum_{input \in batch} MSE(ForwardProp_W(input), output_{real}) / |batch|$$

# Spadek (wzdłuż) gradientu

Jak szukać minimum dla funkcji straty? Trzeba poruszać się w kierunku minimum, patrząc na pochodne cząstkowe (czyli gradient funkcji). Służy do tego procedura zwana *spadkiem (wzdłuż) gradientu* (ang. *gradient descent*). Algorytm działa w krokach - iteracjach.



# Wsteczna propagacja

*Wsteczna propagacja* (ang. *backpropagation*) to sposób uczenia sieci, który idzie od warstwy wyjściowej, "w lewo", do warstwy wejściowej poprawiając wagi. Poprawianie każdej z wagi następuje poprzez spadek gradientu dla niej.

# Algorytm wstecznej propagacji

- Jest wiele wariacji algorytmów uczenia się sieci, ale najbardziej standardowym jest *algorytm wstecznej propagacji*.
- Algorytm nie zmienia struktury sieci (to musi dopasować człowiek).
- Algorytm zmienia wagi na połączeniach między neuronami, by sieć dawała najlepsze odpowiedzi.
- Algorytm jest iteracyjny, działa w wielu epokach, stale poprawiając efektywność sieci.

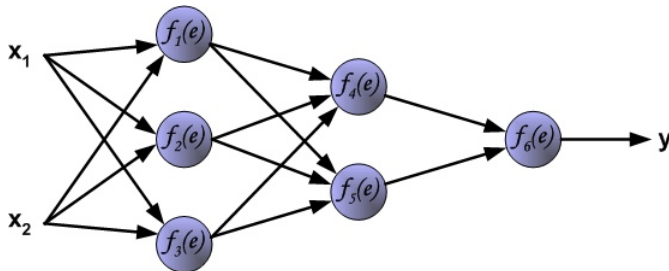
# Algorytm wstecznej propagacji

Algorytm dąży do minimalizacji błędu popełnianego przez sieć (minimalizacja funkcji straty, loss function). Ogólny schemat:

- ➊ Inicjalizuj sieć z losowymi wagami (random  $W$ )
- ➋ Wybierz próbkę (lub batch) danych ze zbioru treningowego. Oblicz wartości wyjściowe sieci dla próbki (dla *input* obliczamy *output<sub>pred</sub>*)
- ➌ Oblicz błąd sieci (np. *MSE* porównujące *output<sub>pred</sub>* i *output<sub>real</sub>*)
- ➍ Popraw wagi  $W$  poprzez propagację wsteczną błędu (minimalizujemy *Loss( $W$ )*)
- ➎ Czy sieć nauczona? Czy upłynęło max epok?
  - TAK – KONIEC
  - NIE – wróć do punktu 2

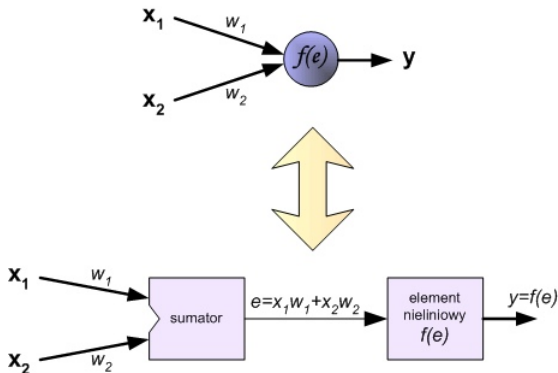
# Algorytm wstecznej propagacji

Pokażemy na przykładzie poniższej sieci jak działa algorytm [wg M.Bernacki, P. Włodarczyk, A.Gołda (2005)].



# Algorytm wstecznej propagacji

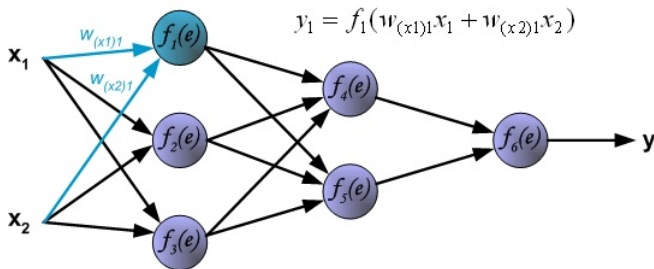
Przypomnienie jak działa neuron:





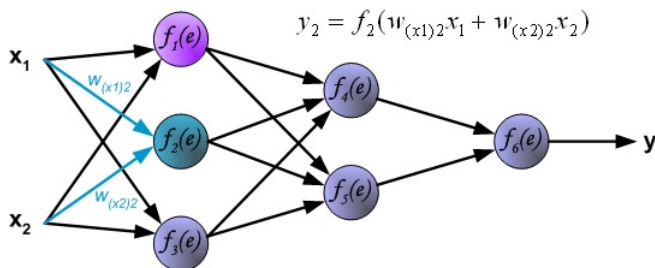
# Algorytm wstecznej propagacji

Sieć ma losowe wagi. Mając zbiór treningowy danych (wejść  $x_1, x_2$  i odpowiedzi  $y$ ) możemy je przekazywać kolejno sieci do uczenia się. Wybieramy jedną parę danych wejściowych i przekazujemy do sieci.



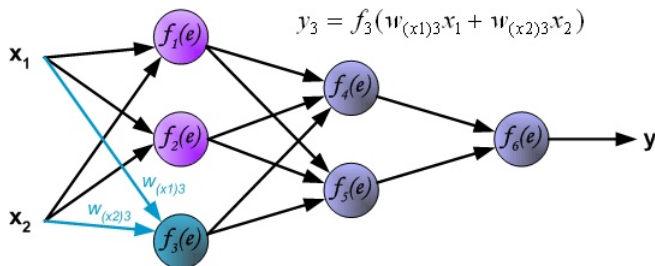
# Algorytm wstecznej propagacji

Następuje propagacja sygnału przez sieć.



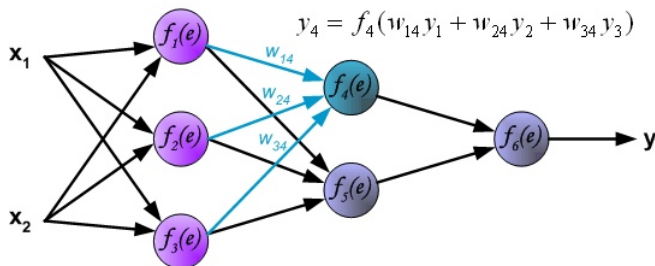
# Algorytm wstecznej propagacji

Następuje propagacja sygnału przez sieć.



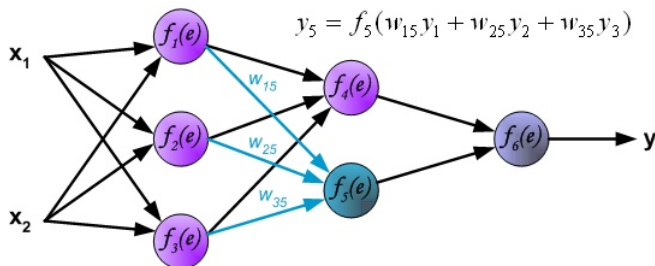
# Algorytm wstecznej propagacji

Następuje propagacja sygnału przez sieć.



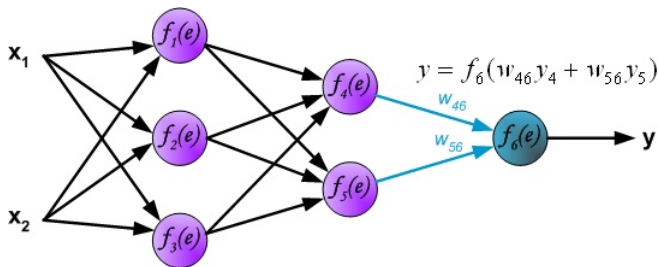
# Algorytm wstecznej propagacji

Następuje propagacja sygnału przez sieć.



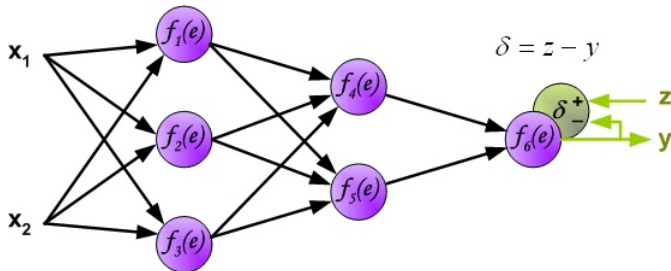
# Algorytm wstecznej propagacji

Następuje propagacja sygnału przez sieć.



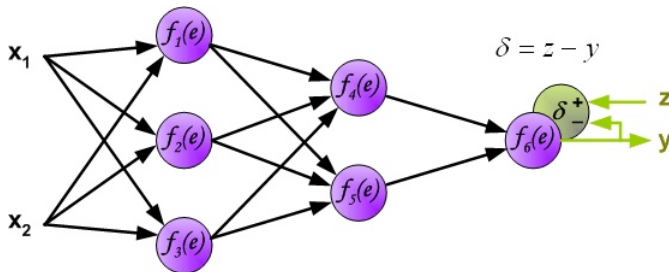
# Algorytm wstecznej propagacji

Dostając odpowiedź  $y$  porównujemy ją z prawdziwą odpowiedzią  $z$  i obliczamy różnicę: błąd  $\delta$  neuronu warstwy wyjściowej.



# Algorytm wstecznej propagacji

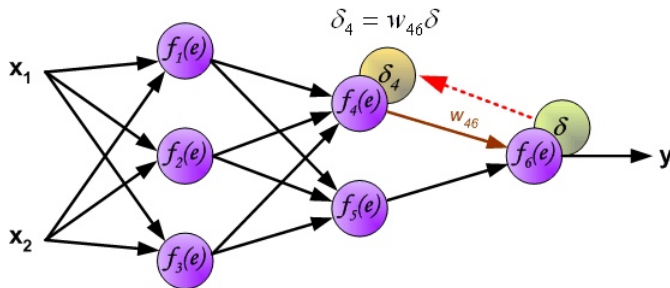
Jak policzyć błąd neuronów warstwy ukrytej? One nie mają odpowiedzi z. Do lat 80-tych nikt nie wiedział. Wówczas jednak zaproponowano algorytm wstecznej propagacji: błąd jest propagowany wstecz do warstwy wejściowej i przekazywany wszystkim neuronom.





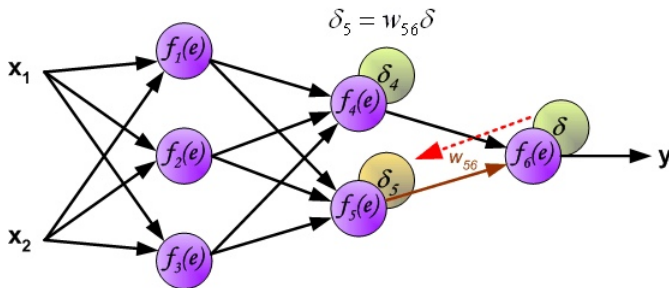
# Algorytm wstecznej propagacji

Widać, że błąd jest modyfikowany przez odpowiednią wagę.



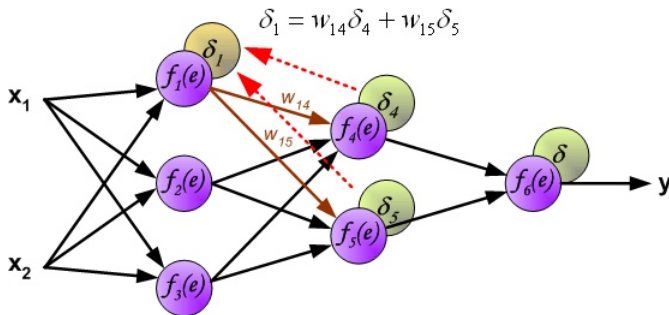
# Algorytm wstecznej propagacji

Przechodząc do warstwy głębiej, błędy z wagami sumują się. Przypomina to "normalną" propagację jakby błąd był daną wejściową "od tyłu".



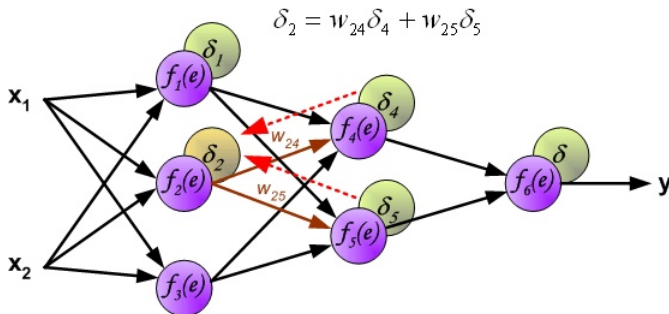
# Algorytm wstecznej propagacji

Przechodząc do warstwy głębiej, błędy z wagami sumują się. Przypomina to "normalną" propagację jakby błąd był daną wejściową "od tyłu".



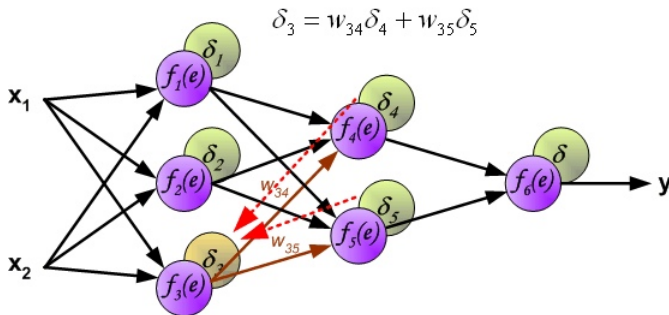
# Algorytm wstecznej propagacji

Przechodząc do warstwy głębiej, błędy z wagami sumują się. Przypomina to "normalną" propagację jakby błąd był daną wejściową "od tyłu".



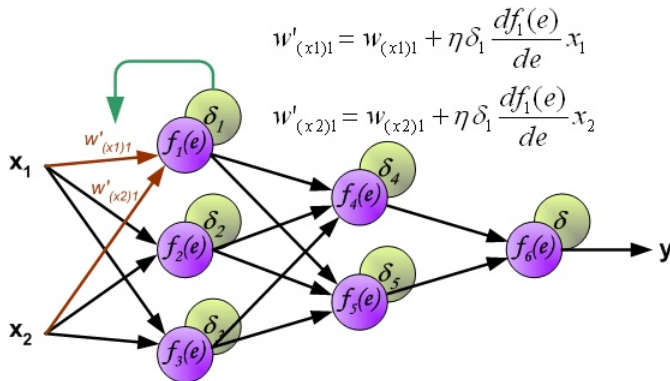
# Algorytm wstecznej propagacji

Przechodząc do warstwy głębiej, błędy z wagami sumują się. Przypomina to "normalną" propagację jakby błąd był daną wejściową "od tyłu".



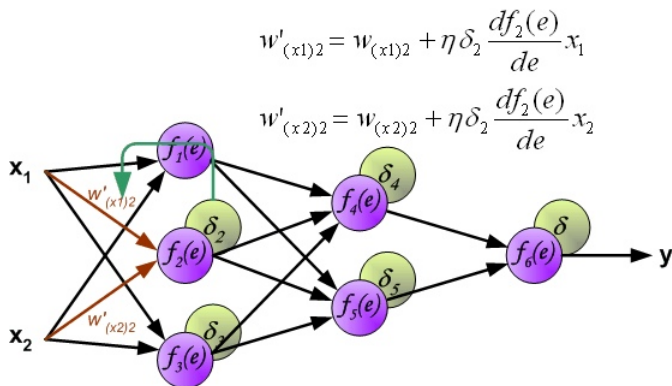
# Algorytm wstecznej propagacji

Dochodząc do warstwy wejściowej zaczynamy poprawianie wag wykorzystując do tego otrzymane błędy i pochodną funkcji wyjściowej neuronu.



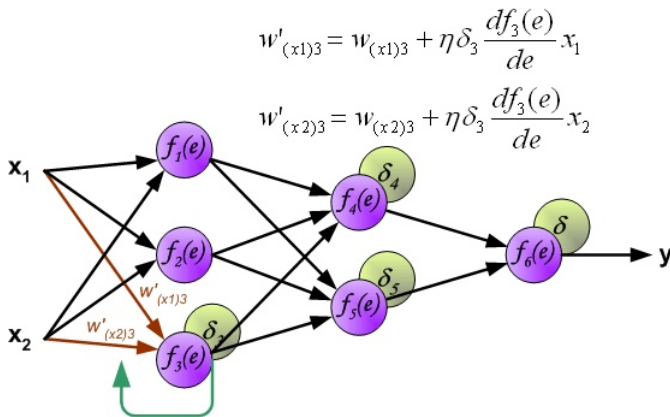
# Algorytm wstecznej propagacji

Pochodną można rozumieć jako styczną do wykresu funkcji. Wyznacza ona więc kierunek redukcji błędów. Ten spadek nazywa się metodą gradientu prostego (gradient descent).



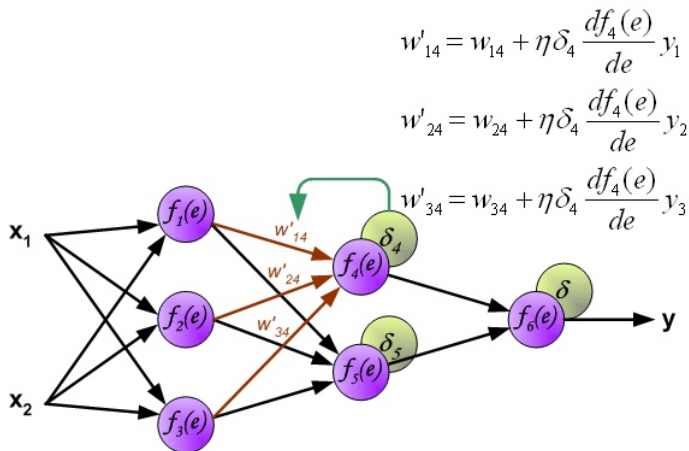
# Algorytm wstecznej propagacji

Współczynnik  $\eta$  jest dodatkowym parametrem, który może ustalić człowiek.



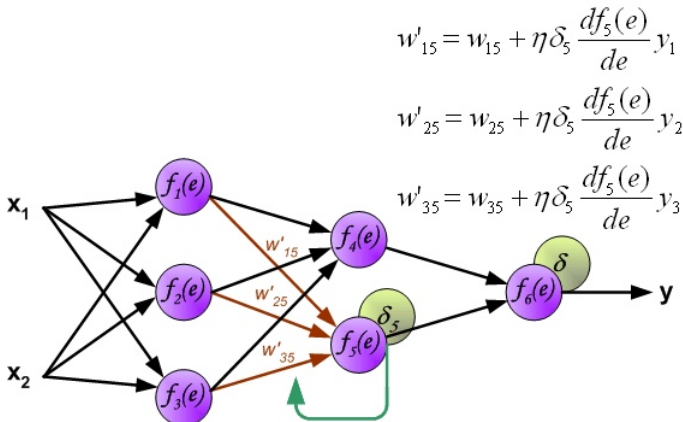


# Algorytm wstecznej propagacji



# Algorytm wstecznej propagacji

Po zakończeniu przebiegu, uruchamiamy algorytm dla kolejne pary danych zbioru treningowego i znowu poprawiamy wagi. Chcemy zredukować błąd do jak najmniejszej wartości.



# Algorytm wstecznej propagacji

Więcej o algorytmie wstecznej propagacji z fajnymi wyjaśnieniami i przykładami:

- *A Step by Step Backpropagation Example*, Matt Mazur
- *An Introduction to Gradient Descent and Backpropagation*, Abhijit Roy
- *Neural networks and back-propagation explained in a simple way*, Assaad MOAWAD

# Współczynnik uczenia się

- *Współczynnik uczenia się* (ang. *learning rate, gain*) to hiperparametr regulujący szybkość uczenia się, często oznaczany jako  $\eta$  (eta).
- *Hiperparametr* to parametr regulujący cały proces uczenia się. Zwykłe parametry (np. wagi w SSN), podlegają zmianom podczas uczenia się.
- Współczynnik ten widoczny był na poprzednich slajdach (obrazki z siecią neuronową). Ze wzorów wynikało, że im większe  $\eta$ , tym bardziej zmieniona będzie waga sieci neuronowej.
- Jeśli jest mały, to proces uczenia się jest wolny, ale stabilny. Jeśli jest duży, to proces jest szybki, ale wykonuje duże "kroki" i może przeoczyć dobre rozwiązanie.

# Współczynnik uczenia się

Współczynnik uczenia się może być stały lub może być zmieniany w czasie uczenia (iteracje, epoki) wg jakiegoś wzoru. We wzorach często używane są inne hiperparametry np. *decay* (zanik)  $d$  lub *momentum* (pęd). Przykładowy wzór na zmianę  $\eta$  pomiędzy  $(n + 1)$  a  $n$  iteracją (zwany: time-based schedule):

$$\eta_{n+1} = \frac{\eta_n}{1 + d \cdot n}$$

# Współczynnik uczenia się

To jak będziemy manipulować współczynnikiem uczenia się oraz wzorem na poprawianie wag może prowadzić do zauważalnych zmian w wydajności i szybkości uczenia się sieci. Na przestrzeni lat powstało wiele optimizerów np.

- *SGD* (Stochastic Gradient Descent)
- *Adam*
- *RMSProp*

# Epoki

- *Epoka* (ang. epoch, czytaj: 'ipok) to fragment procesu uczenia, w którym przeszliśmy dokładnie jeden raz przez każdą próbkę ze zbioru treningowego ucząc się na niej.
- Algorytm uczący ma zadaną pewną *liczbę epok* - hiperparametr mówiący, ile razy musimy przejść przez każdą próbkę w training set (ile razy powtórzyć pojedynczą epokę).
- Liczba epok musi być dopasowana do skomplikowania problemu i sieci. Może wynosić 10, 100 lub nawet 1000.
- Jeśli wielkość zbioru treningowego to 7000, a liczba epok to 100, to wówczas algorytmu wstecznej propagacji uruchomi się 700 000 razy.

# Batch

- Gdy uczymy się ze zbioru treningowego, to czy aktualizujemy wagi w sieci neuronowej po każdym przejściu przez jedną próbkę z tego zbioru?
- Próbki są różne. Mogą dość chaotycznie oddziaływać na proces uczenia się i "ciągnąć go w swoją stronę".
- Jak temu zaradzić? Wprowadzamy batche!



# Batch

- *Batch*, lub *partia*, to fragment epoki. To część proces uczenia, w którym przeszliśmy przez pewną liczbę próbek ze zbioru treningowego, po czym aktualizujemy wagi w sieci neuronowej.
- Wag w sieci neuronowej nie aktualizujemy po każdej próbce, tylko po każdym batchu!
- Każda waga jest średnią z wag obliczonych z próbek batcha.

# Batch

- Jak duże powinny być batche? *Wielkość batcha* (ang. batch size) to kolejny hiperparametr, który silnie oddziałuje na proces uczenia się.
- Duże batche powodują, że uczenie się jest wolniejsze, ale droga do celu jest prostsza.
- Dobór najlepszej wielkości batcha może wymagać kilku eksperymentów i porównywania wyników (np. na krzywej uczenia się).
- Typowe wielkości są potęgami dwójki (ze względu na architekturę komputera), czyli np.: 1, 2, 4, 8, 32, 64, 128, albo nawet wielkość całego zbioru treningowego.
- Wielu badaczy twierdzi, że małe batche (do 32) działają najefektywniej.

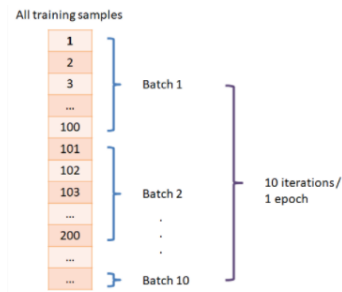
# Batch

Ze względu na wielkość batcha, proces uczenia przyjmuje różne nazwy:

- *Stochastyczny spadek wzdłuż gradientu* (ang. *Stochastic Gradient Descent*, SGD): batch jest wielkości **1**. Słowo "stochastyczny" oznacza, "losowy", "bez tendencji", bo próbki wybieramy w losowej kolejności.
- *Spadek wzdłuż gradientu mini-partiami* (ang. *Mini-Batch Gradient Descent*): batch jest wielkości większej niż 1, ale mniejszej niż wielkość zbioru treningowego.
- *Spadek wzdłuż gradientu na bazie całego zbioru uczącego* (ang. *Batch Gradient Descent*): batch jest wielkości zbioru treningowego.

# Iteracje

- Parametrem, który automatycznie powstaje po wybraniu wielkości batcha jest *liczba iteracji* wynosząca  $\frac{\text{wielkość zbioru treningowego}}{\text{wielkość batcha}}$ . Uwaga: gdy zbiór treningowy nie dzieli się równo na batche, można go zmniejszyć (dopasować) lub po prostu zaakceptować, że ostatni batch będzie mniejszy.



# Zbiór walidacyjny

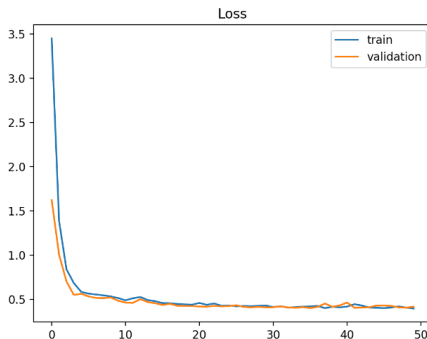
- *Zbiór treningowy* (ang. training set) - używany w procesie uczenia do poprawiania wag sieci.
- *Zbiór testowy* (ang. test set) - używany po zakończeniu procesu uczenia, do sprawdzenia dokładności sieci.
- *Zbiór walidacyjny* (ang. validation set) - zbiór do testowania w trakcie uczenia. Pomaga odpowiedzieć na pytanie: czy uczymy się dalej ze zbioru treningowego, czy już mamy dobrą wydajność i kończymy.

# Krzywa uczenia się

- *Krzywa uczenia się* (ang. learning curve) - wykres przedstawiający jaka jest wydajność sieci neuronowej (oś Y) w czasie uczenia, liczbie epok (oś X).
- Właściwie najczęściej mamy do czynienia z dwiema krzywymi na jednym wykresie: dla zbioru treningowego i zbioru walidacyjnego.
- Przez wydajność możemy rozumieć dokładność sieci jako klasyfikatora (performance, accuracy) lub odstępstwo wyników sieci od tych prawdziwych, funkcja straty (loss, optimization).

# Krzywa uczenia się

Przykład. Sieć dobrze nauczona (good fit). Krzywa dla zbioru treningowego zbiegła się z tą dla zbioru walidacyjnego.



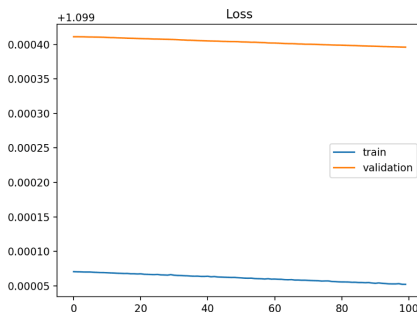
# Niedouczenie się

- Gdy model sieci neuronowej jest zbyt prosty, nie jej się dobrze wyuczyć z danych. Należy dobrać sieć neuronową o odpowiedniej złożoności.
- Mowa wówczas o niedouczeniu się (underfitting).



# Niedouczenie się

Przykład. Sieć jest niedouczona (underfit). Krzywa dla zbioru walidacyjnego jest niższa niż dla zbioru testowego. Potrzebna dalsza nauka lub dobranie lepszego modelu.



# Przeuczenie się

Przykład: Dziecko uczy się dodawać. Ma 4 działania:

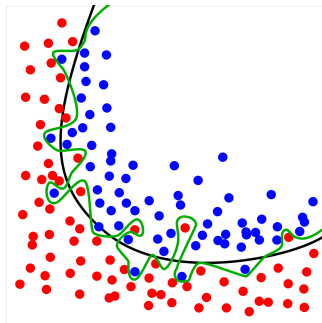
$1+1$ ,  $2+4$ ,  $3+2$ ,  $2+2$

Uczone dziecko ma dwie opcje

- Zrozumieć zasadę dodawania i posługiwać się nią w przykładach, których nie widziało
- Wyuczyć się przykładów na pamięć, wtedy jeżeli zobaczy nowy, nieznany przykład to nie będzie potrafiło go rozwiązać.

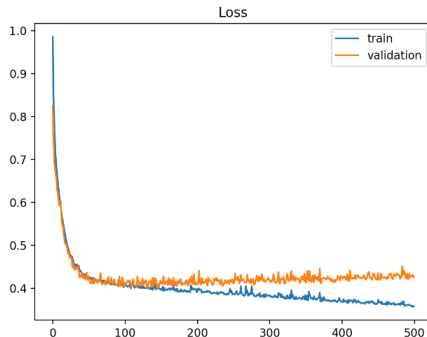
# Przeuczenie się

Sieć przeuczona zbyt dokładnie nauczyła się zbioru treningowego, co może spowodować, że w zbiorze walidacyjnym będzie popełniała więcej błędów. Czarna linia: dobrze nauczona. Zielona linia: przeuczona.



# Przeuczenie się

Sieć przeuczona zbyt dokładnie nauczyła się zbioru treningowego, co może spowodować, że w zbiorze walidacyjnym będzie popełniała więcej błędów.



# Przeuczenie się

Jak zaradzić przeuczeniu się?

- Zatrzymać uczenie w odpowiednim punkcie, tj. w momencie gdy strata dla zbioru walidacyjnego zaczyna rosnąć, lub przecina stratę dla zbioru treningowego.
- Wprowadzenie prostszego modelu sieci.
- W przypadku większych sieci można zastosować technikę *dropout*. Tymczasowe usuwanie niektórych neuronów.

# Przeuczenie się

Czasami zwróćmy uwagę czy model wystarczająco długo się uczy.

