# Sieci neuronowe od podstaw

Mam wrażenie, że wokół sieci neuronowych narosło wiele mitów. Nawet niektórzy starsi członkowie mojej rodziny na hasło „sztuczna inteligencja" robią posępną minę, niezależnie od tego, czy mowa jest o generowaniu obrazów czy – na przykład – o rozpoznawaniu odręcznego pisma czy też tłumaczeniu tekstu z jednego języka na drugi. Tymczasem sieć neuronowa, czy bardziej precyzyjnie wielowarstwowy perceptron, jest stosunkowo prostym matematycznym konstruktem, do zrozumienia którego powinna wystarczyć wiedza na poziomie szkoły średniej.

## SIEĆ NEURONOWA OD ZERA

Ci, którzy znają mnie nieco lepiej, wiedzą, że w mojej programistycznej historii wiele razy podejmowałem próbę przepisania od nowa czegoś, co zostało już dawno napisane, przetestowane i szeroko wdrożone. Napisałem więc własny raytracer, parser XML, generator tokenizerów, a także własny programistyczny edytor i analizator logów. I o ile można dyskutować nad opłacalnością takich działań w wymiarze czysto praktycznym, to każdy z wymienionych projektów przyniósł mi wymierną korzyść w postaci olbrzymiej dawki wiedzy, którą musiałem przyswoić w trakcie ich implementacji, oraz programistycznego doświadczenia, które przydało się w późniejszych etapach mojej kariery.

Dlatego też dziś, na potrzeby niniejszego artykułu, wrócę trochę do korzeni i podejmę się napisania prostego silnika implementującego funkcjonalność konstruowania i uczenia sieci neuronowych (czyli mocno uproszczony klon takich bibliotek jak TensorFlow czy PyTorch). Ponieważ jednak – jak już ustaliliśmy – podstawowym celem takiego działania jest zdobycie nowej wiedzy, postaram się cały proces opisać w taki sposób, aby czytelnik zrozumiał dokładnie, w jaki sposób i dlaczego tak naprawdę sieci neuronowe działają.

Zabierzmy się więc do roboty, bo mamy sporo do zrobienia.

## NA WSTĘPIE

Zacznijmy od ustalenia kilku podstawowych faktów.

Przede wszystkim warto mieć na uwadze, że sieć neuronowa jest tak naprawdę głównie konstruktem matematycznym, a nie programistycznym. Aspekt programistyczny pojawia się praktycznie tylko w jednym miejscu, w którym mamy do czynienia z algorytmem, a cała reszta to tak naprawdę stara, dobra analiza matematyczna. Oznacza to, że będziemy musieli trochę popracować z funkcjami i pochodnymi, ale słowo – nie będzie to nic, co wykraczałoby poza materiał szkoły średniej (w szczególności *mojej* szkoły średniej, gdzie w czwartej klasie mieliśmy podstawy ciągów, granic i pochodnych – nie wiem, jak sytuacja wygląda w tej chwili).

Druga ważna informacja. Jeżeli podjęliście kiedykolwiek próbę pracy z sieciami neuronowymi, z pewnością zderzyliście się z takimi zagadnieniami jak tensory, macierze, mnożenie macierzy i iloczyn skalarny. No i tu dobra wiadomość jest taka, że do zrozumienia istoty działania sieci neuronowych wiedza na temat wspomnianych zagadnień kompletnie nie jest potrzebna.

Skąd jednak w takim razie wszechobecność macierzy w większości bibliotek wspierających tworzenie i uczenie sieci neuronowych? Ano stąd, że nie da się bez nich obejść, jeśli chcemy zaimplementować sieci neuronowe *działające w wydajny sposób*. Skorzystanie z macierzy pozwala na przykład na zepchnięcie niektórych obliczeń na kartę graficzną, która realizuje je bardzo szybko, a do tego jeszcze zrównolegla w stopniu, o którym procesor komputera może sobie tylko pomarzyć. Ponieważ jednak zależy nam przede wszystkim na zrozumieniu istoty działania sieci, możemy z powodzeniem zaimplementować je znacznie mniej wydajnie, ale jednocześnie znacznie prościej.

Przejdźmy teraz powoli do konkretów. Na początku zdefiniujmy, czym tak naprawdę jest sieć neuronowa.

## SIEĆ NEURONOWA

Duża liczba wynalazków dokonanych przez ludzi opiera się na obserwacjach przyrody. Latamy samolotami, które zostały zbudowane w oparciu o obserwacje budowy ciała ptaków, pływamy statkami, które czerpią z anatomii ryb i innych zwierząt morskich. Aparat fotograficzny zbudowany jest bardzo podobnie do rejestrującego fale świetlne ludzkiego oka, a przy budowie budynków korzystamy z doświadczenia niektórych zwierząt, które zupełnie nieświadomie tworzą cuda architektury. Myślę więc, że nie ma nic dziwnego, że w poszukiwaniu sposobu na analizę naprawdę trudnych zestawów danych ludzkość sięgnęła do organu, który całkiem nieźle sobie z tym radzi: mózgu.

Mózg jest oczywiście bardzo skomplikowanym narządem, ale jego budowa opiera się w dużej mierze na sieci połączonych ze sobą komórek nerwowych, czyli neuronów. Każdy z nich otrzymuje na wejściu zestaw sygnałów elektrycznych, które poprzez bardzo skomplikowane procesy biochemiczne przetwarza, a następnie generuje pojedynczy sygnał, który przekazywany jest do kolejnych neuronów.

Matematyczny model neuronu stanowi zanurzenie jego biologicznego odpowiednika w świat liczb. I tak, na wejściu zamiast sygnałów elektrycznych neuron dostaje liczby rzeczywiste, zamiast procesów biochemicznych mamy sparametryzowaną funkcję, która wartości otrzymane na wejściu przetwarza, a następnie generuje wynik, będący oczywiście również liczbą. Liczba taka przekazywana jest potem do kolejnych neuronów (Rysunek 1).

MITC
MODERN INFORMATION TECHNOLOGY CERTIFICATION

# Pierwszy praktyczny egzamin dla branży IT

## Wyróżnij się na rynku pracy!

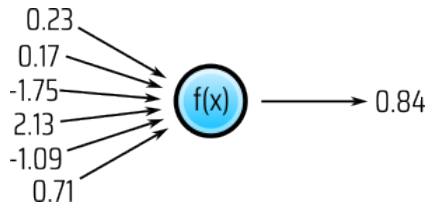
Branża IT wymaga czegoś więcej niż tylko wiedzy teoretycznej.

Udowodnij swoje praktyczne umiejętności testowania dzięki certyfikatowi MITC Software Testing Level 01.

**MITC oferuje unikalne doświadczenie egzaminacyjne w 100% online z nowoczesnym, szybkim i przyjaznym dla użytkownika systemem certyfikacji na miarę naszych czasów.**

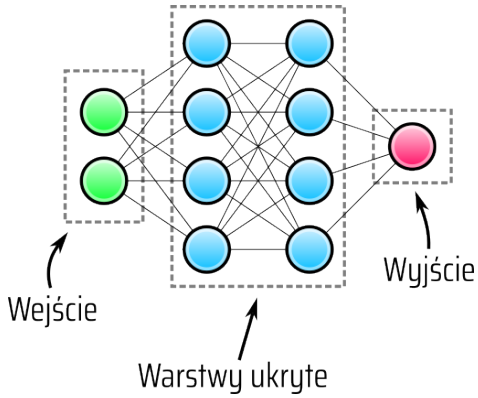
**Get IT today!**

**www.mitc.center**




*Rysunek 1. Matematyczny model neuronu*

Sieć neuronowa zbudowana jest z pojedynczych neuronów, które są ze sobą połączone. Przykładową sieć możemy zobaczyć na Rysunku 2.




*Rysunek 2. Przykładowa sieć neuronowa*

Przetwarzanie odbywa się od lewej do prawej. Neurony po lewej stronie nazywane są neuronami wejściowymi i mają za zadanie tylko przechowywanie danych wejściowych. Ich liczba zależy ściśle od tego, ile wymiarów mają dane, które będą przetwarzane przez sieć. Jeżeli na przykład chcemy wytrenować sieć do przewidywania pogody, przekazując jej dane temperatury, kierunku i siły wiatru oraz ciśnienia i wilgotności powietrza, to w warstwie wejściowych znajdzie się 5 neuronów.

Pamiętajmy o tym, że neurony nie mają za zadanie przechowywania *pojedynczych wystąpień* danych, ale *elementy, z których składa się pojedyncze wystąpienie*. Jeżeli na przykład chcielibyśmy nauczyć sieć sprawdzania, czy przekazywany jej punkt należy do określonego obszaru na płaszczyźnie, w warstwie wejściowej znalazłyby się tylko dwa neurony, którym przekazywalibyśmy współrzędne X i Y konkretnego punktu (składniki pojedynczego wystąpienia danych). I teraz jeżeli takich punktów (wystąpień danych) byłoby 100, konieczne byłoby wywołanie sieci 100 razy, po raz dla każdego punktu wchodzącego w skład przetwarzanego zbioru danych.

Zaraz po warstwie wejściowej znajdują się warstwy nazywane warstwami ukrytymi. Mogą one zawierać dowolne liczby neuronów – w zależności od tego, jaką sieć chcemy zbudować.

Neurony pierwszej warstwy ukrytej na wejściu otrzymują wartości z neuronów wejściowych. Z kolei neurony drugiej warstwy jako dane wejściowe otrzymują wyniki przetworzenia danych wejściowych przez pierwszą warstwę. Neurony z trzeciej warstwy karmione są wynikami wygenerowanymi przez warstwę drugą – i tak dalej.

Siłą rzeczy taka konstrukcja struktury sieci neuronowej jasno definiuje, ile danych wejściowych przyjmuje neuron. Te, które znajdują się w pierwszej warstwie ukrytej, przyjmują tyle argumentów, ile znajduje się neuronów w warstwie wejściowej. Te z drugiej warstwy ukrytej przyjmują tyle argumentów, ile neuronów znajduje się w pierwszej warstwie ukrytej – i tak dalej.

Ostatnia warstwa neuronów nie różni się niczym od warstw ukrytych poza nazwą – jest nazywana warstwą wyjściową, ponieważ z neuronów znajdujących się w niej odczytujemy wynik przetwarzania danych wejściowych przez sieć. Liczba neuronów w tej warstwie musi więc odpowiadać wymiarowi danych wynikowych. Powiedzmy na przykład, że nasza sieć prognozująca pogodę ma zwrócić temperaturę i ciśnienie powietrza za 24 godziny. Warstwa wyjściowa będzie wówczas zawierała dwa neurony.

W jaki dokładnie sposób neuron przetwarza dane? Wbrew pozorom nie ma tu wcale zbyt wielkiej filozofii.

Dla każdego wejścia $x_i$ neuron przechowuje parametr zwany wagą i oznaczany jako $w_i$. Oprócz tego, niezależnie od liczby wejść, neuron przechowuje również dodatkowy parametr oznaczany jako b i nazywany biasem. Jeżeli więc neuron przyjmuje na wejściu 4 argumenty, to będzie on przechowywał 5 parametrów: $w_0$, $w_1$, $w_2$, $w_3$ i b.

W momencie, w którym neuron zostanie poproszony o obliczenie swojej wartości, przemnoży on każdy z argumentów przez odpowiadającą mu wagę, a następnie obliczy sumę tych iloczynów, dodając do niej jeszcze bias (Wzór 1).

$$N(x_1, x_2, x_3, ..., x_n) = (x_1 \cdot w_1) + (x_2 \cdot w_2) + ... + (x_n \cdot w_n) + b$$

*Wzór 1. Sposób przetwarzania danych przez neuron*

Rodzi się tu mały problem. Okazuje się bowiem, że jeżeli zdefiniujemy funkcję neuronu w taki sposób, to każdą sieć neuronową dowolnych rozmiarów możemy zastąpić pojedynczym neuronem, którego wagi i bias stanowią kombinację wag i biasów neuronów wchodzących w jej skład.

Zobaczmy to na przykładzie – weźmy sieć, która ma 2 neurony wejściowe, a następnie warstwy o liczbie neuronów, odpowiednio, 3, 2 i 1. Jeżeli połączymy we właściwy sposób funkcje wszystkich neuronów, całą sieć możemy opisać pojedynczym wzorem (Wzór 2). Zapis LxNyWz oznacza wagę z neuronu y na warstwie x, licząc od 0. LxNyB oznacza bias odpowiedniego neurona. Wejścia oznaczone są jako input0 oraz input1.

$$\begin{aligned}
&L2N0W1 \cdot (L1N1W2 \cdot (L0N2W1 \cdot input1 + L0N2W0 \cdot input0 + L0N2B) + \\
&L1N1W1 \cdot (L0N1W1 \cdot input1 + L0N1W0 \cdot input0 + L0N1B) + \\
&L1N1W0 \cdot (L0N0W1 \cdot input1 + L0N0W0 \cdot input0 + L0N0B) + L1N1B) + \\
&L2N0W0 \cdot (L1N0W2 \cdot (L0N2W1 \cdot input1 + L0N2W0 \cdot input0 + L0N2B) + \\
&L1N0W1 \cdot (L0N1W1 \cdot input1 + L0N1W0 \cdot input0 + L0N1B) + \\
&L1N0W0 \cdot (L0N0W1 \cdot input1 + L0N0W0 \cdot input0 + L0N0B) + L1N0B) + L2N0B
\end{aligned}$$

*Wzór 2. Kompletny wzór prostej sieci neuronowej*

Przetwarzanie tak długiego wzoru stanowi mordęgę, ale możemy pomóc sobie nieco Maximą, żeby go uporządkować. Po rozwinięciu nawiasów, a następnie pogrupowaniu wyrażeń, otrzymujemy kolejny wzór (Wzór 3).

$$\begin{aligned}
&((L0N2W0 \cdot L1N1W2 + L0N1W0 \cdot L1N1W1 + L0N0W0 \cdot L1N1W0)) \cdot L2N0W1 + \\
&(L0N2W0 \cdot L1N0W2 + L0N1W0 \cdot L1N0W1 + L0N0W0 \cdot L1N0W0) \cdot L2N0W0) \cdot input0 + \\
&((L0N2W1 \cdot L1N1W2 + L0N1W1 \cdot L1N1W1 + L0N0W1 \cdot L1N1W0) \cdot L2N0W1 + \\
&(L0N2W1 \cdot L1N0W2 + L0N1W1 \cdot L1N0W1 + L0N0W1 \cdot L1N0W0) \cdot L2N0W0) \cdot input1 + \\
&(L0N2B \cdot L1N1W2 + L0N1B \cdot L1N1W1 + L0N0B \cdot L1N1W0 + L1N1B) \cdot L2N0W1 + \\
&(L0N2B \cdot L1N0W2 + L0N1B \cdot L1N0W1 + L0N0B \cdot L1N0W0 + L1N0B) \cdot L2N0W0 + L2N0B
\end{aligned}$$

*Wzór 3. Uporządkowany Wzór 2*

Wystarczy, że przyjrzymy mu się teraz dobrze, by zauważyć, że w swojej istocie faktycznie stanowi on wzór pojedynczego neuronu – tyle tylko, że wartości wag i biasu zapisane są w bardzo skomplikowany sposób. Zawartość zielonego nawiasu stanowi wagę dla wejścia 0, zawartość czerwonego – wagę dla wejścia 1, zaś cały niebieski nawias to po prostu bias.

Aby zapobiec takiemu „spłaszczeniu" sieci do pojedynczego neuronu, kompletny wzór neuronu zawiera jeszcze jeden element, zwany funkcją aktywacji (Wzór 4).

$$N(x_1, x_2, x_3, ..., x_n) = f_a((x_1 \cdot w_1) + (x_2 \cdot w_2) + ... + (x_n \cdot w_n) + b)$$

*Wzór 4. Kompletny wzór neuronu wraz z funkcją aktywacji*

Choć konstrukcja sieci neuronowej powinna już być w tym momencie jasna, pozostają jeszcze dwa pytania, na które nie mamy odpowiedzi. Jakiej funkcji aktywacji użyć w neuronach? I jakie powinny być wartości wag i biasów w zależności od problemu, który chcemy rozwiązać?

Odpowiedź na pierwsze pytanie jest na szczęście dosyć prosta. Inżynierowie, którzy pracowali nad sieciami neuronowymi, wyróżnili szereg funkcji, które nadają się do bycia funkcjami aktywacji. Są wśród nich funkcje takie jak tangens hiperboliczny, funkcja ReLu, „przeciekająca ReLu" i inne.

Może to na pierwszy rzut oka brzmieć strasznie, ale spieszę uspokoić – funkcje te zostały dobrane przede wszystkim ze względu na ich przebieg, a nie konkretne własności. Na przykład tangens hiperboliczny w okolicach (-2, 2) zachowuje się mniej więcej jak funkcja f(x)=x, argumenty większe od 2 zamienia w wartości zbiegające do 1, zaś argumenty mniejsze od -2 zamienia w wartości zbiegające do -1 (Rysunek 3). Ot i cała tajemnica.

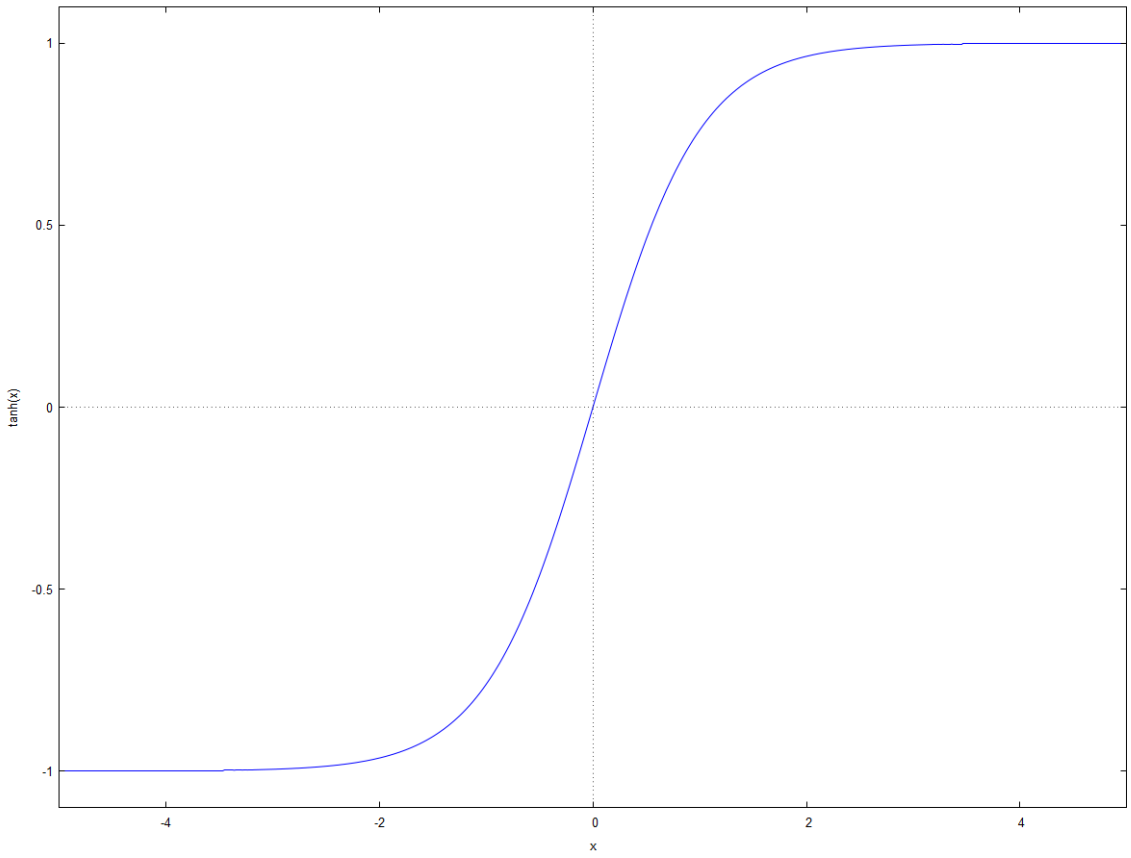
Jeśli zaś chodzi o wagi i biasy, odpowiedź może być nieco zaskakująca: na początku nigdy nie wiemy, jakie powinny być ich wartości. Już tłumaczę, dlaczego.

Jak pamiętamy, parametry sieci neuronowej – czyli właśnie wagi i biasy neuronów – służą do precyzyjnego sterowania jej działaniem. Z drugiej strony jednak zadaniem sieci jest rozwiązanie problemu, do którego trudno byłoby podejść w inny sposób – na przykład klasyfikacja obrazów. Skoro nie mamy pojęcia, jak rozwiązać ten problem zwykłymi, deterministycznymi metodami, trudno jest przypuszczać, że będziemy w stanie podać wartości wag i biasów, które pozwolą problem ten rozwiązać sieci neuronowej.

Dlatego też wartości wag i biasów wyznaczane są w trakcie specjalnego procesu, zwanego procesem uczenia sieci. I to właśnie na tym procesie skupimy się przez większość pozostałej części tego artykułu. Aby jednak zrozumieć jego przebieg, musimy przyswoić sobie kilka matematycznych terminów. Jednak bez obaw – dołożę starań, aby każdy z nich wyjaśnić w jak najprostszy sposób.

## PROBLEMY OPTYMALIZACYJNE

Matematycy – jak chyba większość innych inżynierskich zawodów – mają tendencję do nadawania prostym rzeczom skomplikowanych nazw. Termin „problemy optymalizacyjne" oznacza bowiem ogólnie grupę takich problemów, w przypadku których ze zbioru wielu




*Rysunek 3. Wykres funkcji y = tanh(x)*