Uczenie głębokie — zarys

Bartosz Hanc

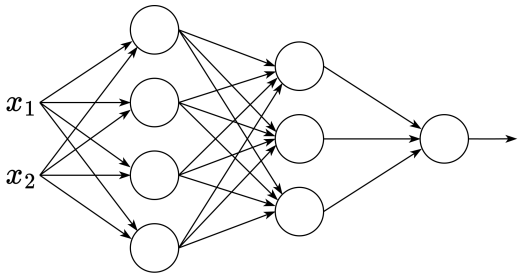
11 lutego 2025

1 Wprowadzenie

Historia sztucznych sieci neuronowych sięga lat 50. XX wieku, kiedy to Frank Rosenblatt zaproponował model perceptronu – prostej 3-warstwowej sieci, której zadaniem miało być rozpoznawanie znaków alfanumerycznych. Podstawową jednostką przetwarzającą w takiej sieci był neuron McCullocha–Pittsa, który realizował następujące przekształcenie $F : \mathbb{R}^n \mapsto \mathbb{R}$

$$F(\boldsymbol{x}) = \varphi\left(\boldsymbol{x}\boldsymbol{w}^\intercal + b\right) ,$$

gdzie $\boldsymbol{x} = (x_1, \ldots, x_n)$ określa sygnały wchodzące do neuronu, φ jest pewną nieliniową funkcję zwaną funkcją aktywacji, a $\boldsymbol{w}$ i b to pewne parametry (zwane również wagami) określające, które z sygnałów mają być wzmocnione, a które osłabione. Perceptron jest zbudowany z wielu takich neuronów połączonych ze sobą. Neurony te są zorganizowane w warstwy, w obrębie których nie ma połączeń, natomiast między sąsiednimi warstwami neurony są połączone „każdy z każdym". Na Rysunku 1 przedstawiono schemat takiego prostego perceptronu wielowarstwowego. Strzałki pokazują przepływ sygnałów w sieci, natomiast każdy wierzchołek reprezentuje odwzorowanie nieliniowe F z właściwymi dla siebie wagami. Efektywnie perceptron wielowarstwowy realizuje więc pewne przekształcenie nieliniowe $\mathbb{R}^n \mapsto \mathbb{R}^m$.



Rysunek 1: Schemat prostego perceptronu wielowarstwowego realizującego odwzorowanie $\mathbb{R}^2 \mapsto \mathbb{R}$

Podany powyżej opis konstrukcji sztucznych sieci neuronowych, choć często przywoływany we wprowadzeniach, nie jest zbyt użyteczny w nowoczesnym podejściu do uczenia głębokiego. Bardziej przydatnym sformułowaniem, czym właściwie są sieci neuronowe jest następująca definicja: sieć neuronowa to dowolny skierowany graf acykliczny, którego wierzchołki reprezentują dowolną sparametryzowaną funkcję o argumentach i wartościach tensorowych, a skierowane krawędzie reprezentują przepływ tych tensorów. Sieć neuronowa realizuje więc pewne przekształcenie wejściowych tensorów (przykładowo może to być 4-wymiarowy tensor reprezentujący zbiór kolorowych obrazków). Trening sieci neuronowej będzie polegał na takiej zmianie jej parametrów, aby sieć realizowała określone zadanie.

Pojęcie tensora w uczeniu maszynowym nie ma wiele wspólnego z tensorami używanymi w algebrze, czy w fizyce. W naszym kontekście tensor to po prostu wielowymiarowa tablica liczb. Standardowo w tym tekście tensory będziemy oznaczać literami wytłuszczonymi, np. $\boldsymbol{A}$, natomiast odnosząc się do ustalonego elementu tensora $\boldsymbol{A}$ będziemy zapisywać $A_{\alpha_1,\ldots,\alpha_k}$ lub A_α, gdzie α będzie oznaczać wielowskaźnik $(\alpha_1, \ldots, \alpha_k)$. Liczbę k nazywamy wymiarem tensora $\boldsymbol{A}$. Zbiór wszystkich tensorów wymiaru k o liczbie elementów wzdłuż osi i równej n_i będziemy oznaczać $\mathbb{R}^{(n_1,\ldots,n_k)}$.

Mamy więc graf skierowany posiadający N wierzchołków. Dla każdego wierzchołka $i \in [N]$ wprowadzimy następujące wielkości:

- $\mathscr{N}_i$ – zbiór następników, tj. wierzchołków $j \in [N]$, dla których istnieje krawędź skierowana (i, j),
- $\mathscr{P}_i$ – zbiór poprzedników, tj. wierzchołków $j \in [N]$, dla których istnieje krawędź skierowana (j, i),
- $\boldsymbol{F}^{(i)}\left(\cdot; \boldsymbol{\theta}^{(i)}\right)$ – funkcję tensorową realizowaną przez dany wierzchołek, sparametryzowaną przez $\boldsymbol{\theta}^{(i)}$,
- $\boldsymbol{v}^{(i)}$ – wartość funkcji $\boldsymbol{F}^{(i)}$ przechowywaną aktualnie w wierzchołku (w szczególności może to być tzw. null – $\varnothing$).

Obliczenie wartości $\boldsymbol{v}^{(i)}$ możemy wyrazić rekurencyjnie jako

$$\boxed{\boldsymbol{v}^{(i)} = \boldsymbol{F}^{(i)}\left[\left(\boldsymbol{v}^{(j)}\right)_{j\in\mathscr{P}_i}; \boldsymbol{\theta}^{(i)}\right].}$$

Jeśli teraz wartości $\boldsymbol{v}$ będziemy obliczać w kolejności zwróconej przez sortowanie topologicznie wierzchołków grafu to przy propagacji sygnału w przód odwiedzimy każdy wierzchołek jedynie raz.

Jako przykład rozważmy wspomniany na wstępie perceptron wielowarstwowy składający się z N warstw w pełni połączonych (ang. *fully connected layers*). Sieć taka jest reprezentowana przez skierowaną ścieżkę, której każdy wierzchołek (warstwa w pełni połączona) realizuje przekształcenie $\boldsymbol{F} : \mathbb{R}^{(\dots,n)} \mapsto \mathbb{R}^{(\dots,m)}$

$$\boldsymbol{F}(\boldsymbol{X}; \boldsymbol{W}, \boldsymbol{b}) = \varphi\left(\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}\right),$$

gdzie $\boldsymbol{W} \in \mathbb{R}^{(n,m)}$, $\boldsymbol{b} \in \mathbb{R}^m$ są parametrami tego przekształcenia, a φ to pewna nieliniowa funkcja aktywacji. Warto zaznaczyć, iż $\boldsymbol{X}$ jest dowolnym $k \geq 1$ wymiarowym tensorem, którego ostatnia oś ma n elementów (nie musi to być koniecznie macierz, czy wektor). Łatwo zrozumieć powyższy wzór wypisując jawne wyrażenie na element wartości funkcji $\boldsymbol{F}$

$$F_{\alpha\mu}(\boldsymbol{X}; \boldsymbol{W}, \boldsymbol{b}) = \varphi\left(\sum_{\beta=1}^{n} X_{\alpha\beta} W_{\beta\mu} + b_\mu\right),$$

gdzie α to wielowskaźnik, natomiast β i μ to skalarne indeksy. Obliczenie wartości $\boldsymbol{v}^{(i)}$ można dla takiej sieci wyrazić zatem rekurencyjnie jako

$$\boldsymbol{v}^{(i)} = \varphi\left(\boldsymbol{v}^{(i-1)}\boldsymbol{W}^{(i)} + \boldsymbol{b}^{(i)}\right)$$

gdzie $\boldsymbol{v}^{(0)}$ to wejście do sieci neuronowej, co wynika z faktu, iż dla ścieżki $\mathscr{N}_i = \{i+1\}$, $\mathscr{P}_i = \{i-1\}$ (zakładając naturalną numerację wierzchołków).

2 Funkcje straty

Idea treningu sieci neuronowych polega na znalezieniu takich wartości parametrów sieci Θ, aby sieć realizowała określone zadanie. Zadanie to jest specyfikowane w postaci zbioru przykładów $\mathcal{X}$, który zawiera wzorce odpowiedzi modelu. W najbardziej typowym przypadku uczenia nadzorowanego zbiór $\mathcal{X}$ składa się z par zawierających wektor cech i prawidłową etykietę dla nich. Zbiór ten może mieć jednak również inną postać, np. w przypadku trenowania modeli generatywnych dla obrazków, składa się on jedynie z pewnych zdjęć i nie zawiera żadnych etykiet. Nie wchodząc w żadne szczegóły dotyczące struktury samego zbioru $\mathcal{X}$, w każdym z nich trening sieci wygląda podobnie. Przede wszystkim wprowadzamy funkcję $L : 2^{\mathcal{X}} \times \Theta \mapsto \mathbb{R}$ zwaną funkcją kosztu lub straty (ang. *cost, loss function*), która w sposób ilościowy określa jak sieć neuronowa z danymi parametrami $\boldsymbol{\theta}$ radzi sobie na dowolnym podzbiorze danych. Następnie wykorzystujemy algorytmy optymalizacji, aby znaleźć parametry które minimalizują stratę (na całym zbiorze lub na jego części)

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} L(\mathcal{X}, \boldsymbol{\theta}).$$

Zauważmy, że w takim podejściu istnieje pewna pułapka. Otóż sieć neuronową (czy też dowolny inny model uczenia maszynowego) uczymy na zbiorze danych treningowych. Nie interesuje nas jednak tak naprawdę jego wydajność na tym zbiorze, tylko na zbiorze nowych, niewidzianych w czasie treningu danych (czyli interesuje nas generalizacja). Szukanie minimum globalnego funkcji kosztu na zbiorze treningowym może wówczas doprowadzić do nadmiernego dopasowania (ang. *overfitting*), czyli sytuacji, w której model bardzo dobrze radzi sobie na danych treningowych natomiast fatalnie radzi sobie na danych testowych, których nie były dostępne podczas treningu.

Pozostaje pytanie, jak właściwie konstruujemy funkcje kosztu. Jednym z lepiej umotywowanych podejść (choć nie jedynym i chyba nawet nie najbardziej rozpowszechnionym) jest konstrukcja w oparciu o kryterium maksymalnej wiarygodności. Zasadnicza idea jest taka, iż modelujemy nasze dane jako pochodzące z pewnej sprametryzowanej rodziny rozkładów prawdopodobieństwa, a parametry tego rozkładu są zależne od parametrów sieci. Następnie maksymalizujemy funkcję wiarygodności dla zbioru danych treningowych, przy czym ową maksymalizację wyrażamy najczęściej jako minimalizację zanegowanej logarytmicznej funkcji wiarygodności (ang. *Negated Log-Likelihood*). Poniżej rozpatrzymy szczegółowo konstrukcję dwóch najczęściej używanych funkcji straty odpowiednio dla zadań regresji i klasyfikacji.

2.1 Błąd średniokwadratowy
2.2 Entropia krzyżowa
3 Metody optymalizacji
4 Wsteczna propagacja błędu
5 Funkcje aktywacji
6 Inicjalizacja
7 Regularyzacja