Podstawy Sztucznej Inteligencji (PSZT)

Projekt 2. – Uczenie maszynowe: Defaults 2

Zespół w składzie:

1. Paweł Szafrański, 293155
2. Paweł Wieczorek, 300283

Treść zadania

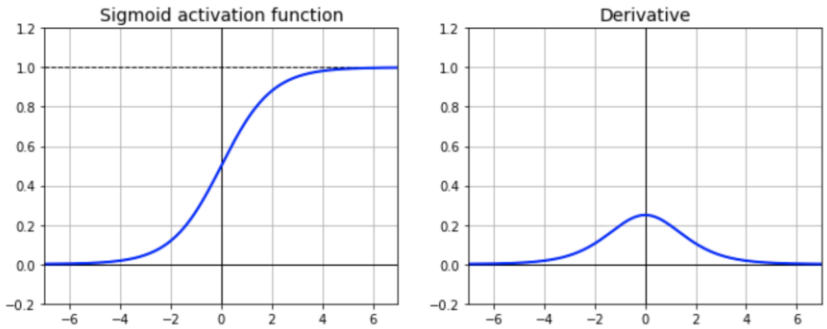
Celem projektu jest przygotowanie sieci neuronowej klasyfikującej klienta banku do jednej z dwóch klas. Należy przy tym wykorzystać zbiór danych *„default of credit card clients”* [1] zawierający informacje z 23 niezależnych kategorii i przynależność do grup reprezentowanych przez cyfry 0 oraz 1. Podczas rozwiązywania projektu szczególny nacisk należy postawić na analizę poprawności predykcji wytrenowanej sieci neuronowej w zależności od ilości warstw ukrytych oraz liczby neuronów w nich się znajdujących.

Opis rozwiązania

W przygotowanym przez nas programie zdecydowaliśmy się zaimplementować jeden z najpopularniejszych rodzajów sieci neuronowej, tj. perceptron wielowarstwowy, włącznie ze specjalnym przypadkiem, gdy warstwa jest tylko jedna (perceptron jednowarstwowy). Jako funkcje aktywacji postanowiliśmy wybrać Leaky reLU z parametrem alfa=0.01 dla neuronów ukrytych i Sigmoid dla neuronu wyjściowego. Zdecydowaliśmy się na takie funkcje z kilku powodów.

Przede wszystkim, sigmoid jest polecany w literaturze jak dobra funkcja aktywacji dla neuronu wyjściowego w klasyfikatorze binarnym, gdyż zwraca wartości z zakresu <0;1>, czyli, w naszym przypadku, prawdopodobieństwo przynależności do klasy „1”. Jednakże, w przypadku użycia sigmoidu dla neuronów ukrytych występuje problem „zanikającego” gradientu. Wynika on z tego, że dla wartości funkcji bliskich zero lub jeden, pochodna dąży do zera, czego ilustracją jest zamieszczony poniżej wykres.

Odpowiedzią na ten problem może być użycie funkcji reLU, ale ona również ma swoją wadę, tj. problem umierających neuronów. W związku z tym, koniec końców zdecydowaliśmy się na zastosowanie funkcji aktywacji dla warstw ukrytych Leaky reLU, która jest pozbawiona wyżej wymienionej wady oraz Sigmoid dla neuronu wyjściowego.



Rys. 1.: Wykres funkcji oraz pochodnej sigmoid

Obie funkcje aktywacyjne są zdefiniowane następującymi równaniami:

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad R(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Równanie dla funkcji Sigmoid ($\sigma(x)$) oraz Leaky reLu (R(x))

Na koniec, jako funkcję starty wybraliśmy entropię krzyżową (cross-entropy). Zdecydowaliśmy się na ten rodzaj funkcji, gdyż w dostępnych nam materiałach była ona polecana do zagadnienia klasyfikacji binarnej. Warto zaznaczyć, że dla problemu binarnego jest ona nieznacznie inna od wersji dla większej liczby kategorii.

$$\xi(t, y) = -t \log(y) - (1 - t) \log(1 - y)$$

Podział zadań

Paweł Szafrański	- wczytanie zbioru danych i podział ich na zestaw testowy oraz weryfikacyjny - wybranie oraz implementacja funkcji aktywacyjnej - obsługa parametrów programu
Paweł Wieczorek	- przygotowanie kodu do obsługi sieci neuronowej - wyświetlanie wykresów zebranych rezultatów - testowanie programu i zbieranie danych do badań

Procedura testowania

Do uruchomienia przygotowanego programu wymagany jest interpreter Python w wersji co najmniej 3.8. Dołączony do kodu projektu plik README.md opisuje sposób instalacji zależności, uruchomienia oraz listę dostępnych opcji. Po uruchomieniu programu zostanie wypisana informacja o wczytanym zbiorze danych wejściowych z podanego pliku CSV oraz wyniki uzyskiwane dla każdego pokolenia treningu. Po zakończeniu trenowania, program zapisuje zebrane rezultaty do pliku *results.csv*, co pozwala na dalszą ich obróbkę w arkuszu kalkulacyjnym.

Sam program wykorzystuje moduł *pandas* do wczytania danych wejściowych z pliku CSV oraz *numpy* do wygodnych operacji na wektorach i macierzach. Przykładowa komenda uruchomienia:

```
./skynet.py --neurons 3 3 --learn_rate 0.1 --epochs 100
```

Wykonane badania

Wpływ ilości i rozmiaru warstw ukrytych

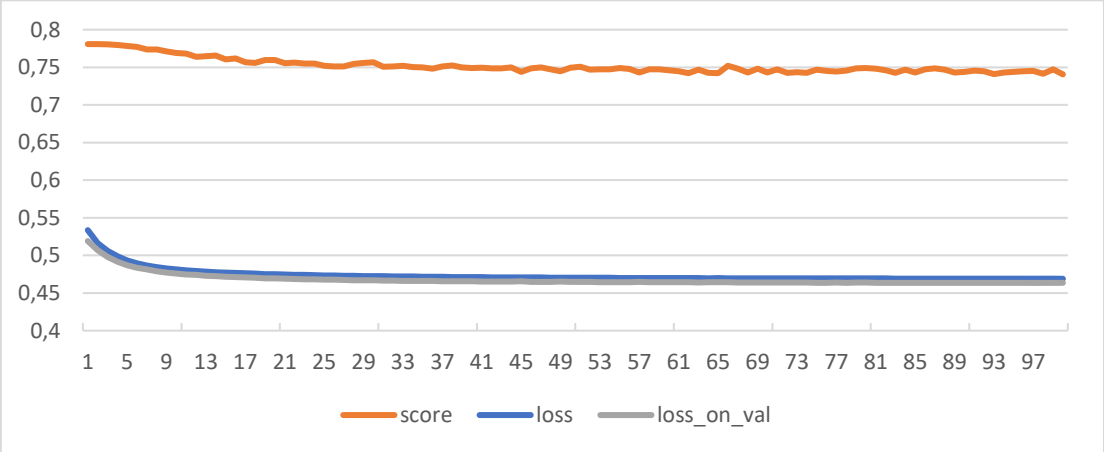
Teza: Dla pewnych ścisłych wymiarów warstw uzyskiwany procent poprawnych odpowiedzi dla danych walidacyjnych będzie najlepszy.

W celu przeprowadzenia tego badania uruchomiliśmy sieć neuronową z różną konfiguracją warstw ukrytych i obserwowaliśmy jej rezultaty po 100 epokach cyklu nauczania. Wyniki ostatnich epok dla badanych sieci zostały zebrane w poniższej tabeli.

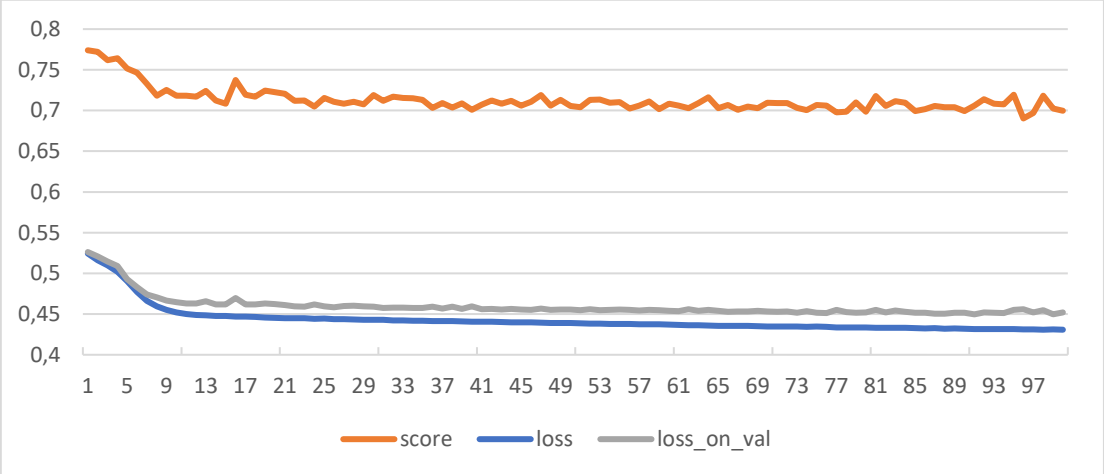
Konfiguracja warstw ukrytych	Procent trafień	F-cja straty dla trenowanych danych	F-cja straty dla danych walidacyjnych
[]	74.5%	0.468	0.463
[5]	74.4%	0.465	0.468
[10]	71.6%	0.444	0.444
[20]	72.0%	0.442	0.442
[5, 5]	71.1%	0.442	0.443
[15, 3]	70.1%	0.432	0.441
[20, 5]	71.5%	0.447	0.450
[3, 3, 3]	78.2%	0.531	0.524
[20, 10, 5]	70.0%	0.431	0.452
[20, 15, 5]	70.3%	0.428	0.436
[20, 20, 15, 5]	71.0%	0.428	0.434

Tab. 1: Wyniki nauczania przy współczynniku uczenia = 0.01

Ponadto, dla wybranych wymiarów warstw ukrytych wygenerowaliśmy wykresy prezentujące wartości mierzonych parametrów dla kolejnych epok. Wykorzystanie tak niskiej ilości epok było podyktowane długim czasem trenowania sieci zbudowanych z wielu obszernych warstw.



Wyk. 1.: Rezultaty dla konfiguracji [0] – brak warstw ukrytych



Wyk. 2: Rezultaty dla konfiguracji [20, 10, 5]

Z powyższych wykresów widać w pierwszej kolejności, że sieć neuronowa nie jest w stanie wiernie przewidzieć właściwej odpowiedzi. Początkowo, wartość procenta trafień oscyluje w okolicach 75-80%, co jest bardzo bliskie ilości kategorii „0” w używanym zestawie danych wynoszącej 77.8% wszystkich wierszy. Oznacza to, że pierwotnie model zwraca wartość klasyfikowaną jako 0 w znacznej ilości przypadków. Wraz z kolejnymi pokoleniami, sieć stara się znaleźć relacje między danymi wejściowymi, co można zaobserwować w postaci malejących wartości funkcji straty dla trenowanych danych (*loss)*. Jednakże, wraz ze spadkiem tego parametru, maleje również jakość predykcji – model zaczyna zwracać coraz więcej jedynek, ale w niewłaściwych miejscach.

Dodatkowo, na drugim wykresie można zaobserwować subtelny przejaw przetrenowania sieci neuronowej. Z każdym kolejnym cyklem rośnie różnica między wartością funkcji straty dla zbioru na którym trenujemy oraz odrębnego zbioru walidacyjnego.

W związku z bardzo marnymi efektami trenowania sieci nie jesteśmy w stanie z dużą pewnością ustalić, które wymiary sieci neuronowej sprawdzają się najlepiej. Jedne co możemy stwierdzić to, że w testowanych przypadkach, sieć [3, 3, 3] uzyskiwała najlepszy procent trafień.

Wnioski końcowe

Na bazie przeprowadzonych eksperymentów doszliśmy do wniosku, że sieć neuronowa nie jest dobrym algorytmem uczenia maszynowego dla otrzymanego zestawu danych. Niezależnie od długości treningu, czy konfiguracji wymiarów i liczności warstw ukrytych, nie zdołaliśmy uzyskać modelu z większym współczynnikiem poprawności, niż trywialne zwracania wartości „0” dla każdego wejścia.

W trakcie prac nad projektem, poświęciliśmy dużo czasu na próbę znalezienia lepszej konfiguracji funkcji aktywacyjnych, odpowiedniego ustalenia początkowej wartości wag połączeń między neuronami, czy właściwej techniki przeprocesowania wejściowych danych. Ani alternatywne funkcje aktywacyjne, ani normalizacja danych wejściowych do przedziału <0; 1> (w celu zlikwidowania przepaści między parametrami binarnymi, jak płeć, oraz liczbowymi, jak stan konta), czy różne wartości współczynnika uczenia nie zdołały znacząco poprawić otrzymywanych rezultatów.

Nasze wątpliwości co do wiarygodności danych wejściowych oraz istnienia korelacji między parametrami wejściowymi, a przydzieloną konfiguracją, wzmógł artykuł na blogu *kaggle.com* [2]. Wpis ten opisuje różne podejścia z dziedziny sztucznej inteligencji mogące posłużyć do stworzenia skutecznego modelu predykcji dla danych użytych w naszym projekcie. Jak zauważył autor tego wpisu, sieć neuronowa radziła sobie fatalnie, uzyskując co najwyżej 80% skuteczności (zapewne też dominowały zera na wyjściu).

Reasumując, opracowany przez nas model uzyskiwał skuteczność w okolicach 75%, która może zadowalać, dopóki nie ustalimy, że jedna kategoria występuje w ponad 3/4 danych wejściowych.

Bibliografia

1. https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
2. https://www.kaggle.com/somaktukai/credit-card-default-model-comparison