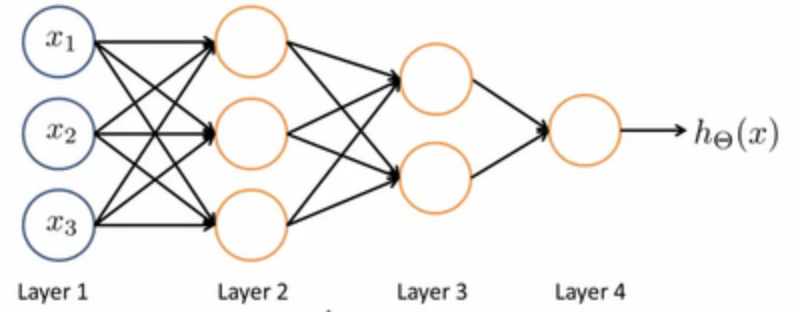
OPIS ALGORYTMU

Zastosowanym algorytmem klasyfikacji jest sieć neuronowa z propagacją wsteczną.

Sieć neuronową budujemy z neuronów - każdy z nich przyjmuje na wejściu kilka zmiennych, a na wyjściu daje jedną. Zmienna wyjściowa powstaje poprzez obliczenie kombinacji liniowej zmiennych wejściowych (wagi odpowiednich wejść są parametrami neuronu którymi możemy sterować). Wynik ten jest dodatkowo obłożony funkcją sigmoidalną, aby otrzymane wyjście znajdowało się w przedziale (0; 1).

Sieć neuronowa składa się z kilku rozłącznych warstw neuronów - wejścia neuronów danej warstwy połączone są z wyjściami neuronów warstwy poprzedniej.

Przykładowa sieć neuronowa:



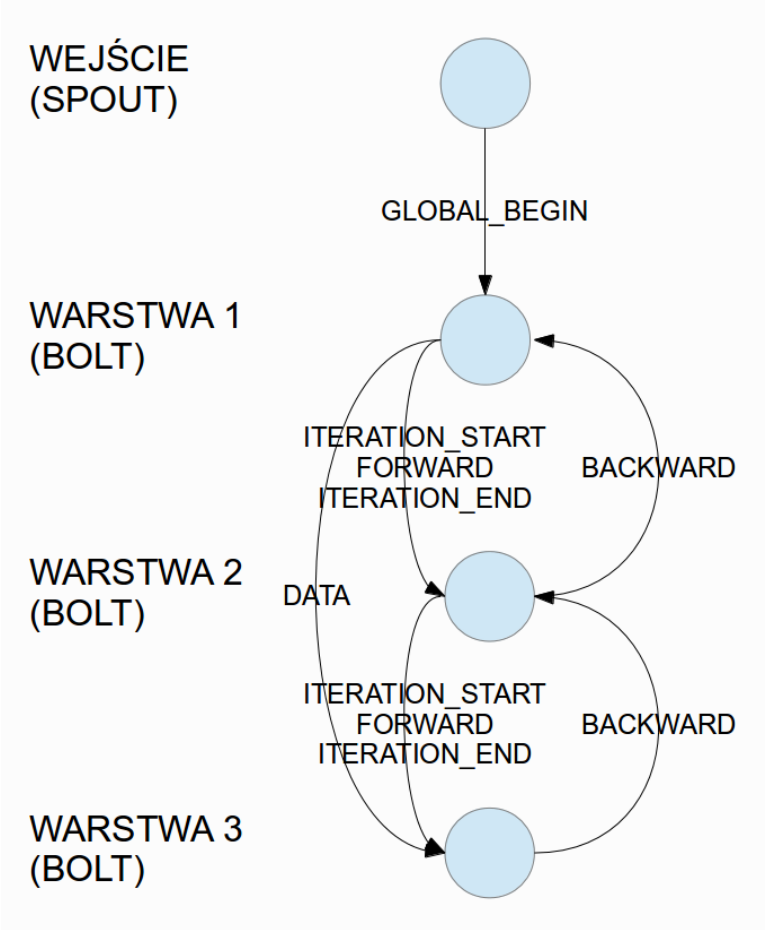
Wyjściem całego algorytmu jest wyjście ostatniego neuronu (w ogólnym przypadku może to być kilka neuronów, co pozwala na klasyfikację wieloklasową). Dzięki takiej budowie sieci, możemy skonstruować nieliniową hipotezę i rozwiązywać nieliniowe problemy.

Trenowanie sieci neuronowej odbywa się za pomocą algorytmu propagacji wstecznej - wykorzystujemy go obliczenia gradientu funkcji kosztu co prowadzi do jej minimalizacji (co najmniej do minimum lokalnego). W uproszczeniu, algorytm ten polega na policzeniu błędu na neuronach ostatniej warstwy, a następnie propagowanie go do warstw poprzednich biorąc pod uwagę wagi wejść neuronów. Mając błędy poszczególnych neuronów, możemy dla nich policzyć pochodne cząstkowe i odpowiednio zaktualizować wagi.

IMPLEMENTACJA NA PLATFORMIE STORM

W naszej implementacji, każda z warstw opakowana jest przez jednego bolta. Wejście, spout, na początku przesyła do warstwy ostatniej prawidłowe dane wyjściowe (co pozwala później obliczyć błąd dla neuronów ostatniej warstwy), a następnie przesyła do pierwszej warstwy informację o rozpoczęciu uczenia sieci. Po kolei, od pierwszej do ostatniej warstwy, przesyłane są dane prowadzące do propagacji do przodu (wyliczenia wyniku), a następnie, w drugą stronę, dane prowadzące do propagacji wstecznej.

Przepływ danych (przykład dla jednej warstwy ukrytej):



Ponieważ dane wyjściowe mogą być bardzo duże, dzielone są one na mniejsze porcje o stałym rozmiarze. Wagi są aktualizowane po przeprocesowaniu całej porcji - jest to pojedyncza iteracja algorytmu. W kolejnych iteracjach procesowane są kolejne porcje, a gdy dane się skończą, porcje przesyłane są ponownie od początku.

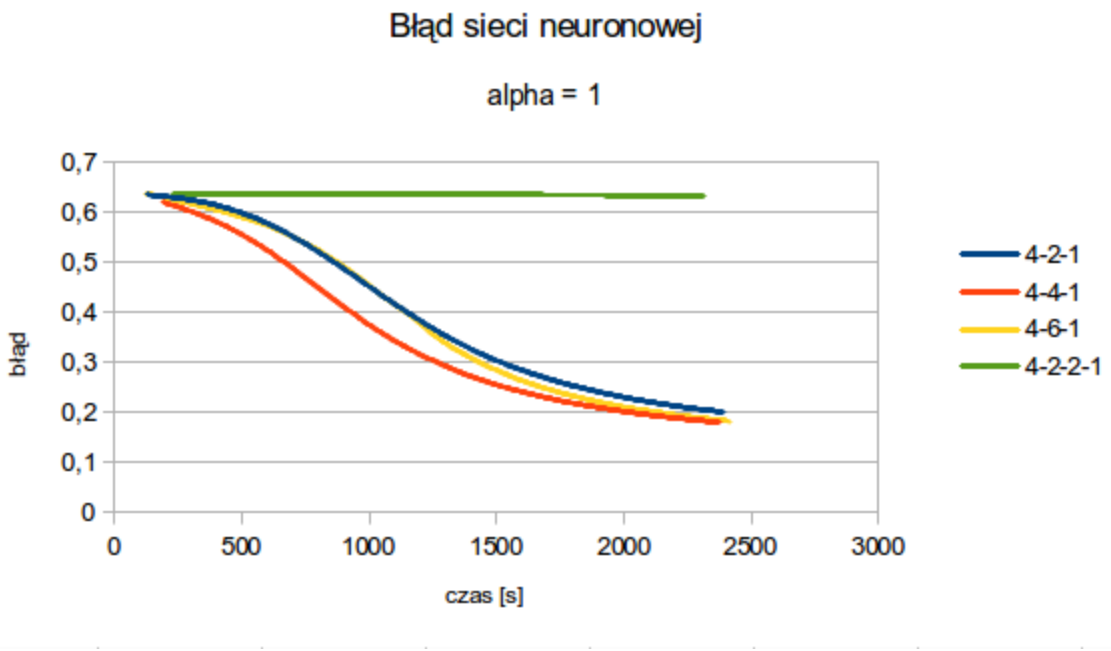
Kolejne kroki:
1. rozpoczęcie działania (GLOBAL_BEGIN)
2. przesłanie oczekiwanych danych wyjściowych dla danej porcji do ostatniej warstwy (DATA)
3. przesłanie informacji o rozpoczęciu iteracji (ITERATION_START)
4. postęp iteracji - cyklicznie *forward propagation* (FORWARD) i *backward propagation* (BACKWARD) dla wszystkich przykładów treningowych z danej porcji
5. przesłanie informacji o zakończeniu iteracji (ITERATION_END) prowadzące do aktualizacji wag neuronów
6. kolejne iteracje dokonują się przez powtarzanie kroków 2-5

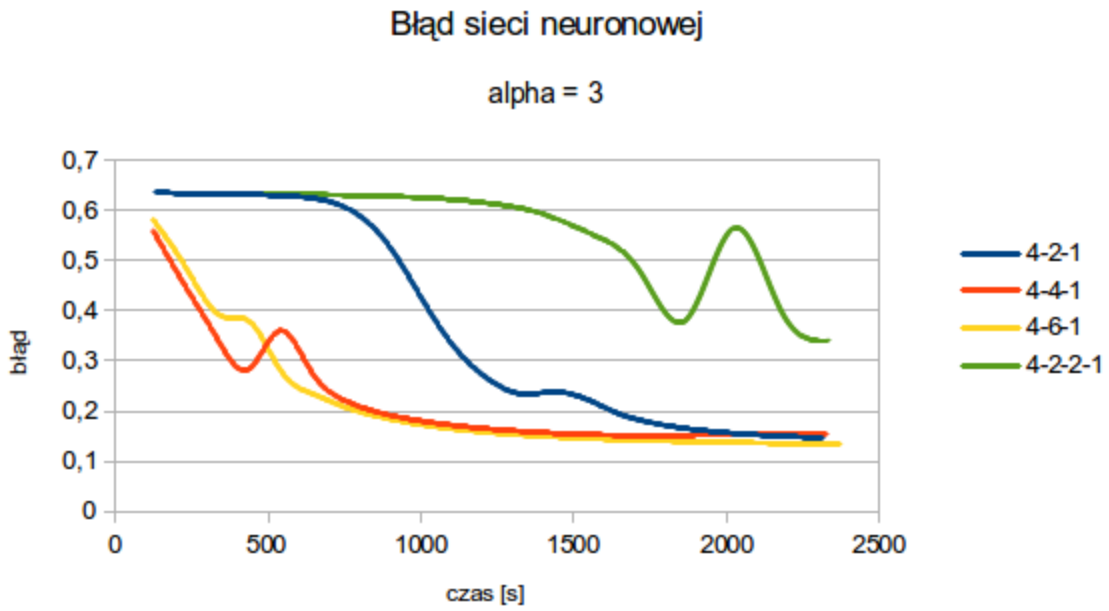
WYNIKI POMIARÓW I WNIOSKI

1. BŁĄD SIECI W ZALEŻNOŚCI OD CZASU DZIAŁANIA ALGORYTMU

Pomiary były dokonane dla dwóch wartości współczynnika *alpha* (*alpha* = 1 i *alpha* = 3) i czterech różnych architektur sieci:
a) jedna warstwa ukryta o rozmiarze 2
b) jedna warstwa ukryta o rozmiarze 4
c) jedna warstwa ukryta o rozmiarze 6
d) dwie warstwy ukryte, każda o rozmiarze 2

Wyniki nie są różnicowane ze względu na rozmiar danych, ponieważ właściwie nie ma on wpływu na sposób działania sieci. Wynika z tego, że dane i tak dzielone są na porcje o stałym rozmiarze i aktualizacja wag sieci następuje po przeprocesowaniu jednej takiej porcji. W związku z tym, nawet procesowanie nawet danych niemal nieskończonego rozmiaru działa niemalże tak samo jak procesowanie danych niewielkich, jeżeli tylko dane są odpowiednio strumieniowane (strumieniowane symulowane jest przez generator).

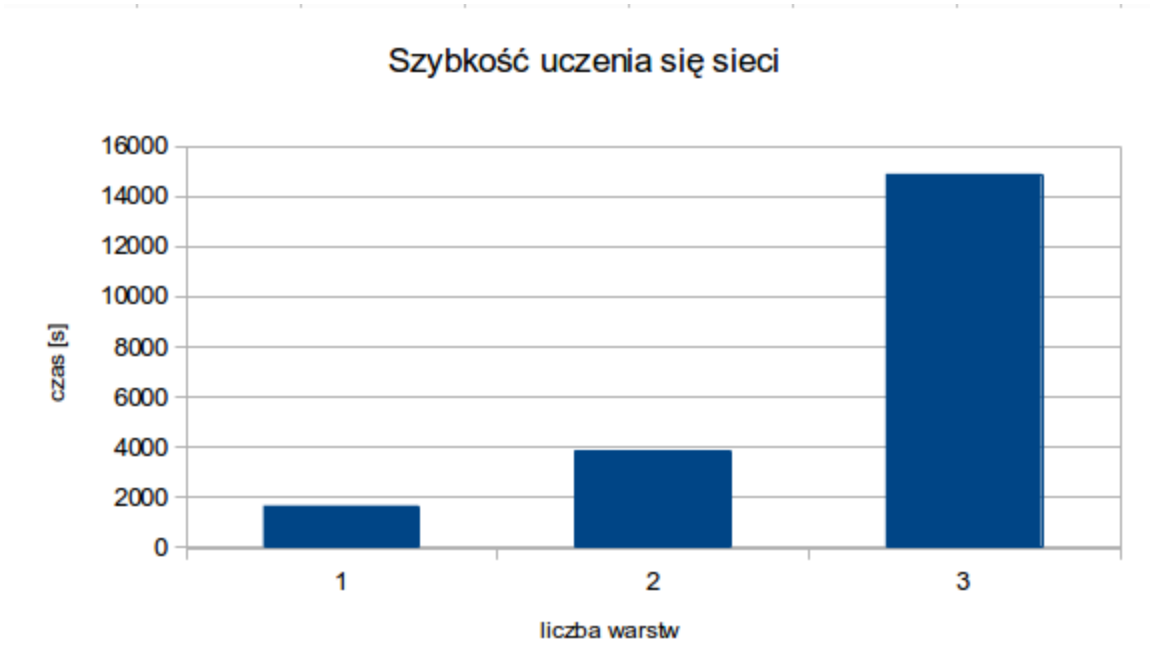




Jak widać, dla badanych danych najlepiej sprawdza się sieć neuronowa o jednej warstwie. Dla mniejszego *alpha* najlepsza okazuje się warstwa z czterema neuronami, natomiast dla większego warstwa z sześcioma neuronami jest minimalnie lepsza. Jednakże dla tak dużego parametru *alpha*, mimo ogólnie szybszej zbieżności algorytmu, widać już problemy wynikające ze zbyt dużego kroku w kierunku gradientu i “przeskakiwania” ekstremów lokalnych funkcji. Jeśli chodzi o dokładanie warstw, okazuje się to zupełnie nieefektywne. Dla małego *alpha* sieć uczy się niemalże niedostrzegalnie powoli, natomiast dla większego widać co prawda postępy, natomiast nadal jest znacznie gorzej niż w przypadku jednej warstwy.

2. SZYBKOŚĆ UCZENIA SIĘ SIECI W ZALEŻNOŚCI OD LICZBY WARSTW

W pomiarze tym każda warstwa ma 2 neurony, współczynnik *alpha* = 3. Błąd graniczny do którego sieć miała się zbiec został ustalony na 0,2.



Jak widać, w przypadku przy dokładaniu kolejnych warstw czas uczenia się sieci rośnie bardzo szybko dla testowanych danych, więc w tym przypadku nie ma sensu stosowanie więcej niż jednej warstwy.