**Dawid Matyasik PSI  
  
Przewidywanie wydajności procesorów komputerowych przy użyciu sieci neuronowych**

**Wstęp**

W tym eksperymencie pokażę jak sieć neuronowa może być użyteczna podczas rozwiązywania zagadnienia przybliżenia i przewidywania. Do wykonania ćwiczenia użyję Frameworka napisanego w języku Java o nazwie Neuroph.

Dane mi będzie również wykazać różnicę pomiędzy tym przybliżeniem a regresją liniową, która jest używana przy popularnym przewidywaniu wydajności CPU.

Przy przewidywaniu często możemy zauważyć, że np. przy użyciu regresji liniowej problem będzie zbyt skomplikowany lub wręcz niemożliwy do rozwiązania.

Sieci neuronowe nie są idealną metodą przewidywania zjawisk, ale ich niewątpliwą przewagą jest fakt, że automatycznie uczą się tylko dzięki danym wejściowym, bez potrzeby dodawania innych informacji podczas procesu nauki (takich jak rodzaj zależności w regresji).

**Wprowadzenie do problemu**

Celem ćwiczenia jest wytrenowanie sieci neuronowej w taki sposób, by przewidywała względną wydajność CPU przy użyciu danych wejściowych. Następnie porównam je z aktualnymi przewidywaniami, które są wyliczane przy użyciu regresji liniowej.

Dane wejściowe mogą być znalezione pod tym adresem: <http://archive.ics.uci.edu/ml/datasets/Computer+Hardware>

Zestaw danych zawiera 209 wpisów (instancji), a każdy zawiera 10 poniższych atrybutów:

1. nazwa producenta/sprzedawcy (String)
2. nazwa modelu (String)
3. czas cyklu maszyny w nanosekundach (int)
4. minimalna ilość pamięci w kilobajtach (int)
5. maksymalna ilość pamięci w kilobajtach (int)
6. ilość pamięci cache w kilobajtach (int)
7. minimalna ilość kanałów, którymi mogą być transmitowane informacje (int)
8. maksymalna ilość ww. kanałów (int)
9. względna wydajność układu (int)
10. względna wydajność, przybliżona i opublikowana (int)

Niestety, jak łatwo zauważyć klikając w powyższy link, nie możemy użyć tych danych bezpośrednio, przez ich różnice w jednostkach, rzędach wielkości etc. (mamy np. czas cyklu maszyny w nanosekundach z przedziału 17-1500, a liczba kanałów waha się od 0 do 52). Żeby możliwe stało się przeprowadzenie tych działań, trzeba skorzystać z normalizacji, o której za chwilę.

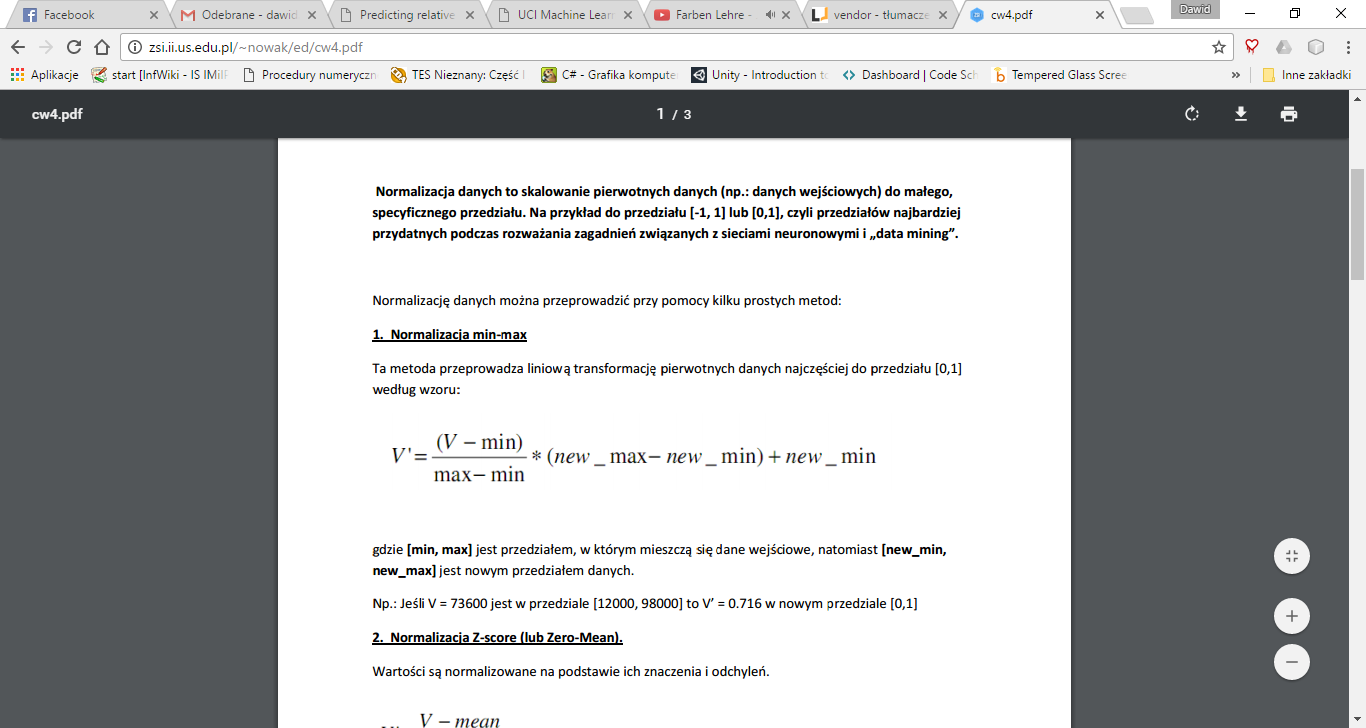
**Zastosowany algorytm trenowania sieci neuronowej:**

1. Normalizacja danych
2. Stworzenie projektu w Neuroph
3. Stworzenie zestawu treningowego
4. Stworzenie topologii sieci neuronowej
5. Wytrenowanie sieci
6. Przeprowadzenie testów, by mieć pewność, że sieć została poprawnie wytrenowana

**Proces normalizacji**

W tym ćwiczeniu użyjemy sposobu normalizacji metodą min-max.

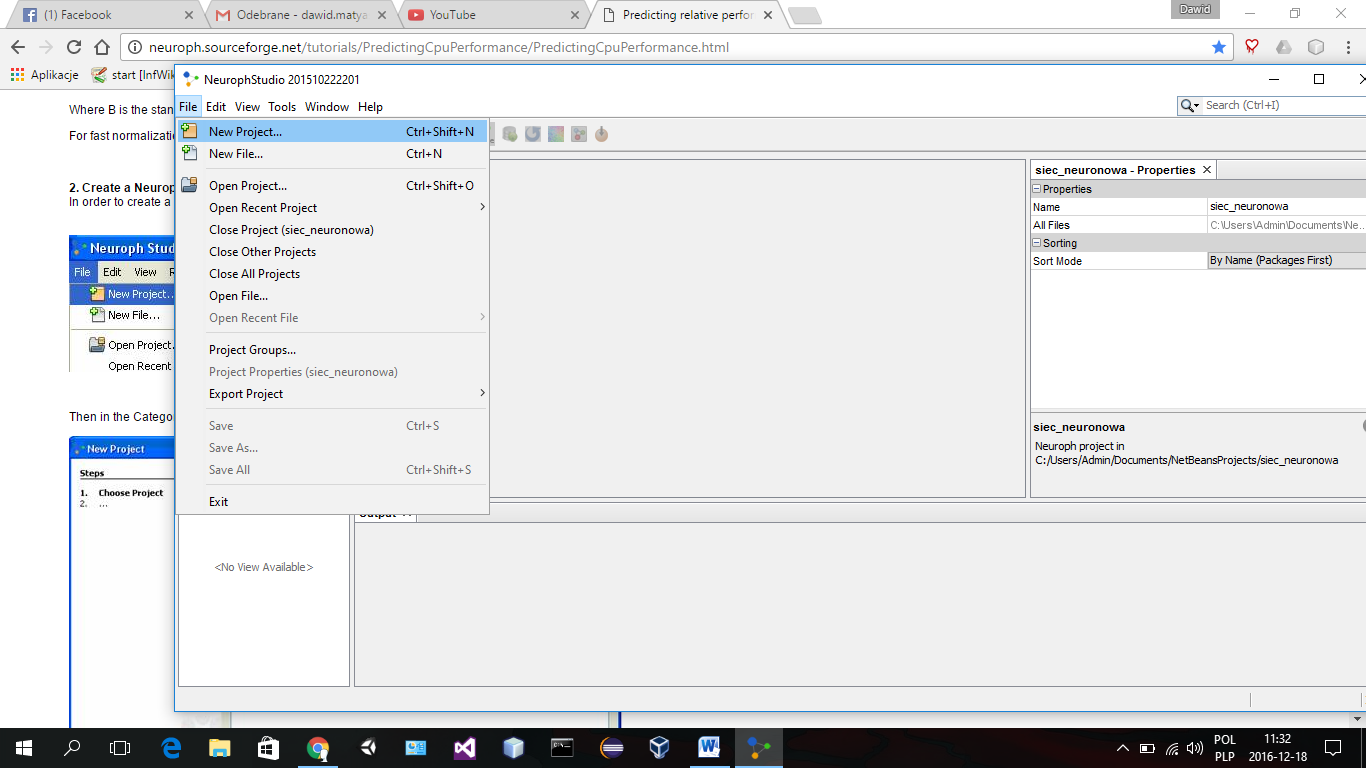
Ta metoda przeprowadza liniową transformację pierwotnych danych najczęściej do przedziału [0,1] według wzoru:



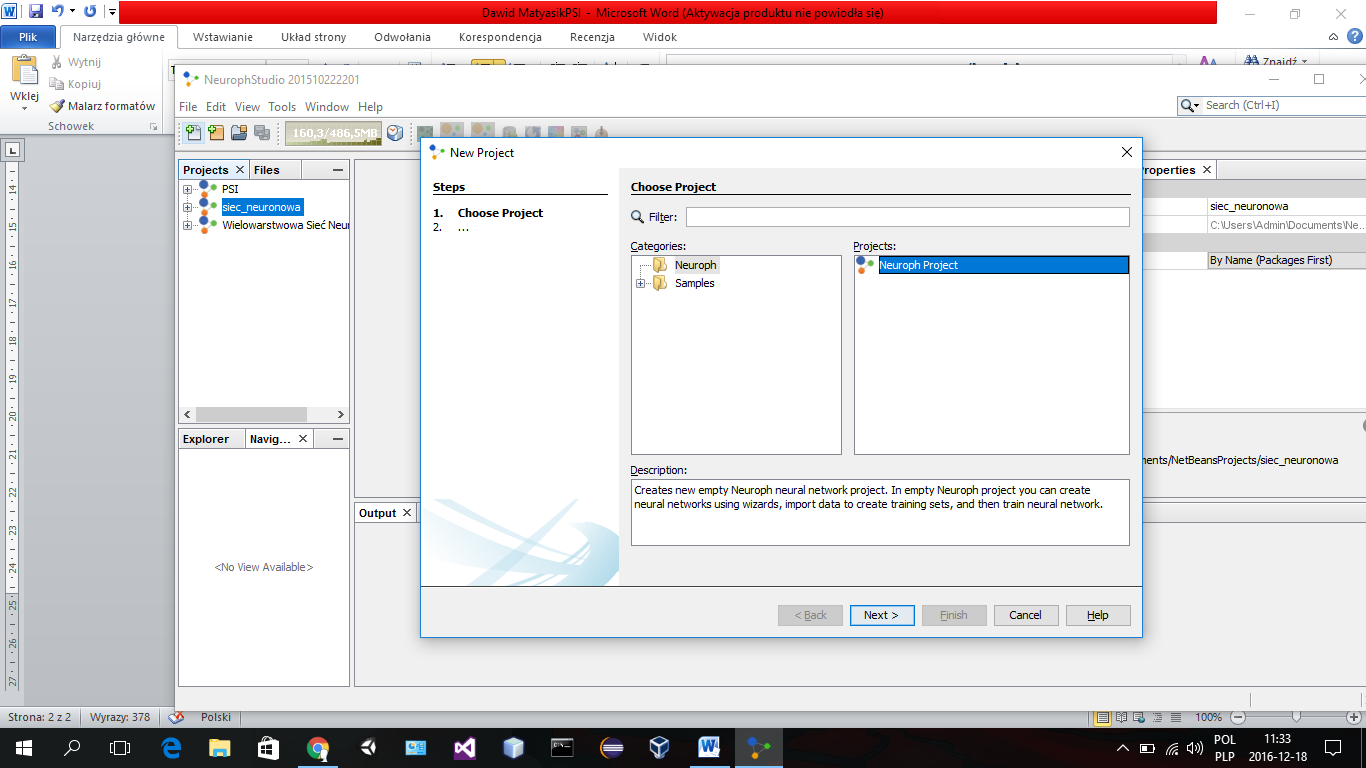
gdzie [min, max] jest przedziałem, w którym mieszczą się dane wejściowe, natomiast [new\_min, new\_max] jest nowym przedziałem danych.

**Tworzenie pierwszego projektu w Neuroph Studio**

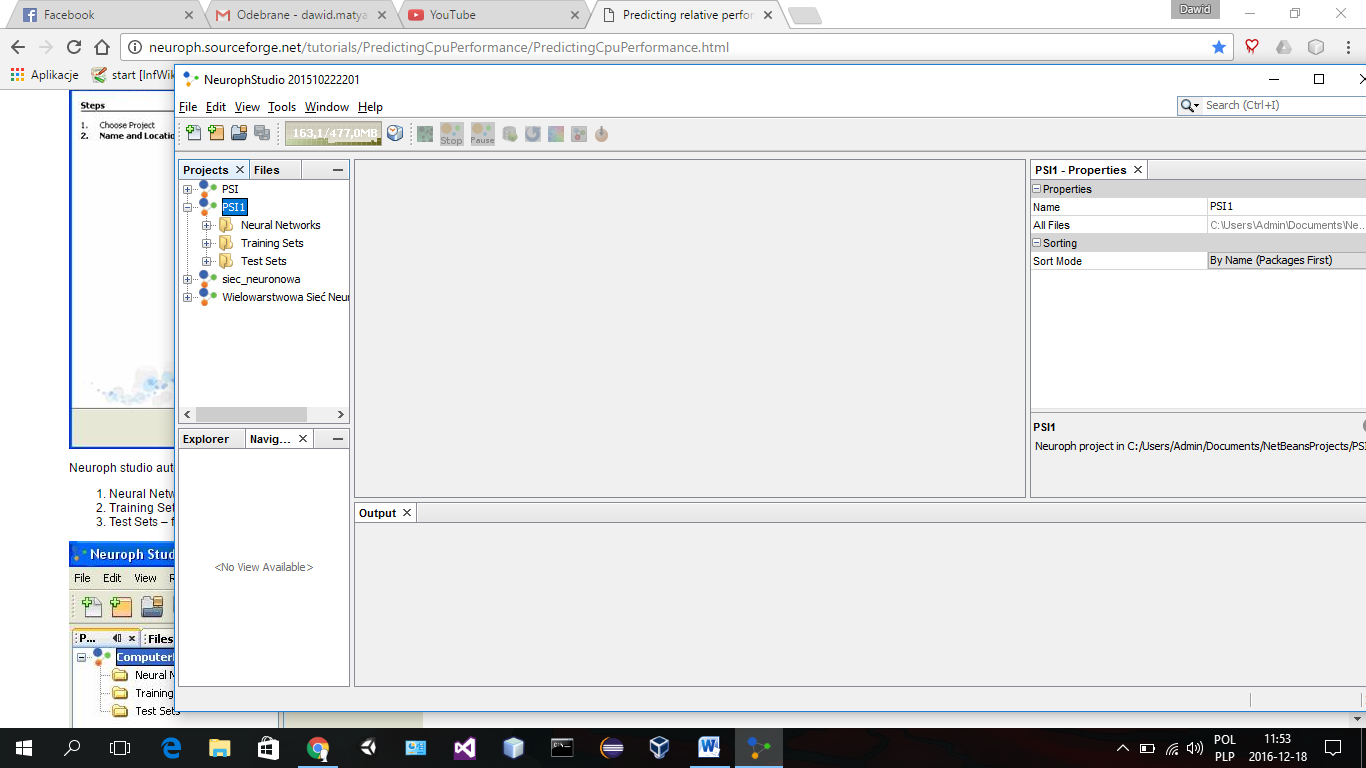
1. Wciskamy File->New Project…



1. Wybieramy Neuroph Project



1. Po utworzeniu projektu naszym oczom ukazuje się struktura projektu

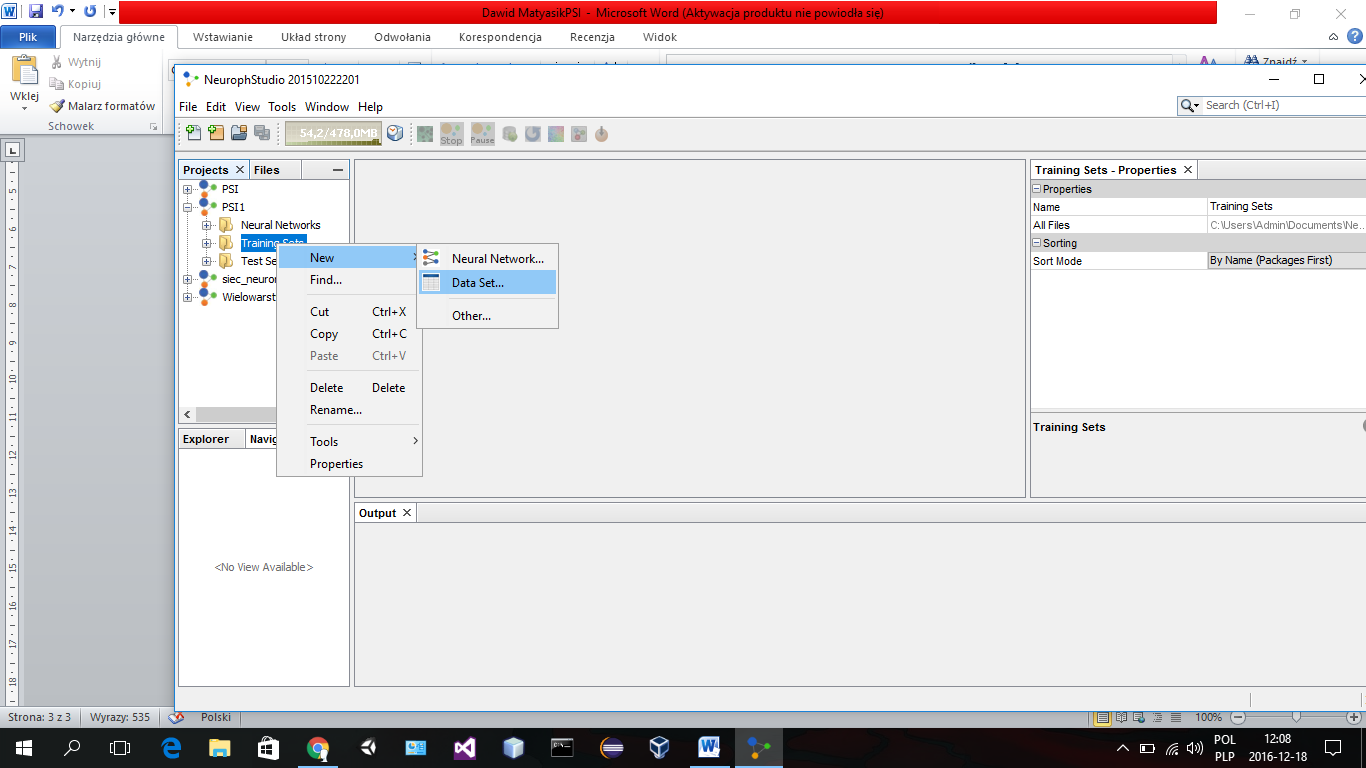


gdzie:  
Neural Networks – folder, w którym magazynujemy nasze sieci  
Training Sets – folder z naszymi danymi treningowymi

**Tworzenie zestawu treningowego**

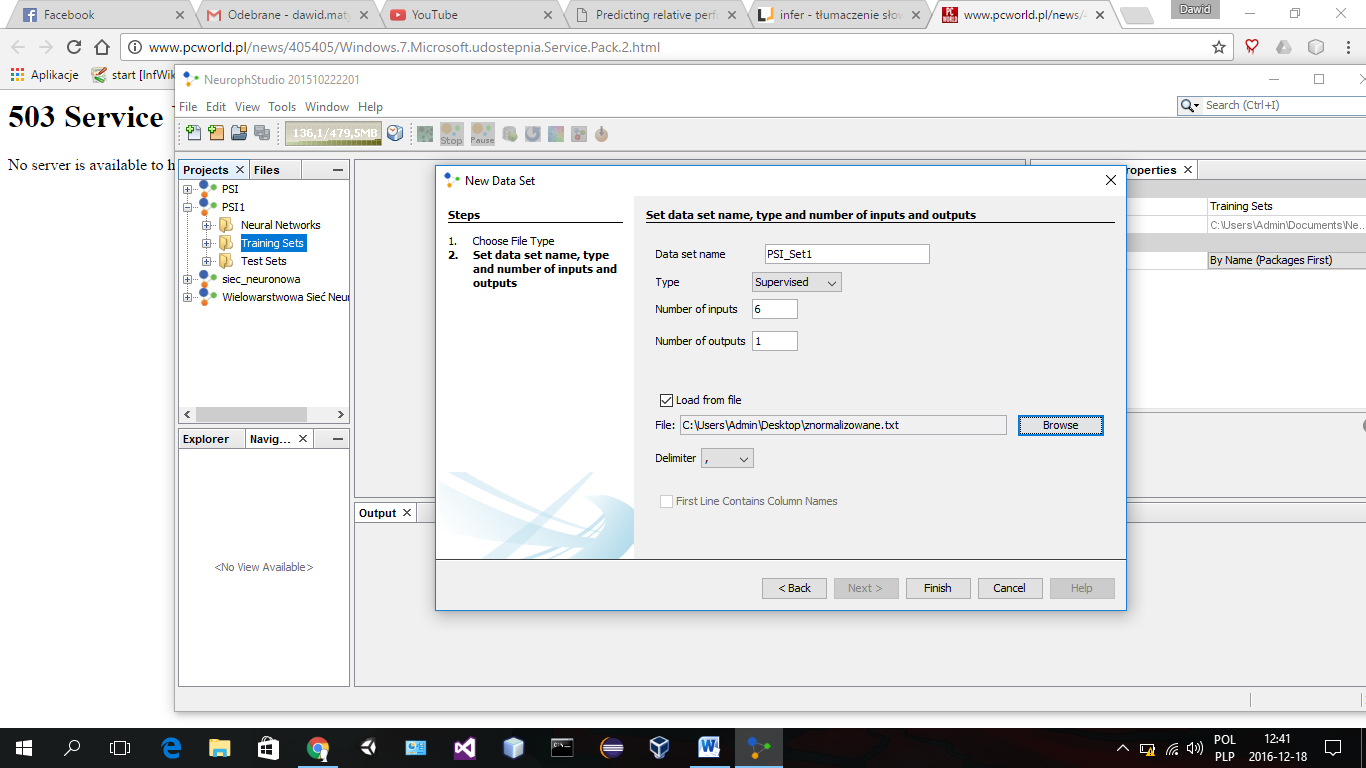
Następnym krokiem jest dodanie ww. zestawu do naszego projektu. Będzie to lista danych, które reprezentują wejścia i wyjścia używane przez sieć neuronową do wytrenowania. **Treningiem będę nazywał odnalezienie takich wag, które będą minimalizowały błąd pomiędzy oczekiwanym a rzeczywistym rezultatem.**

Po kliknięciu prawym przyciskiem myszy na Training Sets postępujemy następująco:



Ustalamy nazwę oraz typ zestawu treningowego. Możemy wybierać spośród dwóch: supervised (nadzorowany) oraz unsupervised (bez nadzorcy). Będę używał nadzorowanego zestawu, który zawiera przykłady dane na wejściu z połączonymi danymi wyjściowymi i uczy się jakie wnioski wyciągać z relacji pomiędzy nimi. Doskonale nadaje się do klasyfikacji, z czego skorzystamy.

Użyjemy 6 inputów i 1 outputu. Potem dodajemy ręcznie nasze znormalizowane dane testowe i klikamy finish.



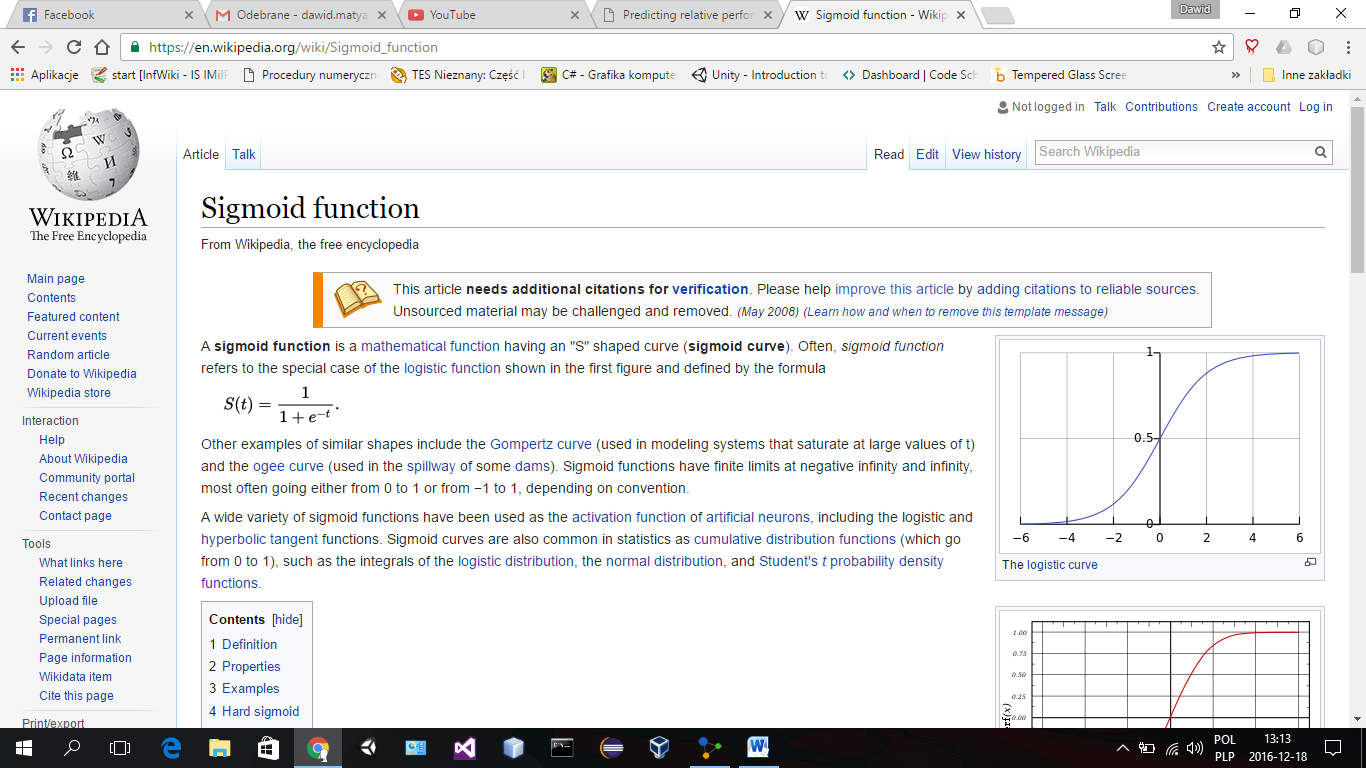
**Pierwsza próba wytrenowania sieci**

1. **Tworzenie sieci**

Na początku musimy stworzyć model sieci neuronowej. Prawym klikamy na Neutral Network, potem New->Neural Network.

Nasza sieć będzie wielowarstwowa, więc jako typ wybieramy: Multi Layer Perceptron. Właśnie takich sieci używa się, by rozwiązywać problemy klasyfikacji.

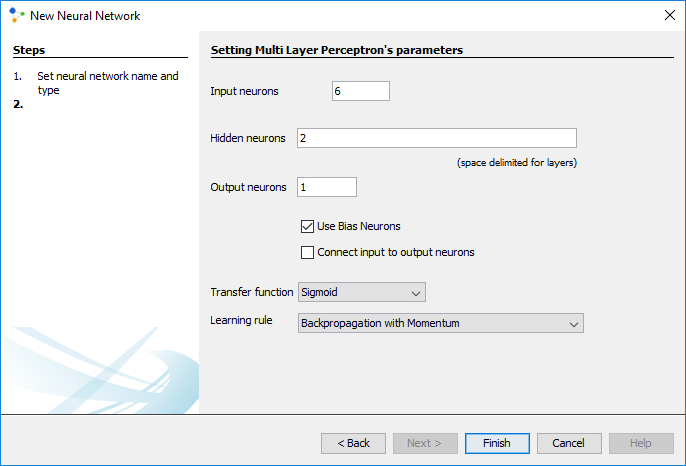
Po nazwaniu sieci pora na ustawienie parametrów. Będziemy używać 6 neuronów na wejściu i jednego na wyjściu, ale będziemy zmieniać liczbę neuronów w warstwie ukrytej. Jako naszej funkcji użyjemy funkcji o wzorze:



Jako zasady nauki użyjemy algorytmu wstecznej propagacji w wersji zmodyfikowanej, z tzw. momentum, czyli z wprowadzoną zmianą polegającą na wprowadzeniu do procesu uaktualniania wagi bezwładności. Dzięki tej modyfikacji można zwiększyć efektywność tempa uczenia bez straty stabilności procesu uczenia.

W pierwszej próbie użyjemy dwóch ukrytych neuronów, potem właśnie tym ustawieniem będziemy sterować, by uzyskać jak najmniejszy błąd przy jak najmniejszej ilości neuronów i – rzecz jasna – jak najszybciej.

Oczywiście, użyjemy nauronów z biasem.



1. **Trenowanie sieci**

Na początku musimy ustalić za pomocą którego zestawu danych treningowych będziemy trenować naszą sieć. Po prostu przeciągamy nasz zestaw na DataSet w schemacie sieci.

Teraz, po wciśnięciu przycisku Train, pojawi się okno, w którym możliwe stanie się ustalenie parametrów nauki. *Learning rate* musi być liczbą dodatnią, mniejszą niż 1. Zwykle, im jest ona większa, tym szybciej następuje trening. Ale należy pamiętać, że zbyt duża z kolei wartość może uniemożliwić sieci skończenie treningu. Na początek, na próbę ustawimy tę wartość jako 0.2.

W kwestii *momentum* – zwykle nadaje się mu wartość od 0.1 do 0.8. My zaczniemy od 0.8 właśnie.

Kiedy trening ma się skończyć? Możemy to określić na dwa sposoby, albo określimy limit iteracji, albo zatrzymamy naukę, gdy błąd osiągnie określony limit. My ustalimy jako cel 0.01.

Wynik, który uzyskałem (około 0,022797) zdecydowanie nie jest satysfakcjonujący – trzeba będzie pokombinować z innymi wartościami *Learning* i *momentum*.

**Podsumowanie pierwszych prób treningu**

Z takim modelem sieci nie udało mi się niestety osiągnąć wystarczająco dobrego wyniku. Poniżej tabela wartości i błędów, które uzyskiwałem dla różnych kombinacji ww. wartości, za każdym razem ustawiając losowe wartości wag.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer próby** | **Learning rate** | **Momentum** | **Liczba iteracji do zakończenia treningu** | **Błąd** |
| 1 | 0,2 | 0,8 | 17 | 0,022 |
| 2 | 0,2 | 0,7 | 30 | 0,018 |
| 3 | 0,3 | 0,6 | 33 | 0,022 |
| 4 | 0,4 | 0,8 | 11 | 0,017 |
| 5 | 0,1 | 0,8 | 53 | 0,016 |

**Kolejne próby – nowa sieć neuronowa**

Przyszła kolej na zmianę