Biologically Inspired Artifical Intelligence

Rozpoznawanie marek samochodów na podstawie ich log

Raport z wykonania projektu

Mateusz Boś

2013-06-26

Spis treści

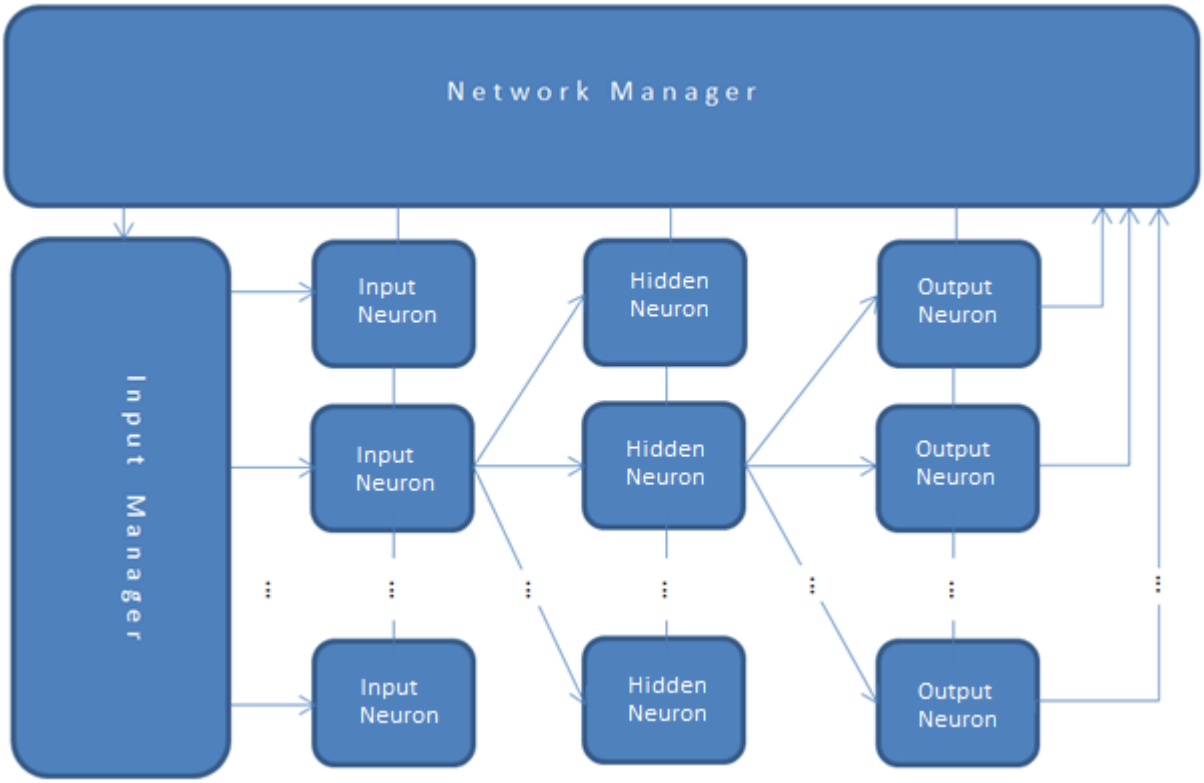
Założenia

Przyjęte założenia projektowe zgodnie z kartą projektu to:

1. Dane wejściowe w postaci plików bmp
2. Pliki wejściowe o rozdzielczości 200x200px
3. Rozpoznawanie minimum 5 wybranych marek
4. Algorytm uczenia przez propagację wsteczną
5. Użycie funkcji sigmoidalnej jako funkcji pobudzenia
6. Implementacja w języku JAVA

Przyjęta architektura sieci

Mój projekt postanowiłem wykonać od podstaw samemu. W tym celu zaprojektowałem i wykonałem sieć wielowarstwową, jednokierunkową, uczącą się algorytmem wstecznej propagacji. Poniżej widoczna jest przyjęta architektura sieci z pokazanymi tylko niektórymi, przykładowymi połączeniami.



Rysunek 1 Schemat blokowy architektury sieci

Cały projekt zgodnie z założeniami został wykonany w języku JAVA w wersji 1.7u21[4] z użyciem IDE IntelliJ Idea Community Edition[5]. Główną klasą w tym projekcie jest *NetworkManager* klasa ta zarządza uczeniem całej sieci jak również rozpoznawaniem oraz zarządcą danych wejściowych. Zarządca danych wejściowych *InputManager* zapewnia podział wejściowego obrazu i wyliczenie wartości wejściowych sieci dla podanego obrazu.

Wszystkie neurony muszą implementować interfejs *INeuron* dostarczający metod potrzebnych w celu obliczenia wartości neuronu, pobrania jego błędu. Klasą po której dziedziczą wszystkie neurony w

tym projekcie jest abstrakcyjna klasa *Neuron*. Poszczególne neurony przeciążają metody wyliczające błąd oraz aktualizujące wagi przyporządkowane neuronom wejściowym.

Przygotowanie danych wejściowych

Wersja pierwsza

Celem zmniejszenia wpływu szumów oraz kolorów obecnych na obrazie przed przystąpieniem do jego podziału oraz próbkowania postanowiłem wykonywać algorytm wykrywania krawędzi. Algorytmem który został wybrany był algorytm Canny ‘ego jako, że implementacja tego algorytmu nie była celem projektu skorzystałem z gotowej klasy dostarczanej przez Toma Gibara’ę[3]. Poniżej widoczne jest kilka par obrazów oryginalnych oraz przetworzonych, przygotowanych do przetworzenia przez sieć.

Wersja druga, ostateczna

Druga, ostateczna wersja przygotowania danych opiera się na dodatkowym założeniu, logo musi znajdować się na prawie idealnie białym tle. Algorytm przygotowania polega na tym, że kolor każdego piksela będącego poniżej wartości 0xF3 na jednym z kanałów zostaje zamieniany na czarny 0x00. Powoduje to jednak skupienie się czysto na kształcie loga a nie jego detalach, które mogą być uchwycone przy pierwszym sposobie przygotowania oraz wzroście podatności na szumy. Widać to dokładnie na poniższych obrazach.

Ponownie, poniżej widoczne jest kilka par obrazów oryginalnych oraz przetworzonych, przygotowanych do przetworzenia przez sieć.

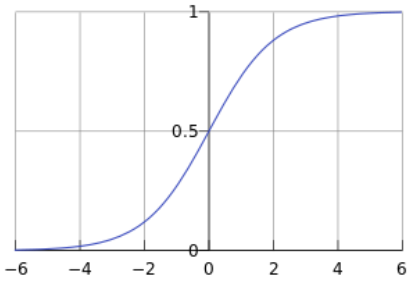
Proces uczenia

Obraną funkcją celu jest błąd średniokwadratowy:

$$E = \frac{1}{2}\sum_{i=1}^{M}(y_i - d_i)^2$$

Jako funkcję aktywacji neuronów zgodnie z założeniami wybrałem unipolarną funkcję sigmoidalną, której wzór widoczny jest poniżej:

$$S(t) = \frac{1}{1 + e^{-t}}$$

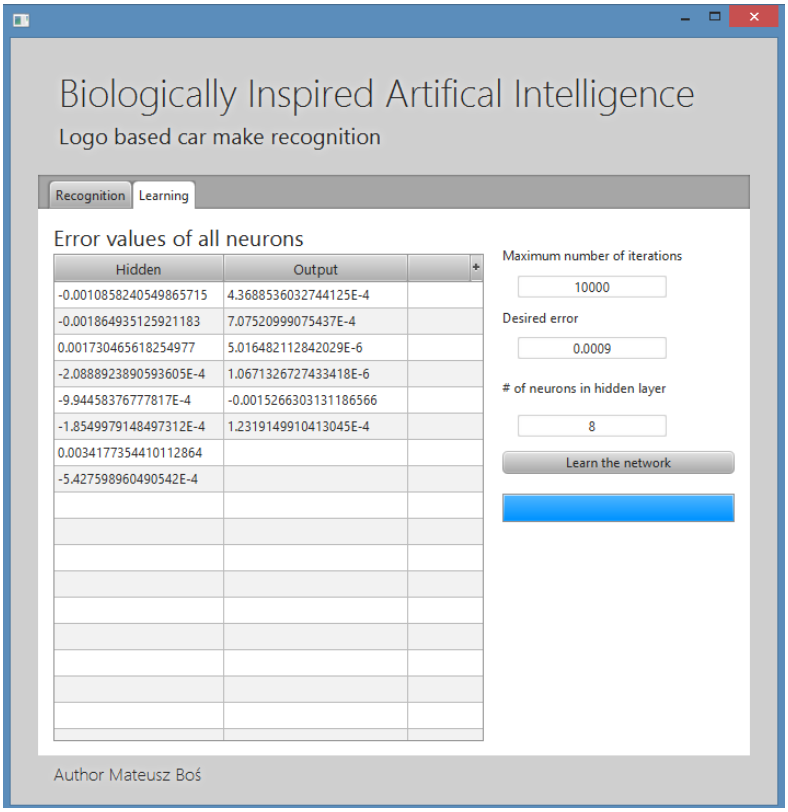


Wykres 1 Funkcja sigmodialna

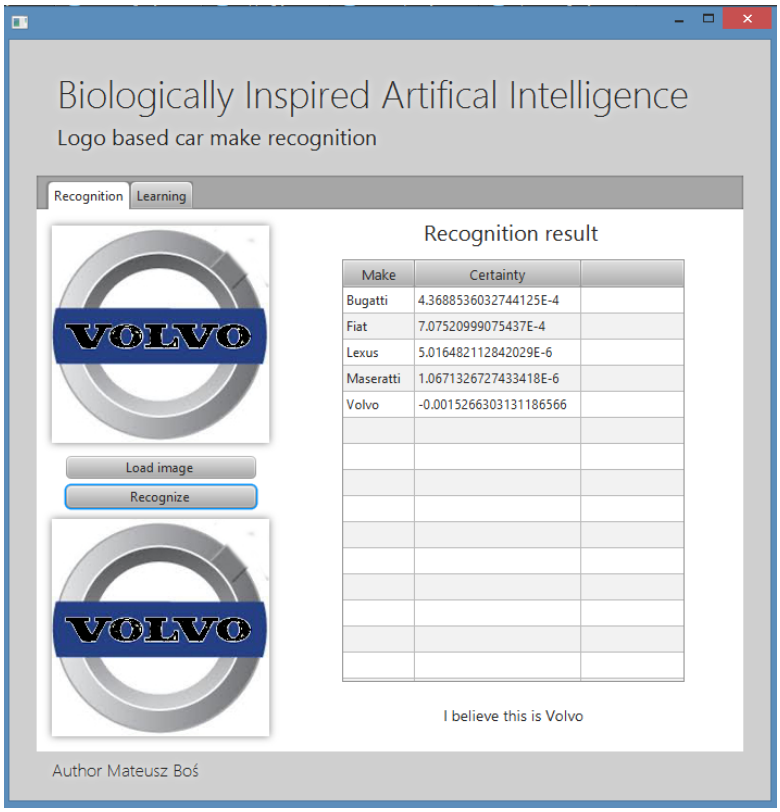
Uczenie polega na wyliczeniu wartości wszystkich neuronów sieci, zgodnie z kierunkiem przepływu sygnałów. Później należy zastąpić funkcje aktywacji jej pochodną i w zależności od różnicy między wcześniej obliczoną wartością wyjścia a żądaną wartością wyjścia i obliczyć wartości różnic wstecznych. W tym momencie należy uaktualnić wagi neuronów wejściowych(tutaj następuje właściwe uczenie sieci).

Interfejs użytkownika

Zaprojektowany interfejs użytkownika wygląda następująco:



Na widocznym obok zrzucie ekranu widać efekt rozpoznania, oprócz napisu informującego o tym jaka jest to najprawdopodobniej marka widoczna jest lista prezentująca marki i odpowiadające im błędy.



Na tym zrzucie widać efekt uczenia sieci w postaci błędów każdego z neuronów w sieci.

Napotkane problemy i spostrzeżenia

Początkowo sieć po kilku iteracjach traciła wartości wszystkich wag. Przyczyną okazało się wyliczanie ale nie przypisywanie nowych wag podczas procedury uczenia.

Kolejnym problemem, który niestety nie pozostał rozwiązany jest czas potrzebny na nauczenie sieci. W obecnej formie potrzeba 1600 neuronów wejściowych (obraz 200x200 pikseli podzielony na części 5x5 pikseli). Czas uczenia tak rozległej sieci jest nieakceptowalny. Zasadniczo sieć w przeciągu godziny oraz po 2,5 milionach iteracji nie zakończyła nauki z założonym maksymalnym błędem na poziomie 0,009.

W przypadku wcześniej przygotowanej testowej implementacji z czteroma neuronami wejściowymi nauka kończyła się po około 3 milionach iteracji z błędem na poziomie 0,0009. Pozwalało to na rozpoznanie jednego z 16 wejściowych wzorców (liczb z zakresu 0-15 w postaci binarnej, zakodowanych w postaci obrazów). Poniżej widoczne są błędy poszczególnych neuronów po procesie uczenia. Jak widać ostatni z neuronów sprawia problem z nauczeniem się, działo się tak za każdym razem gdy sieć się uczyła. Tej sytuacji nie poprawiło dodanie neuronu osiowego

0,000000000039320
0,000000000001871
0,000000000021334
0,000000000000009
0,000000000000001
0,000000000003328
0,000000000008365
0,000000000000113
0,000000000000000
0,000000000001933
0,000000000000011
0,000000000048141
0,000000000008689
0,000000000002744
0,000000000838001
0,049611082184390

Tabela 1 Błędy nauczonej, testowej sieci

Po implementacji pierwszego sposobu przygotowania danych wejściowych okazało się, że czas uczenia nie jest bliski nawet zadowalającemu. Sieć nie potrafiła nauczyć się z założoną dokładnością po nawet pięciu milionach(!) iteracji.
Doprowadziło to do wykonania drugiego sposobu przygotowania danych, który rozwiązał problem. Rozwiązanie to pociąga za sobą jednak pewne skutki jak wspomniane wcześniej założenie o tle log oraz zmniejszeniu wrażliwości a w zasadzie jej braku na detale zawarte w samym logu.

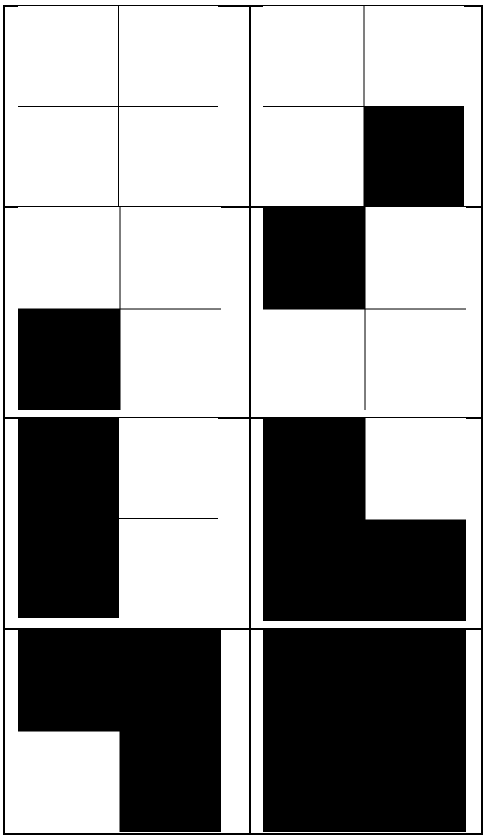


Tabela 2 Przykładowe dane testowe

Osiągnięte cele

Zgodnie z założeniami moja implementacja sieci neuronowej uczy się poprzez propagację wsteczną. Używa także funkcji sigmoidalnej jako funkcji aktywacji oraz jest zaimplementowana w języku JAVA. Pliki wejściowe zgodnie z założeniami muszą mieć wymiary 200x200 pikseli i obsługiwane formaty to między innymi bmp.

Po modyfikacji sposobu przygotowania danych udało się sprawić aby moja sieć rozpoznawała marki samochodów na podstawie ich log.

Wszystkie z wyznaczonych założeń/celi zostały osiągnięte.

Źródła

[1] Stanisław Osowski, *Sieci Neuronowe w ujęciu algorytmicznym,* WNT Warszawa 1996r.

[2] Nowoczesne Techniki Informatyczne, rozdział 5 [dostępne online 05-06-2013]
http://www.neurosoft.edu.pl/jbartman/NTI%20cwiczenie_5.pdf

[3] Klasa wykonująca algorytm Canny rozpoznawania krawędzi
http://www.tomgibara.com/computer-vision/canny-edge-detector

[4] Java JDK 7u21 http://www.oracle.com/technetwork/java/javase/downloads/index.html

[5] IntelliJ IDEA Community Edition http://www.jetbrains.com/idea/free_java_ide.html