Inżynieria Oprogramowania: Wzorce projektowe

Instrukcja laboratoryjna

Jakub Nalepa

Instytut Informatyki, Politechnika Śląska, Gliwice
jakub.nalepa@polsl.pl

1 Wprowadzenie

Celem ćwiczenia jest wykorzystanie w praktyce wzorca *Strategy* (*Strategia*)—należy przeprowadzić refaktoryzację kodu odziedziczonego do wzorca projektowego.

2 Zadanie do wykonania

W trakcie ćwiczenia laboratoryjnego należy zrefaktoryzować odziedziczony kod źródłowy[1] do wzorca projektowego *Strategia*. Wzorzec ten należy zastosować do wygodnej zmiany funkcji aktywacji w sztucznej sieci neuronowej (w obecnej implementacji dostępna jest wyłącznie sigmoidalna funkcja unipolarna $y(x) = \frac{1}{1+\exp^{-\beta x}}$, gdzie β jest stałą z zakresu $\beta \in (0, 1]$).

2.1 Sieć neuronowa

W niniejszym ćwiczeniu, sztuczna sieć neuronowa (trenowana za pomocą algorytmu wstecznej propagacji błędu) została wykorzystana do segmentacji ludzkiej skóry w obrazach barwnych—zbiór treningowy zawiera przykłady pikseli (znormalizowane wartości R, G i B) przedstawiających skórę i nie-skórę (zbiór danych można podejrzeć w folderze `Datasets`). Nasza sieć neuronowa składa się z czterech warstw:

1. Wartwa wejściowa: 3 neurony,
2. Pierwsza warstwa ukryta: 6 neuronów,
3. Druga warstwa ukryta: 3 neurony,
4. Warstwa wyjściowa: 1 neuron.

[1] Autorami kodu są Michał Myller (`michmyl139@student.polsl.pl`) oraz Szymon Piechaczek (`szympie318@student.polsl.pl`), studenci Makrokierunku na wydziale Automatyki, Elektroniki i Informatyki.

Neurony sąsiadujących ze sobą warstw są połączone ze sobą synapsami (wagami), które—podczas treningu sieci—są uaktualniane tak, aby z jak największą dokładnością odzwierciedlić zbiór treningowy (innymi słowy—staramy się znaleźć jak najlepszą funkcję przybliżającą zbiór treningowy, tj. „wejście" i „wyjście" przykładów z tego zbioru). Każdy neuron możemy traktować jak (prostą) jednostkę obliczeniową, której wyjściem jest suma ważona sygnałów wejściowych (w poniższym przykładzie, $n-1$ wag pomiędzy rozważanym neuronem, a neuronami z poprzedniej warstwy):

$$o = \sum_{i=0}^{n} w_i \cdot x_i = \mathbf{w}^T \mathbf{x}, \tag{1}$$

gdzie $x_0 = 1$ (zastosowanie takiego dodatkowego sygnału pozwala na „przesunięcie" progu aktywacji neuronu wzdłuż osi X, co w praktyce często ułatwia trening). Dodatkowo stosujemy funkcję aktywacji (liniową lub nieliniową) neuronu (kiedy neuron jest „aktywny", a kiedy nie):

$$y = f(o). \tag{2}$$

Przykładowe funkcje aktywacji (wraz z ich pochodnymi) zostały przedstawione w Tablicy 1.

Tablica 1. Przykładowe funkcje aktywacji (wraz z parametrem β).

Funkcja sigmoidalna	$f(x) = \frac{1}{1+\exp^{-\beta x}}$	$f'(x) = \beta \cdot (1 - f(x)) \cdot f(x)$
Tanh	$f(x) = \tanh(\beta \cdot x)$	$f'(x) = \beta \cdot (1 - f(x)^2)$
ArcTan	$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
ReLU	$f(x) = 0$ dla $x < 0$	$f'(x) = 0$ dla $x < 0$
	$f(x) = x$ dla $x \geq 0$	$f'(x) = 1$ dla $x \geq 0$

Która z tych funkcji okaże się „najlepsza" w naszym wypadku?

2.2 Algorytm genetyczny

W niniejszym ćwiczeniu, algorytm genetyczny został wykorzystany do doboru zbioru treningowego dla sztucznej sieci neuronowej. Każdy osobnik reprezentuje pewien podzbiór pełnego zbioru treningowego, a populacja takich zbiorów zredukowanych ewoluuje w czasie w poszukiwaniu jak najlepszego zbioru treningowego, tj. takiego, który pozwala na wytrenowanie jak najlepszego klasyfikatora (w naszym wypadku—sztucznej sieci neuronowej).

Kod algorytmu genetycznego znajduje się w folderze `GeneticAlgorithm`. W czasie tego ćwiczenia **nie będziemy modyfikować** algorytmu genetycznego (ale zachęcam do zapoznania się z kodem i bardzo chętnie odpowiem na pytania dotyczące algorytmów ewolucyjnych).

3 Środowisko testowe – szczegóły implementacyjne

3.1 Struktura pakietu

Pakiet po rozpakowaniu zawiera następujące elementy (foldery zaznaczone są **pogrubioną czcionką**):

- **src**: folder zawiera kod źródłowy;
  - **Datasets**: folder zawiera zbiór danych (piksele przedstawiające skórę i nie-skórę);
  - **Exceptions**: implementacja wyjątków;
  - **GeneticAlgorithm**: implementacja algorytmu genetycznego do doboru zbioru treningowego dla sztucznej sieci neuronowej;
  - **NeuralNetwork**: implementacja sztucznej sieci neuronowej (**uwaga:** pliki w tym folderze będą modyfikowane w trakcie trwania tego laboratorium);
  - IO-DesignPatterns-Labs.sln: plik *rozwiązania* (zawierający jeden projekt) Visual Studio;
- **instrukcja**: folder zawierający instrukcję do laboratorium.

3.2 Kod źródłowy

Po otwarciu solucji `IO-DesignPatterns-Labs` w Visual Studio, należy zwrócić uwagę na pliki `.h` i `.cpp` w folderach `NeuralNetwork`, zwłaszcza zaś na pliki `Neuron.h` oraz `Neuron.cpp`, w których znajduje się deklaracja i definicja funkcji sigmoidalnej i jej pochodnej (metoda `Derivative`). Algorytm wstecznej propagacji błędu został zaimplementowany w pliku `NeuralNet.cpp` (metody `BackPropagationForLastLayer` oraz `BackPropagationForHiddenLayers`)—ten plik także będzie modyfikowany.

3.3 Uruchomienie środowiska

Środowisko uruchamia się standardowo, zarówno w trybie `Debug` jak i `Release` (nie ma potrzeby definiowania dodatkowych parametrów wywołania programu).