

Kompresja obrazów za pomocą sieci neuronowych typu NEAT - raport

25 lutego 2015

Paweł Bielicki
Robert Jakubowski

1 Wstęp

Dokument przedstawia opis testów przeprowadzonych na programie stworzonym na podstawie dokumentacji wstępnej. W trakcie testów dokonano modyfikacji programu w celu przetestowania możliwości sieci oraz potencjalnemu polepszeniu wyników.

2 Parametry

Sieci neuronowe charakteryzują się dużą ilością parametrów. Charakterystyka topologii typu NEAT oraz specyfika problemu kompresji obrazów dodatkowo zwiększają tę ilość. Mnogość parametrów postanowiliśmy ograniczyć do tych, które będą według nas istotne:

- `inputLayerSize` - długość wektora wejściowego,
- `middleLayerSize` - długość wektora skompresowanego,
- `numberOfSpecies` - liczba osobników w populacji,
- `maxIteration` - maksymalna liczba iteracji (epok),
- `maxError` - maksymalny dopuszczalny błąd (program zostaje przerwany, gdy błąd sieci jest mniejszy niż `maxError`),
- `mutationRatio` - współczynnik mutacji,
- `crossoverRatio` - współczynnik krzyżowania.

W powyższym zestawieniu istnieje tylko jeden współczynnik do mutacji, który określa z jakim prawdopodobieństwem mutowany będzie osobnik. Zgodnie z specyfikacją sieć mogła się mutować na kilka sposobów (usuwanie wierzchołka, dodawanie wierzchołka, dodanie połączenia, usunięcie połączenia, zmiana wagi połączenia). Każdy ze sposobów mutacji miał określone własne prawdopodobieństwo ustawione w wyniku przetestowania różnych wariantów, ostatecznie przyjęliśmy następujące wartości:

- dodanie połączenia - 0.25,
- dodanie neuronu - 0.03,
- usunięcie połączenia - 0.02,
- zmiana wagi połączenia - 0.67,

- usunięcie neuronu - 0.03.

Do łatwego definiowania nowych testów wykorzystaliśmy narzędzie Gradle.

3 Wstępne testy i dokonane modyfikacje

Pierwsza wersja programu dawała wyniki dalekie od oczekiwań. Dla prostych obrazów wyniki były bardzo niedokładne. Nawet dla obrazu, który składał się z 2 pasów - czarnego i białego - sieć uczyła się dobrze tylko czarnego koloru, natomiast drugi pasek był jasnoszarym szumem. Dla bardziej skomplikowanych obrazów otrzymywaliśmy obraz, który był szarym szumem. W celu poprawienia wyników zastosowaliśmy następujące zmiany w programie:

1. Usunęliśmy ograniczenie na wagi - mogą przyjmować dowolne wartości, zamiast początkowych wartości z przedziału $[0,1]$.
2. Funkcja aktywacji została zamieniona na liniową.
3. Zmienione zostało parsowanie wyniku sieci na obraz - usunęliśmy dzielenie wartości przez liczbę połączeń neuronu wyjściowego.
4. Powyższe zmiany spowodowały, że wyjście mogło być dowolną liczbą (wcześniej musiała to być liczba z przedziału $[0,1]$). Zmiana polegała na tym, że wartości spoza przedziału $[0,1]$ są rzutowane na jego ograniczenia.

Zmiany 1-4 spowodowały, że sieć zaczęła wyraźnie zbiegać do oczekiwanych rezultatów. Zauważyliśmy jednak możliwość kolejnych modyfikacji, ponieważ obrazy wyjściowe, w początkowych iteracjach, były białe. Aby temu zaradzić wprowadziliśmy kolejne zmiany:

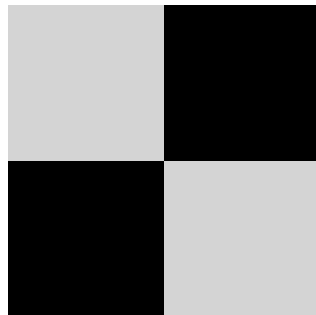
5. Problem znajdował się w początkowym losowaniu wartości wag. Zmieniliśmy je tak, aby wartością oczekiwaną losowania były wartości, które w efekcie działania sieci dają uśrednianie obrazu. Implementacja polegała na tym, że dla danego połączenia prowadzącego z neuronu A do neuronu B, losowana jest liczba z zakresu $[0,2]$, a następnie dzielona jest przez liczbę połączeń wejściowych neuronu B. Tak otrzymana liczba staje się wagą tego połączenia.

6. Przetestowaliśmy również dobór/zmianę wag z pomocą rozkładu normalnego. Spowodowało, to uśrednienie wartości wyjściowych, przez co dla wejścia/wyjścia o rozmiarze 4 dostajemy dość dobre obrazy wizualne, aczkolwiek po przybliżeniu, widać uśrednienie.
7. Następnie postanowiliśmy poprawić zauważone uśrednianie wyników. Założyliśmy, że to ze względu na sposób liczenia błędu - była to suma kwadratów błędów dla każdego piksela. Ustawiliśmy błąd na prostą sumę błędów, ale to sprawiło, że sieci jeszcze mniej chętniej chciały wychodzić z uśredniania. Spróbowaliśmy więc zastosować liczenie błędu jako sumy czwartych potęg błędów. W ten sposób w początkowych etapach uczenia widać więcej artefaktów, ale sieć szybciej dochodzi do wyszukiwania szczegółów.

4 Testy

W tym rozdziale opisane są wykonane testy. W podpisie każdego z obrazów są umieszczony parametry uruchomienia.

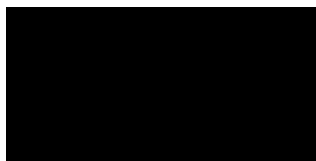
W pierwszych testach sprawdziliśmy trywialny przypadek z 4 pikselami: Wynik okazał się identyczny z obrazem wejściowym. Program zakończył swo-



Rysunek 1: inputLayerSize = 4; middleLayerSize = 1; numberOfSpecies = 50; maxIteration = 1000; maxError = 0.000017; mutationRatio = 0.6; crossoverRatio = 0.1

je działania po kilkunastu iteracjach, gdy błąd przekroczył granicę maxError.

Następnie przetestowaliśmy poniższy obraz:



Rysunek 2: Obraz wejściowy złożony z dwóch pasków: czarnego i białego o rozmiarze 8x8

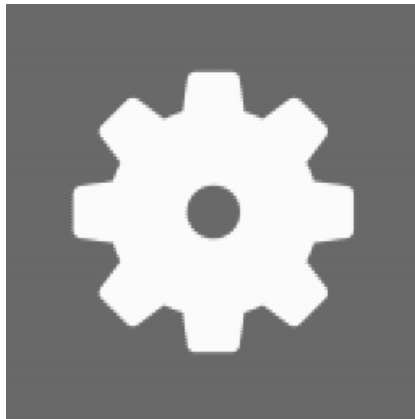


Rysunek 3: Obraz wyjściowy. Parametry: `inputLayerSize = 64`; `middleLayerSize = 4`; `numberOfSpecies = 100`; `maxIteration = 5000`; `maxError = 0.00001`; `mutationRatio = 0.9`; `crossoverRatio = 0.1`

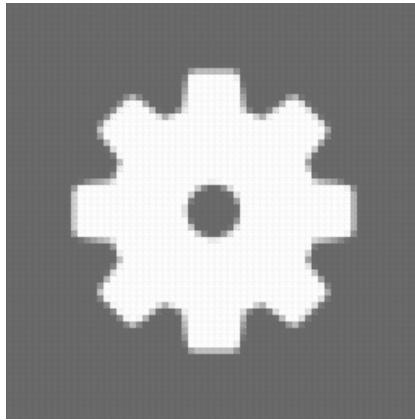
Program zakończył się po kilku tysiącach iteracji przekraczając granicę błędu. Mimo małej wartości parametru `maxError` widać delikatne artefakty. Następne testy przeprowadziliśmy z krótszymi wektorami wejściowymi. Jako obraz testowy wzięliśmy zębatkę:



Rysunek 4: Obraz wejściowy



Rysunek 5: Obraz wyjściowy. Parametry: $\text{inputLayerSize} = 4$; $\text{middleLayerSize} = 2$; $\text{numberOfSpecies} = 100$; $\text{maxIteration} = 500$; $\text{maxError} = 1$; $\text{mutationRatio} = 0.5$; $\text{crossoverRatio} = 0.1$



Rysunek 6: Obraz wyjściowy. Parametry: $\text{inputLayerSize} = 16$; $\text{middleLayerSize} = 2$; $\text{numberOfSpecies} = 100$; $\text{maxIteration} = 800$; $\text{maxError} = 1$; $\text{mutationRatio} = 0.9$; $\text{crossoverRatio} = 0.1$

Dla obrazu z $\text{inputLayerSize}=4$ program się skończył, ponieważ granica błędu została przekroczona, natomiast dla drugiego przypadku (z 16 neuronami wejściowymi) po 800 iteracjach błąd wynosił 13.

5 Podsumowanie

Wyniki testów pokazują, że założony w dokumentacji wstępnej rozmiar wektora wejściowego (64) jest użyteczny tylko dla prostych obrazków. Nawet testy z użyciem wektora o długości 16 dały umiarkowane wyniki. Widać z kolei, że testy, w których długość wektora wejściowego jest równa 4 dają dobre rezultaty - nawet dla skomplikowanych obrazów. Warto też zauważyć, że dla tych obrazów w warstwie pośredniej był tylko 1 neuron. Według naszych przewidywań sieć byłaby w stanie się nauczyć trudnych obrazów również dla sieci o większych rozmiarach, ale wymagałoby to dużo dłuższego czasu uczenia.