Przetwarzanie obrazu za pomoca konwolucyjnej sieci neuronowej - wyniki pracy

Józef Jasek

January 22, 2019

1 Wstep teoretyczny

1.1 Przyjete oznaczenia

Rozpatrujac pojedyncza warstwe przyjmiemy nastepujace oznaczenia:
X - wejściowa macierz danych
k - liczba przykładów w wejściowej macierzy danych
m - liczba neuronów warstwy poprzedniej
n - liczba neuronów warstwy obecnej
A - macierz wyników otrzymana po przepuszczeniu X przez warstwe
Y - wyznaczone pochodne z poprzedniej warstwy lub w przypadku funkcji kosztu poprawne etykiety danych wejściowych

1.2 Podstawa matematyczna propagacji wstecznej

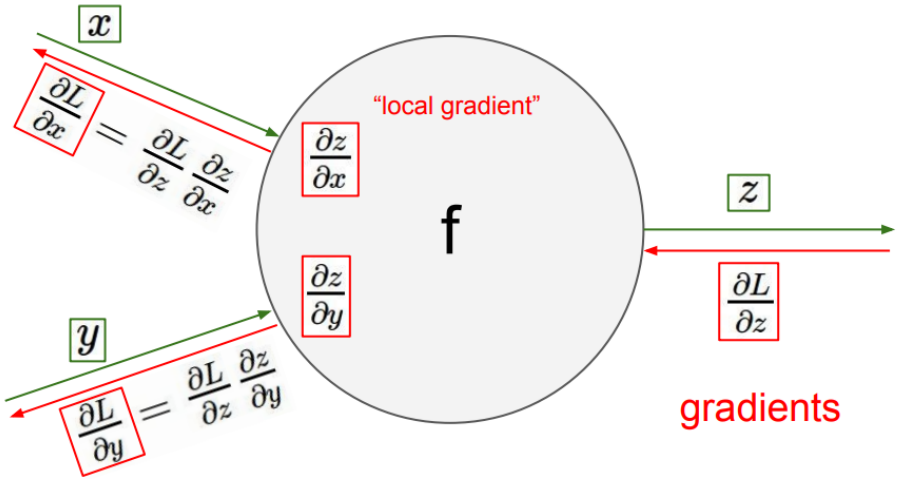

Figure 1: Funkcja wyjścia i pochodne funkcji kosztu po funkcjach wejścia w pojedynczym neuronie

Do wyznaczenia pochodnych funkcji kosztu wzgledem wag wykorzystamy zależność

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}$$

Stad możemy wyznaczyć pochodne wszystkich wag jako

$$\frac{\partial L}{\partial \omega_n}$$

$$\frac{\partial L}{\partial \omega_{n-1}} = \frac{\partial L}{\partial x_n} \cdot \frac{\partial x_n}{\partial \omega_{n-1}}$$

$$\frac{\partial L}{\partial \omega_{n-2}} = \frac{\partial L}{\partial x_n} \cdot \frac{\partial x_n}{\partial x_{n-1}} \cdot \frac{\partial x_{n-1}}{\partial \omega_{n-2}}$$

$$\dots$$

$$\frac{\partial L}{\partial \omega_1} = \frac{\partial L}{\partial x_n} \cdot \dots \cdot \frac{\partial x_2}{\partial \omega_1}$$

1.3 Funkcja kosztu

Wykorzystywana funkcja kosztu jest

$$L = \frac{-1}{mk} \sum_{m,k} (Y_{m,k} \cdot log(A_{m,k} + (1 - Y) \cdot log(1 - A_{m,k}))$$

lub w postaci macierzowej

$$L = -Avg(Y \cdot log(A) + (1 - Y)log(1 - A))$$

Możemy teraz wyznaczyć niezbedne pochodne do algorytmu wstecznej propagacji

$$\frac{\partial L}{\partial A} = \frac{-1}{mk}\left(\frac{Y}{A} - \frac{1-Y}{1-A}\right) = \frac{-1}{mk}\left(\frac{Y(1-A) - A(1-Y)}{A(1-A)}\right) = \frac{-1}{mk}\left(\frac{Y-A}{A-A^2}\right)$$

Ponieważ mk jest stała możemy ja pominać rekompensujac odpowiednio zmienionym hiperparametrem α. Otrzymujemy wiec

$$\frac{\partial L}{\partial A} = -\frac{Y-A}{A-A^2}$$

1.4 Sigmoida

Sigmoida dana jest wzorem

$$A = \frac{1}{1+e^{-x}}$$

Wzór używany w propagacji wstecznej to

$$\frac{\partial A}{\partial X} = \frac{e^{-x}}{(1+e^{-x})^2}$$

1.5 Softmax

Funkcja softmax dana jest wzorem

$$A_j = \frac{e^{X_j}}{\sum_{i=1}^{m} e^{X_i}}$$

Wzór używany w propagacji wstecznej to

$$\frac{\partial A_i}{\partial X_j} = \begin{cases} A_i(1 - A_j) & \text{dla } i = j \\ -A_i A_j & \text{dla } i \neq j \end{cases}$$

1.6 Konwolucja

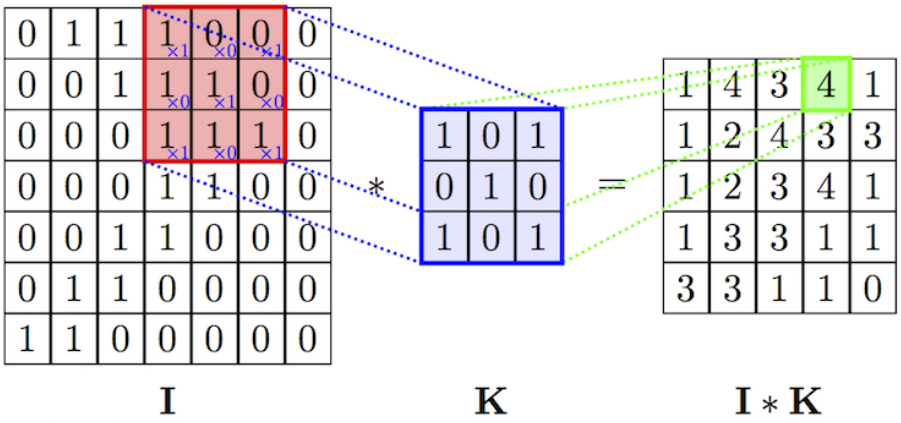

Figure 2: Warstwa konwolucyjna

W przypadku warstwy konwolucyjnej zamiast po prostu mnożyć element wejściowy przed wage, wykorzystujemy filtry małej wielkości, którymi wielokrotnie przejeżdżamy po naszym wejściu i dla każdego fragmentu stosujemy mnożenie macierzy po elementach. Każdy filtr generuje jedno dwuwymiarowe przekształcenie danych wejściowych. Po obliczeniu konwolucji transformujemy wynik funkcja sigmoidalna wg wzoru jak wyżej.

1.7 Maxpool

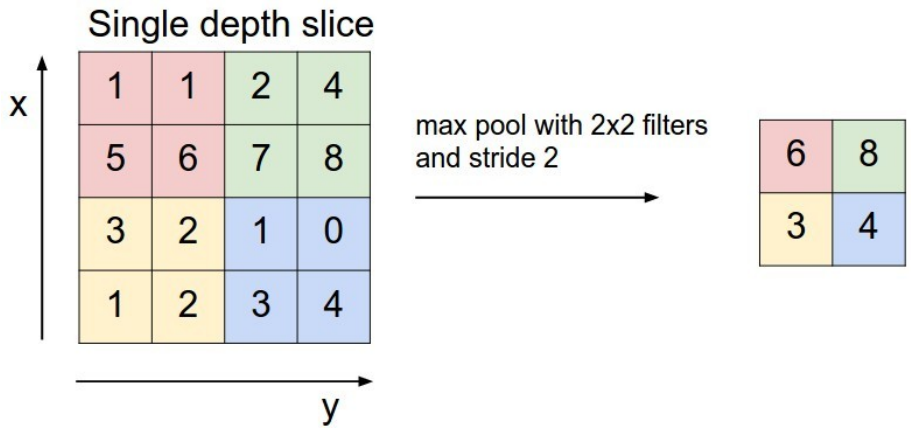

Figure 3: Warstwa maxpool

Warstwa maxpool działa jak warstwa konwolucyjna, ale zamiast nakładać filtr wybiera element maksymalny z danego fragmentu i podaje go na wyjściu. Przy propagacji wstecznej ten element, który został wybrany ma pochodna równa pochodnej wyjścia, a pozostałe pochodne sa równe 0.

2 Model sieci

Wstepnie zaproponowany model sieci przedstawia sie nastepujaco

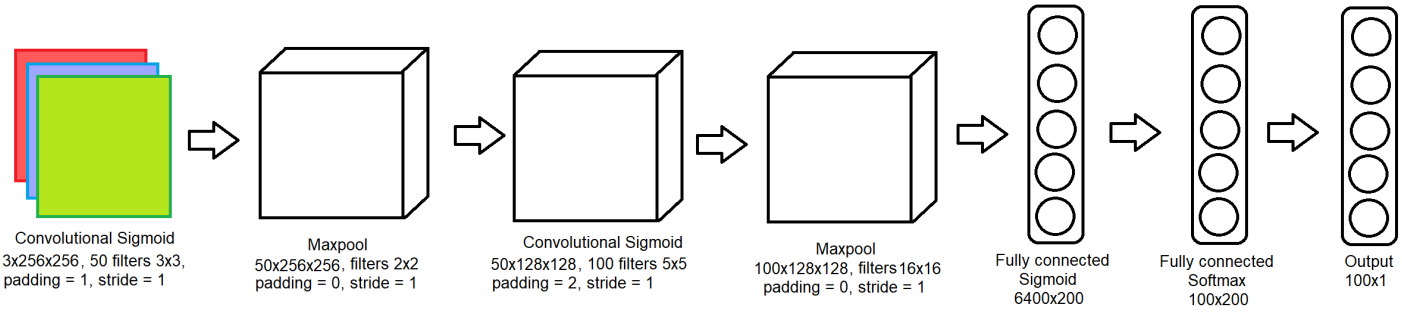

Figure 4: Proponowany model sieci

3 Implementacja

W ramach realizacji projektu zaimplementowano w C++ nastepujace elementy uczenia maszynowego

- implementacja dwuwymiarowych macierzy (podstawowe operacje matematyczne, operacje per element, transpozycja, zapisywanie do strumienia danych, inicjalizacja wartości wg rozkładu Gaussa, funkcja sigmoidalna, eksponenta, logarytm per element, suma po wybranych wymiarach)
- warstwa fully-connected sieci neuronowej nakładajaca funkcje sigmoidalna z operacjami propagacji wprzód i wstecz, inicjalizacja wartości wg rozkładu Gaussa, aktualizacja wag na podstawie propagacji wstecz, zapisywanie wag do strumienia danych)
- warstwa softmax ze wszystkimi elementami jak wyżej
- warstwa maxpool i konwolucyjna ze wszystkimi elementami jak wyżej oraz dodatkowo operacja transformacji 4-wymiarowego inputu/outputu

warstwy konwolucyjnej na 2-wymiarowy input/output warstwy w pełni połaczonej

- testy jednostkowe do wszystkich wyżej wymienionych elementów

Dodatkowo w ramach obróbki analizowanych zdjeć zaimplementowano w jezyku Python

- generator obrazów zawierajacych słowa z listy top 100 słów w jezyku angielskim z możliwościa transformacji przez obrót, skalowanie, zmiana koloru czcionki i tła, zmiana rodziny czcionki i jej rozmiaru
- program do wyświetlania i analizy wyjść warstwy konwolucyjnej

4 Realizacja

Po zaimplementowaniu sieci do trenowania użyliśmy 10 najpopularniejszych słów zamiast 100 ze wzgledu na duże narzuty pamieciowe, które uniemożliwiły trenowanie wiekszej liczby słów na naszym modelu. Nasza próbka danych treningowych wyniosła 180 obrazków. Z uwagi na fakt, że nie udało sie zrealizować projektu do końca, nie zdażyliśmy stworzyć zbioru danych testowych. Trenowanie trwało 150 epok. Po trenowaniu i przetestowaniu outputu sieci neuronowej mogliśmy zauważyć, że zwraca ona 10% prawdopodobieństwo przynależności do każdej klasy dla każdego obrazka.

5 Podsumowanie

Niestety przez zastosowanie funkcji sigmoidalnej w naszej sieci doszło do tzw. problemu znikajacego gradientu. Już przy czwartej warstwie gradient zwracany przez propagacje wsteczna był na poziomie błedu reprezentacji zmiennoprzecinkowej skutecznie uniemożliwiajac poprawne wytrenowanie sieci. Przypuszczamy, że użycie np. warstwy ReLU (Rectified Linear Unit) cechujacej sie stałym gradientem dla wartości dodatnich pozwoliłoby zniwelować problem, niestety zabrakło czasu na jej implementacje.

6 Bibliografia

- `https://en.wikipedia.org/wiki/Vanishing_gradient_problem`
- `https://bit.ly/2s6lZto`
- `https://www.researchgate.net`
- `https://www.youtube.com/watch?v=d14TUNcbn1k`
- `https://www.youtube.com/watch?v=bNb2fEVKeEo`

- https://www.coursera.org/learn/machine-learning
- https://www.coursera.org/specializations/deep-learning