SIECI NEURONOWE – ćwiczenie 3

Ćwiczenie 3 obejmuje algorytm propagacji wstecznej, który powinien być znany już z wykładu. Na wykładzie podane zostały wzory na poszczególne parametry na różnych warstwach sieci neuronowych i ich wyprowadzenie, na laboratoriach zajmiemy się implementacją. Propagacja wsteczna jest po prostu zastosowaniem reguły łańcuchowej:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

W wyliczaniu gradientu kosztu modelu po parametrach. Dla użytej konwencji notacji:

- Operujemy na wektorach i mnożeniu macierzowym, kolejność ma znaczenie
- L jest skalarem, w notacji gdzie x to wektor kolumnowy, $\frac{\partial L}{\partial x}$ jest wektorem o wymiarze odpowiadającym x^T (aby zgadzało się mnożenie macierzowe)
- $\frac{\partial y}{\partial x}$ to macierz pochodnych cząstkowych wszystkich wyjść po wszystkich wejściach o wymiarach (dim_y, dim_x)

(Istnieje też konwencja odwrotna, jeżeli natkniesz się na nią szukając materiałów, tutaj ściągawka: https://en.wikipedia.org/wiki/Matrix_calculus#Layout_conventions)

Jak ta matematyka przekłada się na implementację? Weźmy przykład dowolnej funkcji element-wise, tzn. takiej która aplikuje przekształcenie niezależnie od siebie do każdego elementu wektora np.

$$y = f(x) = x^2$$

Dla skalarów wiemy, że pochodna z tej funkcji to *f'(x) = 2x* czyli jeśli nałożymy na nią kolejną funkcję, tak że *h(x) = g(f(x)),* wtedy *h'(x) = g'(x)2x*.

Dla wektorów powinniśmy mieć jako $\frac{\partial y}{\partial x}$ macierz kwadratową (dim_x, dim_x) i wykorzystać ją w regule łańcuchowej, ale zauważmy, że wszystkie elementy tej macierzy poza diagonalą będą zerowe, natomiast diagonala to po prostu elementy wektora 2x.

Pamiętajmy też że wyliczanie odpowiednich wartości wyjściowych z każdej operacji musi odbyć się w kolejności tych operacji, natomiast wyliczanie gradientu po konkretnych wejściach – w kolejności odwrotnej. Stąd jeżeli chcemy zaimplementować operację: podniesienie wektora do kwadratu, w praktyce będziemy to robić mniej więcej tak:

```
// x.shape=(x,1), żeby zgadzać się z opisaną wyżej konwencją
// zapisu. W backward() potrzebna będzie transpozycja
def forward(x):
     cache_x = x
     return x*x
```

```
// derivative_y.shape=(1,x)
def backward(derivative_y):
     return derivative_y*2*cache_x.transpose()
```

Istotne jest tutaj że:

- Przy przejściu w przód, musimy zapisywać wartość wektora x na potrzeby późniejszego przejścia po operacjach wstecz
- Możemy zaimplementować pochodną/gradient jako funkcje `backward`, która na wejściu przyjmuje wyliczone do tej pory $\frac{\partial L}{\partial y}$, zaś na wyjściu daje $\frac{\partial L}{\partial x}$
- `backward` to nie zawsze musi być pełnym mnożeniem przez $\frac{\partial y}{\partial x}$, np. wyżej przemnożenie przez macierz diagonalną jest zastąpione numpy'owym operatorem *, czyli mnożeniem element-wise wektorów. Efekt jest ten sam, możemy skorzystać z uproszczenia
- W obrębie `backward`, jeżeli korzystamy z uczonych parametrów modelu, powinniśmy wyliczyć i zapisać gradient – pochodne cząstkowe po tych parametrach
- Jeżeli każda operacja jaką wykorzystujemy ma zaimplementowane te dwie funkcje, możemy bez problemu zbudować dowolną ich sekwencję i wyliczyć gradienty na dowolnym poziomie tej sekwencji

Zadaniem na dwa kolejne laboratoria jest zbudowanie modelu wielowarstwowego sieci neuronowej z dowolną funkcją aktywacji i funkcja kosztu taką, jak dla regresji logistycznej. Następnie należy przebadać jak model zachowuje się na zbiorze heart disease dla:

- Różnej wymiarowości warstwy ukrytej
- Różnej wartości współczynnika uczenia
- Różnych odchyleń standardowych przy inicjalizacji wag
- Danych znormalizowanych i nieznormalizowanych
- Różnej liczby warstw

Ćwiczenie oceniane jest w skali 0-20 pkt.