Metody głębokiego uczenia, projekt nr 1

Własna implementacja algorytmu wstecznej propagacji błędu w perceptronie wielowarstwowym (MLP)

Tymoteusz Makowski Olaf Skrabacz

19 marca 2019

Opis zadania

Celem projektu była implementacja perceptronu wielowarstwowego (ang. *multilayer perceptron*) z szeregiem wymaganych funkcjonalności takich jak:

- wybór liczby warstw oraz liczby neuronów ukrytych w każdej warstwie,
- wybór funkcji aktywacji,
- możliwość ustawienia:
  - liczby iteracji,
  - wartości współczynnika nauki (ang. *learning rate*),
  - wartości współczynnika bezwładności,
- możliwość zastosowania sieci zarówno do klasyfikacji, jak i do regresji.

Implementacja

Do wykonania zadania projektowego wybraliśmy język programowania Python3 i skorzystaliśmy z jego możliwości obiektowych.

Funkcje aktywacji

Zaimplementowaliśmy wiele funkcji aktywacji, które można wybierać dla poszczególnych warstw. Oprócz funkcji liniowej zaimplementowaliśmy:

ReLU (Rectified Linear Unit)

$$\mathrm{relu}(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{1}$$

Funkcja sigmoidalna

$$\mathrm{sigmoid}(x) = \frac{\mathrm{e}^x}{1 + \mathrm{e}^x} \tag{2}$$

Funkcja *tanh*

$$\tanh(x) = \frac{2}{1 + \mathrm{e}^{-2x}} - 1 \tag{3}$$

Funkcja wektorowa *softmax*

$$\mathrm{softmax}\big((x_i)_{i=1}^n\big) = \left(\frac{\mathrm{e}^{x_i}}{\sum_{j=1}^n \mathrm{e}^{x_j}}\right)_{i=1}^n \tag{4}$$

Funkcje straty

W projekcie są do wyboru dwa sposoby obliczania strat. Jest to błąd średniokwadratowy (ang. *mean squared error*) oraz entropia krzyżowa (ang. *cross entropy*). Pierwsza metoda jest wykorzystywana do regresji, zaś druga do klasyfikacji.

Klasa warstwy `Layer`

Podczas tworzenia każdej z warstw podajemy następujące parametry:

- liczba neuronów, którą ma zawierać ta warstwa,
- liczba neuronów poprzedniej warstwy albo, w przypadku pierwszej warstwy, wymiar danych wejściowych,
- jedna z funkcji aktywacji wymienionych powyżej.

Przykład tworzenia warstwy o 3 neuronach, gdzie dane wejściowe mają dwa wymiary (albo poprzednia warstwa ma dwa neurony), a funkcją aktywacji jest funkcja sigmoidalna:

```
Layer(3, 2, "sigmoid")
```

Klasa `Layer` nie zawiera metod, które są wykorzystywane z perspektywy użytkownika.

Klasa sieci `NeuralNetwork`

Konstruktor klasy `NeuralNetwork` przyjmuje następujące parametry:

- rodzaj funkcji błędu,
- wartość współczynnika bezwładności.

Klasa ta zawiera dwie główne metody – `add` oraz `train`, które służą do, odpowiednio, dodawania warstwy do sieci i ćwiczenia sieci. Funkcja `train`, oprócz nauki, zwraca na koniec wartości funkcji straty na zbiorze treningowym w kolejnych etapach procesu uczenia.

Przykład budowy i uczenia, sieci dwuwarstwowej o liczbie neuronów, kolejno, 1 i 2, do klasyfikacji zbioru na płaszczyźnie.

```
nn = NeuralNetwork("cross_entropy", momentum=0)
nn.add(Layer(1, 2, "relu"))
nn.add(Layer(2, 2, "softmax"))
nn.train(X=train_set_X, Y=train_set_y, epochs=30, learning_rate=0.01)
```

Gdzie `train_set_X` i `train_set_y` to dane treningowe, `epochs` to liczba iteracji uczenia, a `learning_rate` to współczynnik nauki.

Analiza działania sieci

Regresja

Testy

...

Wnioski

...

Klasyfikacja

Testy

...

Wnioski

...

Podsumowanie

...