Dokumentacja Projektu
Języki Skryptowe
Klasyfikator ręcznego pisma

Chłąd Paweł

2019
Grudzień

1 Opis działania

1.1 Wstęp

Główny program składa się z serwera HTTP, którego zadaniem jest obsługa użytkownika. Drugą częścią jest strona internetowa która stanowi interfejs pomiędzy serwerem a użytkownikiem. Trzecią częścią jest program uczący, za jego pomocą została wytrenowana sieć neuralna, która potrafi rozpoznawać napisane przez użytkownika liczby. Sprawność sieci wynosi 95% ale ze względu na brak normalizacji danych wejściowych, procent ten będzie niższy dla prawdziwych (nieznormalizowanych) danych.

1.2 Instrukcja Obsługi

Program uruchamiamy przez Launch.bat albo server.exe. Launch.bat przedstawi nam dodatkowe opcje:

- Start server - uruchomi server.exe
- Backup - utworzy kopię sieci neuralnej i zebranych zdjęć
- Exit - wyjdzie z pliku wsadowego

2 Strona techniczna

2.1 Struktura

Serwer HTTP oraz program uczący zostały zrealizowane w języku Python (ver. 3.7.5). Strona internetowa w języku markupowym HTML oraz skrypty w języku JavaScript.

Wszelkie pliki źródłowe znajdują się pod adresem: https://github.com/Madoxen/HandwritingMLService
Lista użytych bibliotek:

- numpy
- PIL (Python Imaging Library)

Program ma następującą strukturę

- bin - folder zawierający pliki wykonywalne
- src - folder zawierający repozytorium git
- data - folder zawierający dane do nauki
- backup - folder zawierający kopię zapasową danych do nauki i repozytorium git

2.2 Opis kodu

2.2.1 server.py

Moduł python zawierający klasę Server. Klasa ta odpowiada za uruchamianie i wstrzymywanie faktycznego serwera HTTP (http.server) oraz jego handler'a (http_handler).

Klasa Server zawiera następujące właściwości:

- addr - tuple zawierający adres i port serwera
- server - instancja serwera http
- server_thread - instancja wątku serwera
- restarting - flaga oznaczająca restart serwera

Klasa Server zawiera następujące metody:

- __init__(addr) - konstruktor klasy Server; Addr - krotka (tuple) złożona z adresu IP oraz portu, na którym ma nasłuchiwać serwer.
- run() - metoda która uruchamia wątek serwera i oczekuje wprowadzenia potencjalnej komendy od użytkownika na wątku głównym
- server_loop() - metoda która jest używana podczas konstrukcji wątku serwera

2.2.2 http_handler.py

Moduł python zawierający klasę webServerHandler, której bazą jest klasa http.server.BaseHTTPRequestHandler. webServerHandler odpowiada za obsługę zapytań GET i POST.

Klasa webServerHandler zawiera następujące właściwości:

- root - ścieżka do folderu zawierającego serwowaną stronę internetową

Klasa webServerHandler zawiera następujące metody:

- do_GET() - metoda wywoływana podczas zapytania GET - serwuje stronę i jej skrypty znajdujące się w root
- do_POST() - metoda wywoływana podczas zapytania POST - obsługuje wymianę danych dot. wpisanego numeru. Użytkownik klikając przycisk "wyślij" wysyła dane o stworzonym przez siebie obrazie, w pliku JSON. Następnie plik ten jest czytany przez serwer, który następnie dokonuje klasyfikacji przy użyciu już wytrenowanej sieci neuralnej.

2.2.3 nnetwork.py

Moduł python zawierajacy klasę Network. Klasa ta zawiera w sobie dane o sieci neuralnej oraz algorytm uczenia.

Sieć neuralna składa się z czterech warstw:

- 784 neurony wejściowe (obraz 28x28 pix, wartości od 0 - 1 gdzie 1 to zarysowany obszar)
- Dwie warstwy ukryte po 30 neuronów
- 10 neuronów wyjściowych (oznaczające pewność klasyfikacji dla cyfr od 0 do 9)

Do nauczenia sieci został wykorzystany algorytm *gradientu prostego (gradient descend)* połączony z algorytmem *propagacji wstecznej (backpropagation)*.

Gradient prosty Metoda gradientu prostego opiera się na prostej obserwacji, nasza sieć neuralna to tak naprawdę pewna skomplikowana funkcja. Problem polega na tym, że nie możemy zminimalizować sieci bezpośrednio (co by to w ogóle znaczyło?), natomiast możemy zminimalizować pewną inną powiązaną funkcję. Taką funkcję nazywa się funkcją kosztu (eng. cost/loss function). Funkcją kosztu może być dowolna funkcja, która będzie wskazywała na pewien "odchył" od prawidłowego stanu.
W przypadku programu funkcją kosztu jest funkcja kosztu kwadratowego:

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2. \tag{1}$$

Gdzie:

- w - wagi
- b - biasy
- $y(x)$ - prawidłowe wartości na neuronach wyjściowych (wektor 10 wymiarowy)
- a - rzeczywiste wartości powstałe na neuronach wyjściowych (wektor 10 wymiarowy)

Musimy więc znaleźć minimum funkcji kosztu, tradycyjna metoda niestety tutaj nie zadziała, gdyż funkcja ta w przypadku tego programu będzie miała tysiące zmiennych, dlatego też użyjemy metody przybliżonej. Wyliczając gradient takiej funkcji możemy łatwo określić *kierunek* w którym musimy się poruszać, aby dotrzeć do jej minimum.

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} ... \frac{\partial C}{\partial v_n} \right)^T. \tag{2}$$

Pozwala nam to na określenie wektora zmian

$$\Delta v = -\eta \nabla C, \tag{3}$$

gdzie η to parametr uczenia (tzn. jak daleko przemieścimy się w następnym kroku). To wszystko przekłada się na generalną "zasadę" poruszania się w kierunku minimum.

$$v \to v' = v - \eta \nabla C. \tag{4}$$

Więc w naszym przypadku przekładamy to na odpowiednie wagi i biasy

$$w_k \quad \to \quad w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \tag{5}$$

$$b_l \quad \to \quad b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \tag{6}$$

Klasa Network zawiera następujące właściwości:

- num_layers - liczba warstw neuronów
- sizes - tuple zawierający liczbę neuronów w każdej warstwie
- biases - lista, zawierająca tablice (numpy) biasów odpowiednich warstw (pomijając warstwę wejściową)
- weights - lista, zawierająca tablice (numpy) wag odpowiednich warstw, np. dla warstwy (10) i warstwy (2) tablica ta będzie miała wymiar 10x2 gdyż do każdego neuronu musimy powiązać każdy neuron z następnej warstwy

Klasa Network zawiera następujące metody:

- __init__(sizes=None, path=None) - konstruktor klasy Network; dołączenie *sizes* spowoduje utworzenie tablic o zadanych rozmiarach, z losowymi elementami (losowymi w rozkładzie gaussa pomiędzy 0 a 1); dołączenie argumentu path spowoduje próbę odczytania sieci z zadanego pliku. (Formatem sieci jest prosty plik JSON)
- feedforward(a) - oblicza wektor wyjściowy na podstawie zadanego wektora wejściowego
- SGD(training_data, epochs, mini_batch_size, eta, test_data=None) - wykonuje algorytm prostego gradientu; training_data - lista tupli, w których przechowywane są parami wektory wejściowe i poprawne wektory wyjściowe; epochs - ilość powtórzeń; mini_batch_size - rozmiar małych paczek z danymi wejściowymi (ilość w tuplach na paczkę), ustawienie tego parametru na 1 spowoduje wykorzystanie zwykłego algorytmu gradientu prostego, ustawienie parametru na wielkość większą od jeden spowoduje użycie przybliżonego algorytmu gradientu prostego; eta - szybkość uczenia; test_data - tak samo jak training_data, jeśli zostanie podane spowoduje to, że w każdym epochu/powtórzeniu sieć zostanie przetestowana, a postęp nauki wyświetlany co powtórzenie.
- backprop(x,y) - funkcja obliczająca $\frac{\partial C}{\partial w_k}$ oraz $\frac{\partial C}{\partial b_l}$ stosując algorytm propagacji wstecznej; x - wektor wejściowy; y - spodziewany wektor wyjścia.
- evaluate(test_data) - funkcja sprawdzająca liczbę poprawnych ewaluacji sieci neuralnej.
- cost_derivative(output_activations, y) - oblicza wartość pochodnej dla funkcji kosztu.

2.2.4 ml_service.py

Moduł python zawierający klasę MLService. Klasa ta jest odpowiedzialna za przygotowywanie informacji wejściowych.

Klasa MLService zawiera w sobie jedną właściwość statyczną:

- net - instancja klasy Network

Klasa MLService zawiera w sobie następujące metody:

- prepareImage(img_data) - przygotowuje obraz w gotowy do użycia wektor 784 wymiarowy; img_data - dane obrazu w formie JSON
- evaluate(img_data) - zwraca wektor wyjściowy; img_data - wektor 784 wymiarowy z danymi obrazu.