Wydział Matematyki
i Nauk Informacyjnych

POLITECHNIKA WARSZAWSKA

Metody głębokiego uczenia

Projekt 3

Sieci rekurencyjne i LSTM

Michał Rdzany, Piotr Sowiński

Spis treści

Wstęp

Opis problemu

Zadanie polega na stworzeniu modelu rozpoznającego komendy głosowe. Wymagane jest wykorzystanie sieci LSTM, sieci rekurencyjnych lub ich modyfikacji.

Cel badań

Głównym celem projektu jest poznanie metod tworzenia sieci rekurencyjnych i LSTM w praktyce.

Opis danych

Dane pochodzą ze zbioru „Speech Commands Data Set v0.01”[1]. Zbiór treningowy zawiera 64727 plików audio, przypisanych do 20 kategorii (pojedyncze słowa). Klipy trwają około 1s. Zbiór zawiera także kilka plików audio z szumem. Zbiór testowy składa się z 158538 plików. Jest on duży, aby ograniczyć oszukiwanie w konkursie; tylko część z tych danych jest faktycznie brana pod uwagę przy wyliczaniu skuteczności. Liczba przewidywanych klas jest ograniczona do 10 różnych słów. Pozostałe nagrania powinny być przypisane do klasy *silence* lub *unknown*.

Sposób weryfikacji rezultatów

Ze zbioru treningowego został wydzielony zbiór walidacyjny do lokalnego sprawdzania skuteczności modelu. Weryfikacja klasyfikacji na zbiorze testowym następuje poprzez zarejestrowanie wyników na platformie kaggle (https://www.kaggle.com/c/tensorflow-speech-recognition-challenge).

Przygotowanie danych

Dostosowanie długości audio

Większość plików ma długość 1s (16000 próbek). Pliki z inną częstotliwością próbkowania są odpowiednio przeskalowywane. Na koniec krótszych plików dopisywane są 0, aby zachować ten sam rozmiar. Z dłuższych plików (pojawiających się w folderze *_background_noise_*) wyciągane są losowe fragmenty o pożądanym rozmiarze.

Generowanie spektrogramów

Jako dane wejściowe do modelu przekazywane są logarytmy spektrogramów generowanych z wczytanych danych. Przy generacji spektrogramów wykorzystano okno Hanna z szerokością 240 i 50-procentowy overlap.

Standaryzacja

Na przygotowanych danych zostały wyliczone wartości średnie i wariancje. Wykorzystując te parametry, dane wejściowe do modelu są standaryzowane.

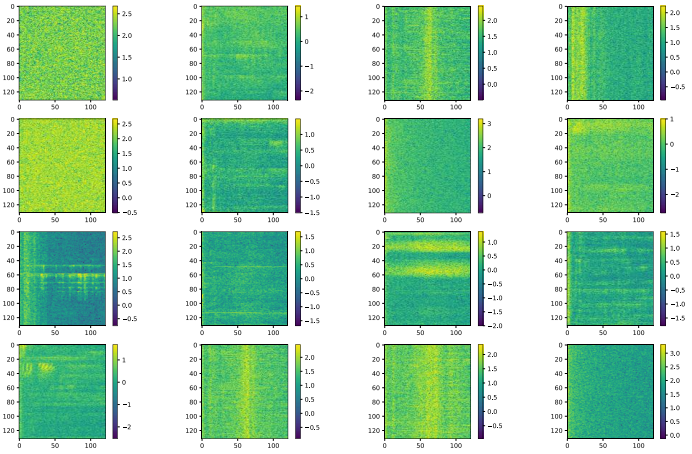
Selekcja i transformacje danych

Dane testowe mają wyraźnie inną proporcję klas niż dane treningowe. Klasa *unknown* stanowi około 60% danych treningowych, ale zaledwie 10% danych testowych (wśród plików faktycznie branych pod uwagę przy wyliczaniu skuteczności). W efekcie modele trenowane na wszystkich dostępnych danych zbyt często klasyfikowały wypowiadane słowa jako nieznane. Problem ten został rozwiązany poprzez odrzucenie części danych treningowych z klasy *unknown*. Najlepsze rezultaty dawały modele wykorzystujące 10% tych

danych. Aby wykorzystać wszystkie dostępne pliki można utworzyć komitet modeli trenowanych na różnych częściach danych z klasy *unknown*.

Sytuacja wydaje się podobna w klasie *silence*, gdzie mamy tylko 6 plików w zbiorze treningowym, jednak z uwagi na ich długość, jest z nich tworzone więcej danych i ilość ta jest wystarczająca dla modeli do poprawnego klasyfikowania podobnych plików w zbiorze testowym.

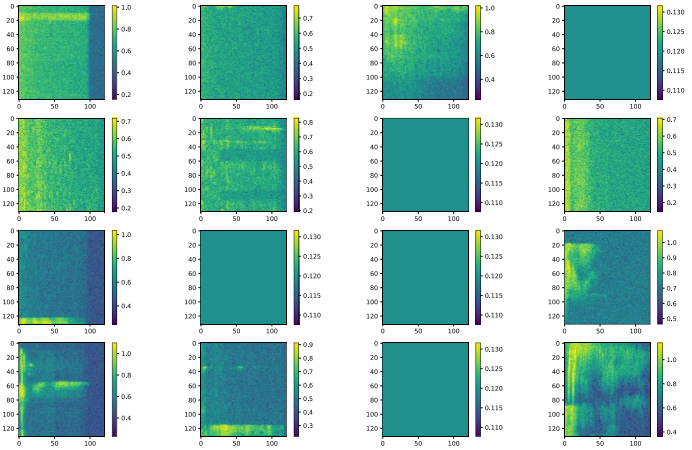
W klasie *silence* mamy natomiast inny problem. Słuchając plików ze zbioru testowego, których model nie klasyfikuje do żadnej z klas z wysokim prawdopodobieństwem (poniżej 30% dla wszystkich klas), okazuje się, że są to bardzo ciche nagrania, w których amplituda sygnału jest równa lub bliska zeru.



Przykłady z klasy *silence*, wygenerowane z danych treningowych

Oznaczenie części tych plików (np. wśród pierwszych 2000 plików ze zbioru testowego) jako *silence* (skopiowanie do folderu *_background_noise_*) i wytrenowanie modelu razem z tymi dodatkowymi danymi podnosi skuteczność na zbiorze testowym o niemal 5 punktów procentowych.

Aby nie wykorzystywać w treningu danych testowych spróbowano dodać do zbioru treningowego kilka pustych plików i przetransformować dane wczytane z folderu *_background_noise_*, tak aby część z nich miała mniejsze amplitudy, jednak mimo wielu różnych sposobów transformacji, nie wpłynęło to znacząco na skuteczność modelu.



Przykłady testowe nie klasyfikowane wyraźnie jako żadna z klas

Z uwagi na brak możliwości wygenerowania z danych treningowych przykładów reprezentatywnych dla zbioru testowego, zdecydowano się zatrzymać te pliki w zbiorze treningowym. Autorzy modeli z najlepszymi wynikami opisywali wykorzystanie podobnych technik[2] (chociaż bez tak wyraźnego wzrostu skuteczności), więc uznaliśmy, że jest to akceptowalne.

Zbiór walidacyjny

Wydzielając zbiór walidacyjny standardowo (jako losową próbkę z danych treningowych) okazało się, że skuteczność modelu na tym zbiorze nie odzwierciedla skuteczności na zbiorze testowym (mimo opisanych selekcji i transformacji). Próbując rozwiązać ten problem podzielono dane w taki sposób, aby osoby mówiące w zbiorze walidacyjnym nie pojawiały się w zbiorze treningowym, lecz niestety nie przyniosło to pożądanych rezultatów. Na zbiorze testowym nadal błędnie klasyfikowanych jest około trzy razy więcej plików niż na zbiorze walidacyjnym.

Odsłuchując pliki ze zbioru treningowego i testowego nie natrafiono na wyraźne różnice. Większy procent plików w danych testowych był gorszej jakości (niezrozumiałe słowa), ale przesłuchany został jedynie niewielki fragment zbiorów i mógł to być przypadkowy rezultat. Nie mamy możliwości stwierdzenia, które pliki faktycznie są brane pod uwagę przy ocenie skuteczności, może mają one jakieś cechy szczególne odróżniające je od innych i w efekcie utrudniające klasyfikacje.

Opis zaimplementowanej sieci

Instrukcja użycia aplikacji

Aby wytrenować bazowy model wystarczy posiadać podany we wstępie zbiór danych i uruchomić kilka pierwszych bloków z pliku ipynb. Na systemach innych niż Windows może być konieczna modyfikacja kodu wczytującego pliki. Aby uzyskać opisywane w tym raporcie wyniki należy dodatkowo dodać do zbioru treningowego pliki z klasy *silence* w sposób opisany w poprzednim rozdziale, w sekcji „Selekcja i transformacje danych", a także wykorzystać przy treningu inne dane ze zbioru testowego, co jest opisane w sekcji „Uczenie częściowo nadzorowane".

Architektura sieci

Wykorzystywana jest sieć CNN-LSTM bazująca na rozwiązaniu jednego z użytkowników platformy kaggle[3].

Rozwiązanie to osiągało niską skuteczność i zostało gruntownie zmodyfikowane, skutkując znacznym polepszeniem wyników.

Większe architektury (z większą ilością warstw lub z większymi warstwami) nie poprawiają wyraźnie skuteczności modelu.

Wykorzystanie GPU do treningu sprawia, że nawet przy ustawieniu jednakowego ziarna, wyniki mogą być różne. Trenowanie modelu na jednym wątku CPU, aby zapewnić odtwarzalność wyników, nie było możliwe z uwagi na długi czas treningu. Odchylenie standardowe skuteczności na zbiorze testowym to około 0.002.

Input: Array 132x121
Gaussian Noise 0.2
Conv1D 256 strides 2
ELU + Batch Normalization
Dropout 0.3
Conv1D 512 strides 2
ELU + Batch Normalization
Dropout 0.4
Bidirectional LSTM 512 (recurrent dropout 0.5)
Dropout 0.5
Dense 512
ELU + Batch Normalization
Dropout 0.5
Dense 12 - softmax

Wykorzystane techniki

Patrząc na przedstawione w tym rozdziale wykresy, należy wziąć pod uwagę, że na zbiorze testowym błędnie klasyfikowanych jest około trzy razy więcej plików niż na zbiorze walidacyjnym, co zostało dokładniej opisane we wcześniejszej części raportu. Mimo to z przeprowadzonych eksperymentów wynika, że większa skuteczność na zbiorze walidacyjnym skutkuje większą skutecznością na zbiorze testowym, więc można go użyć do porównania modeli.

Funkcje aktywacji i straty

Jako funkcja aktywacji w sieci wykorzystywana jest funkcja ELU[4]. Przetestowano także funkcje ReLU i PReLU, jednak dawały one gorsze rezultaty.

ELU nie jest używana w warstwie LSTM, gdzie zastosowane są domyślne funkcje (sigmoida i tangens hiperboliczny), a także w ostatniej warstwie, gdzie używana jest funkcja softmax.

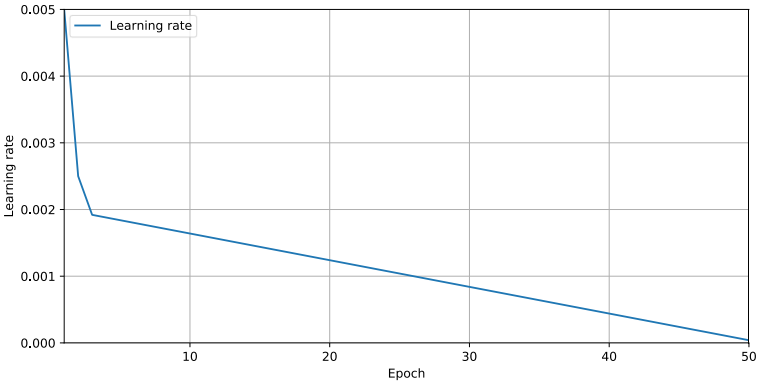
Używana funkcja straty to cross-entropy.

Optymalizacja

Model wykorzystuje algorytm Nadam[5]. Przetestowano także algorytmy Adam i SGD, które działały jednak nieco gorzej. Bardzo istotne okazało się poprawne dobranie współczynnika nauki w kolejnych epokach.

Przez dwie pierwsze epoki współczynnik nauki lr jest zmniejszany dwukrotnie po każdej epoce.

W dalszej części procesu uczenia współczynnik nauki maleje liniowo do zera ($lr - ax$).



Uczenie batchowe

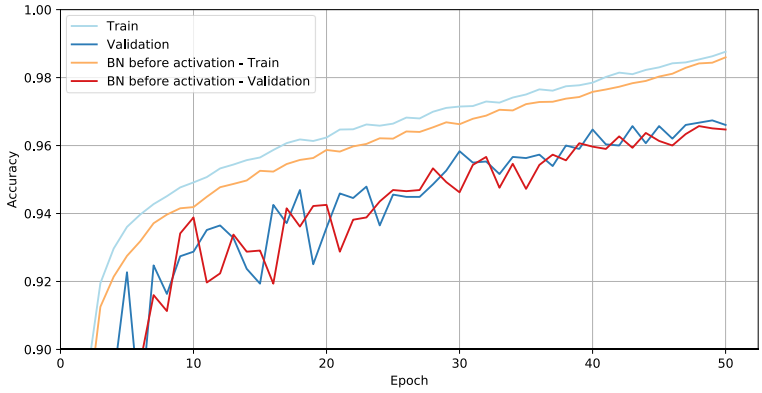
Zastosowano uczenie batchowe z batchami rozmiaru 256. Rozmiar batchy został wybrany na podstawie wyników osiąganych przez model.

Inicjalizacja

W warstwach wykorzystujących ELU używana jest inicjalizacja He_uniform, a w pozostałych Glorot_uniform. Zastosowanie inicjalizacji Glorot_uniform we wszystkich warstwach daje podobne rezultaty.

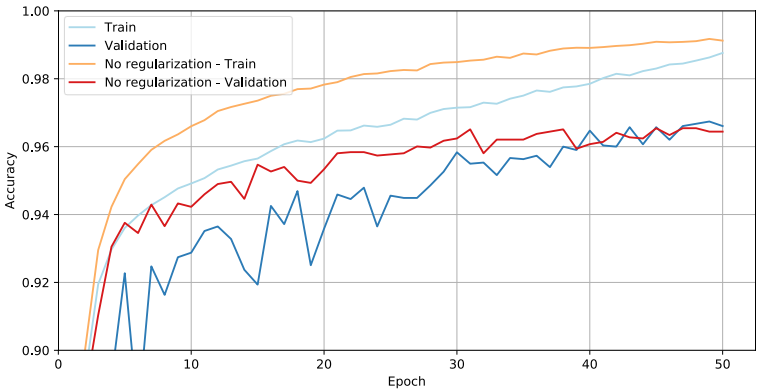
Batch normalization

Zastosowanie tych warstw znacząco polepsza działanie modelu i pozwala na wykorzystanie większych współczynników nauki i modeli z większą ilością warstw. W pracy prezentującej tę technikę[6], warstwa ta jest umieszczona przed funkcją aktywacji. Okazuje się jednak, że w naszym modelu umieszczenie jej po funkcji aktywacji daje stale nieco lepsze rezultaty.



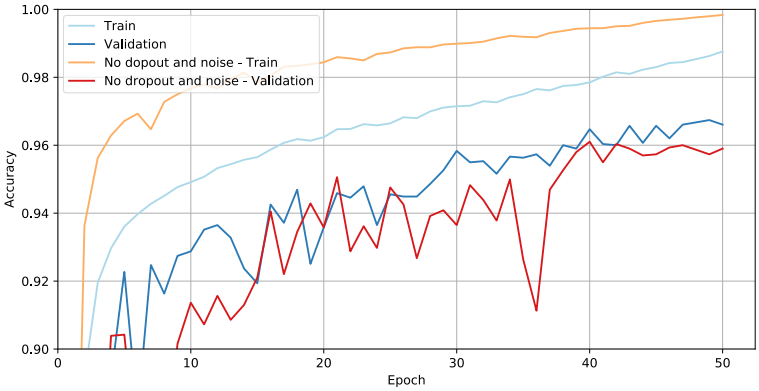
Regularyzacja

Regularyzacja nie daje tak dobrych rezultatów jak w poprzednim projekcie (dotyczącym sieci CNN), ale pozwala w niewielkim stopniu podnieść skuteczność modelu.



Dropout

Dropout zadziałał lepiej niż regularyzacja, ale też gorzej niż w poprzednim projekcie. Najlepsze wyniki osiągały modele z dość dużym dropoutem, co może być spowodowane niewielkim rozmiarem zbioru treningowego. Najskuteczniejsze okazało się umieszczenie dropoutów w całej sieci i zwiększanie ich wraz z głębokością.



Uczenie częściowo nadzorowane

Dane ze zbioru testowego, przypisywane przez model do danych klas z dużym prawdopodobieństwem (powyżej 99.99%) zostały wykorzystane jako dane treningowe przy budowie ostatecznego modelu. Wyjątkiem jest klasa *unknown*, gdzie zamiast korzystać z danych testowych, użyto większej części danych treningowych (25%). Technika ta nieznacznie podniosła skuteczność modelu na danych testowych (o 0.25 punktów procentowych). Pomysł na zastosowanie tej techniki pochodzi z dyskusji prezentującej różne rozwiązania, opublikowanej przez zwycięzcę konkursu[7].

Dostosowanie dystrybucji słów w klasie *unknown*

Analizując predykcje modelu na zbiorze testowym, dla których przypisywane prawdopodobieństwo nie było zbyt duże (od 90% do 99.99%), odsłuchując pliki, okazało się, że większość błędnie klasyfikowanych słów pochodzi z klasy *unknown*. Możliwe, że jest to pożądane zachowanie, ponieważ duża część danych z klasy unknown w zbiorze testowym najprawdopodobniej nie jest brana pod uwagę przy wyliczaniu skuteczności. Większość zaobserwowanych błędów dotyczyła konkretnych słów:

- *backward* często klasyfikowane jako różne inne słowa (najczęściej *up* lub *left*)
- *five* często klasyfikowane jako *right*
- *follow* i *four* często klasyfikowane jako *on* lub *off*
- *forward* klasyfikowane jako *on*
- *nine* klasyfikowane jako *down* lub *right*

Bazując na tych informacjach dostosowano zawartość danych z klasy *unknown* wykorzystywanych przy treningu, zwiększając liczbę przykładów z kłopotliwymi dla modelu słowami. Niestety nie miało to żadnego wpływu na skuteczność na danych testowych, więc zrezygnowano z tego pomysłu.

Inne techniki, możliwe do wykorzystania przy dalszym rozwoju modelu

Augmentacja danych

Losowe modyfikacje plików używanych do treningu np. poprzez rozciąganie dźwięku czy modyfikacje częstotliwości, mogłyby pomóc w osiągnięciu lepszej skuteczności na zbiorze testowym. Podobną funkcję spełnia wykorzystana w modelu warstwa *Gaussian Noise*.

Komitety

Możliwe jest utworzenie komitetu kilku modeli, wytrenowanych na różnych częściach danych z klasy *unknown*. Podejście to przetestowano na mniejszym modelu, gdzie taki komitet 10 modeli osiągał skuteczność o około 0.4 punktów procentowych lepszą od pojedynczego modelu. Technika ta nie została wykorzystana w ostatecznym rozwiązaniu z uwagi na długi czas treningu.

Wyniki

Skuteczność modelu

Przy ostatecznym treningu modelu nie wydzielano zbioru walidacyjnego. Modele wykorzystywane do selekcji wykorzystywanych danych ze zbioru testowego były trenowane przez 200 epok, a ostateczny model przez 100 epok, z uwagi na większy rozmiar zbioru treningowego. W konkursie na platformie Kaggle model osiągnął skuteczność 89.19%. Patrząc na tabelę wyników, jest to dość dobry wynik, co może sugerować, że sieci LSTM są w stanie uzyskiwać jedne z lepszych rezultatów dla tego problemu. Szacujemy, że przy dalszym rozwoju modelu możliwe byłoby osiągnięcie skuteczności w okolicach 90%. Porównując wynik do innych rozwiązań o podobnej architekturze, udało nam się polepszyć skuteczność modelu, na którym bazowaliśmy o około 15 punktów procentowych. W dyskusjach dotyczących wykorzystania sieci rekurencyjnych w tym konkursie, użytkownicy raportują skuteczność do 85%[8][9].

Cytowane prace

1. **Warden, Pete.** *Speech Commands: A public dataset for single-word speech recognition.* Dataset available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz, 2017.

2. **Pavel Ostyakov.** *Silence Trick.* https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/discussion/47674, 2018.

3. **Himanshu Rawlani.** *A CNN + LSTM model.* https://www.kaggle.com/himanshurawlani/a-cnn-lstm-model, 2019.

4. **Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter.** *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).* 2016.

5. **Timothy Dozat.** *Incorporating Nesterov Momentum into Adam.* 2016.

6. **Sergey Ioffe, Christian Szegedy.** *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015.

7. **Heng CherKeng.** *My Tricks and Solution.* https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/discussion/46945, 2018.

8. **Vivek Pandey.** *Using recurrent neural networks.* https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/discussion/46439, 2018.

9. **Pete Warden.** *Hello Edge: Keyword Spotting on Microcontrollers.* https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/discussion/45037, 2017.