Standardowa sieć rekurencyjna do rozpoznawania fonemów w sygnale mowy (U7)

Mateusz Czyżnikiewicz (269275)

Ośrodek Kształcenia na Odległość, Politechnika Warszawska, Polska
mateusz.czyznikiewicz.stud@pw.edu.pl

1 Wstęp

Automatyczne rozpoznawanie mowy jest jednym z podstawowych zadań przetwarzania języka naturalnego oraz jednym z podstawowych elementów wielu inteligentych systemów agentowych. Moduł rozpoznający mowę stanowi pierwszy moduł przetwarzający dane w wirtualnych asystentach (Google Assistant [1], Bixby [2] czy Siri [3]) oraz różnego rodzaju czatbotach. Z tego powodu jest tematem wielu badań i na przestrzeni lat zaproponowano wiele różnych rozwiązań tego zadania.

Projekt zrealizowany w ramach przedmiotu *Inteligentne techniki obliczeniowe* jest uproszczoną wersją systemu rozpoznawania mowy. W szczególności polegał on na realizacji neuronowej sieci rekurencyjnej klasyfikującej kolejne ramki z cechami sygnału mowy do odpowiadających im tokenów (fonemów lub liter). Sygnały mowy zostały wstępnie przetworzone, tzn. posegmentowane oraz zostały z nich wyekstrahowane odpowiednie cechy akustyczne. Przeanalizowane zostały wyniki uczenia oraz ewaluacji kilku takich sieci o różnych parametrach.

Poniższy raport jest raportem końcowym z realizacji tego projektu. W drugim rozdziale, szczegółowo omówiono sieci neuronowe, ze szczególnym uzwględnieniem sieci rekurencyjnych, pod kątem przetwarzania danych sekwencyjnych, jako metodykę sztucznej inteligencji. Trzeci rozdział zawiera opis zadania automatycznego rozpoznawania mowy jako problemu rozwiązywanego przy użyciu inteligentnych technik obliczeniowych, w szczególności sieci neuronowych. W czwartym rozdziale opisane zostało wykonanie projektu, łącznie z modelowanymi danymi, podziałem projektu na moduły oraz sposobem testowania i ewaluacji. W piątym rozdziale znalazła się analiza wyników uzyskanych podczas eksperymentów, natomiast ostatni rozdział stanowi podsumowanie.

2 Rekurencyjne sieci neuronowe

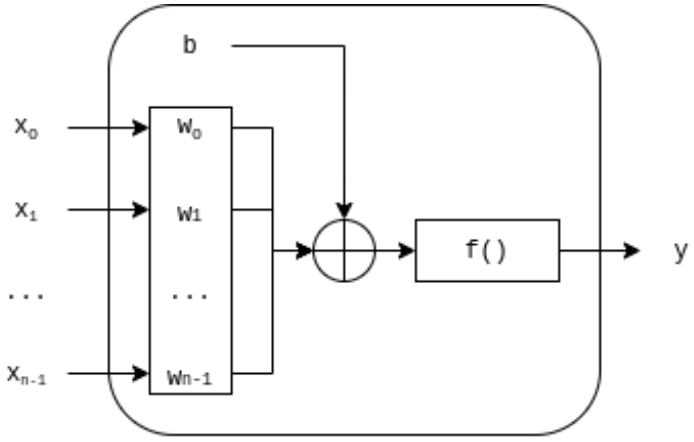
Poniższy rozdział zawiera wprowadzenie do tematyki sieci neuronowych. Opisane zostały podstawowe pojęcia takie jak neuron, warstwa czy funkcja aktywacji. W szczególności skupiono się na sieciach rekurencyjnych jako metodzie przetwarzania danych sekwencyjnych. Pokrótce omówiono także stosowane obecnie, a bardziej zaawansowane architektury sieci, takie jak LSTM [4], GRU [5] czy Transformer [6]. Opisano także proces treningu sieci neuronowej poprzez wsteczne propagowanie błędów.

2.1 Podstawowe pojęcia

Sztuczna sieć neuronowa to zestaw prostych jednostek obliczeniowych przetwarzających dane. Model ten jest luźno inspirowany działaniem ludzkiego mózgu, a w szczególności pojedynczych neuronów. Podstawową jednostką w sieci neuronowej jest neuron, który schematycznie przedstawiono na rysunku 1. Działanie sztucznego neuronu można przedstawić za pomocą równania:

$$y = f(\Sigma_{i=0}^{n-1} w_i x_i + b) = f(w^T x + b) \tag{1}$$

gdzie x_i jest i-tym wejściem, w_i jest i-tą wagą, b jest progiem (ang. bias), f jest funkcją aktywacji, natomiast y jest wyjściem neuronu lub w zapisie wektorowym w jest wektorem wag, natomiast x wektorem wejść.



Rysunek 1. Schemat sztucznego neuronu.

Zadaniem funkcji aktywacji jest wprowadzenie do przetwarzania danych nieliniowości, co pozwala na składanie wielu neuronów. W przeciwnym przypadku, przez to, że wiele operacji liniowych można zastapić jedną, składanie neuronów bez funkcji aktywacji nie miałoby sensu. Obecnie najczęściej spotykaną funkcją aktywacji jest ReLU (ang. Rectified Linear Unit) $f(x) = max(x, 0)$ [7]. Jest tak ze względu na szybkość oraz skuteczność jej działania. Inne czasem spotykane funkcje aktywacji to m.in. tangens hiperboliczny $f(x) = tanh(x)$ czy też funkcja sigmoidalna $f(x) = \frac{1}{1+e^{-x}}$.

W praktyce częściej rozważa się całą warstwę zamiast pojedynczych neuronów. Pozwala to na znaczne uproszczenie formalizmu przy pomocy operacji na macierzach. Podstawową i często wykorzystywaną warstwą jest warstwa w pełni połączona (ang. fully-connected), w której każdy neuron jest połączony z każdym neuronem warstwy poprzedzającej. Działanie takiej warstwy można opisać przy pomocy równania:

$$y = f(xW^T + b) \tag{2}$$

gdzie x to wektor zawierający wejścia do warstwy, b to wektor zawierający odpowiednie progi (ang. bias), natomiast W to macierz zawierająca wagi neuronów warstwy. Do innych często stosowanych warstw należą:

- warstwy konwolucyjne - stosowane w przetwarzaniu obrazów, chociaż w ostatnim czasie często także do przetwarzania języka i innych danych sekwencyjnych,
- warstwy rekurencyjne (standardowe, GRU [5], LSTM [4]) - głównie używane do przetwarzania danych sekwencyjnych, np. sygnałów dźwiękowych czy języka (słów/liter w tekstach).

Sieci neuronowe, wykorzystywane w praktyce, są najczęściej złożeniem wielu warstw. Jest to widoczne szczególnie w głębokich sieciach neuronowych (ang. deep neural networks), które mogą się składać z dziesiątek a nawet setek warstw ułożonych jedna po drugiej.

Jedną z pierwszych i podstawowych, a do dzisiaj stosowanych, architektur jest perceptron wielowarstwowy (ang. multilayer perceptron), który składa się z danej liczby warstw w pełni połączonych o danej liczbie jednostek (neuronów). Sieć taka może być wykorzystywana zarówno do klasyfikacji jak i do regresji.

2.2 Przetwarzanie danych sekwencyjnych

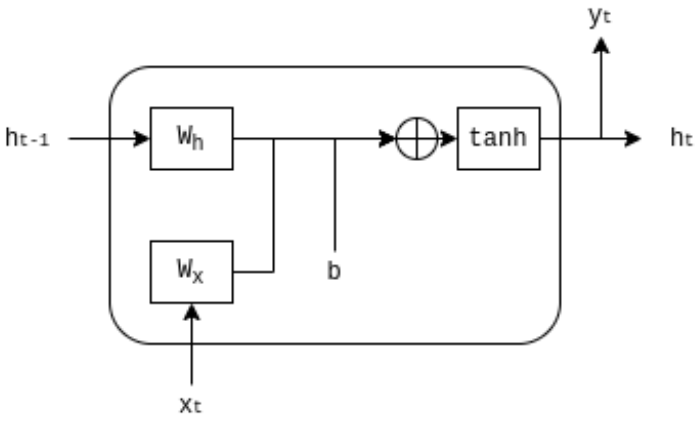
Dużym problemem w przetwarzaniu danych sekwencyjnych, takich jak sygnał dźwiękowy czy tekst, jest brak stałości rozmiaru (długości) danych wyjściowych i wejściowych. Warstwy w pełni połączone i konwolucyjne wymagają, aby przetwarzane dane miały stałe rozmiary. Istnieją oczywiście sposoby stosowania tych warstw do przetwarzania danych sekwencyjnych [8], [9], jednak sieci rekurencyjne są bardziej naturalnym podejściem, przy czym zapewniają też wysoką efektywność.

Jedna z podstawowych komórek rekurencyjnych została zaproponowana w [10]. Jej schemat został przedstawiony na rysunku 2, a jej działanie można opisać równaniami:

$$\begin{aligned} h_t &= tanh(x_t W_x^T + h_{t-1} W_h^T + b) \\ y_t &= h_t \end{aligned} \tag{3}$$

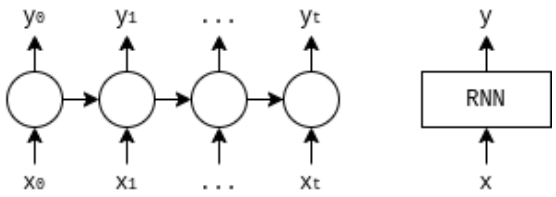
gdzie x_t jest wektorem wejść w momencie czasowym t, h_t jest wektorem stanu ukrytego w momencie czasowym t, y_t jest wektorem wyjść w momencie czasowym t, $W_{h|x}$ jest macierzą odpowiednich wag, a b jest wektorem progu (ang. bias). Przy czym wykorzystanie tangensa hiperbolicznego jako funkcji aktywacji jest standardowym podejściem, jednak nic nie stoi na przeszkodzie, żeby stosować też inne funkcje aktywacji.

Taka komórka jest stosowana we wszystkich momentach czasowych t, zachowując przez cały czas te same wagi i przekazując stan ukryty h_t między kolejnymi krokami. Pozwala to na przetworzenie sekwencji danych o dowolnej długości, produkując przy tym sekwencję wyjściową o tej samej długości oraz przekazując informację między kolejnymi momentami czasowymi.



Rysunek 2. Schemat komórki standardowej sieci rekurencyjnej.

W praktyce często stosuje się rozwinięcie w czasie takiej komórki, przedstawiono je na rysunku 3. Mówi się wtedy o warstwie rekurencyjnej, która na wejściu przyjmuje sekwencję wektorów, na wyjściu produkując przetworzoną sekwencję o tej samej długości. Dzięki przejściu na ten poziom abstrakcji możemy wyeliminować potrzebę rozważania stanów ukrytych pomiędzy kolejnymi krokami czasowymi, które są zarządzanie wewnętrznie w warstwie.



Rysunek 3. Schemat warstwy rekurencyjnej. Po lewej na poziomie pojedynczych wektorów sekwencji. Po prawej na poziomie sekwencji wejściowej reprezentowanej przy pomocy macierzy.

Warstwa rekurencyjna przedstawiona na rysunku 3 przekazuje stan ukryty w kierunku rosnących momentów czasowych, jednak często nic nie stoi na przeszkodzie, żeby stan ukryty był przekazywany także w drugim kierunku. Mówimy wtedy o dwukierunkowej warstwie rekurencyjnej. Są one często stosowane do kodowania sekwencji danych, które są z góry znane. Dobrym przykładem jest tutaj model językowy ELMo [11].

Standardowe komórki rekurencyjne nie sprawdziły się jednak w praktycznych zastosowaniach przez to, że stan ukryty jest cały czas nadpisywany, co powoduje trudności w zapamiętywaniu przydatnych informacji z poprzednich momentów

czasowych. W celu rozwiązania tego problemu zaproponowano inne komórki rekurencyjne. Dwie najbardziej popularne i najczęściej używane to LSTM [4] (ang. Long Short-term Memory) oraz GRU [5] (ang. Gated Recurrent Unit). Ich przewagą nad standardową komórką rekurencyjną jest posiadanie tzw. bramek, które w praktyce są dodatkowymi wagami odpowiedzialnymi za odpowiednie przetwarzanie stanu ukrytego. Jednak komórki te nie są przedmiotem projektu, więc więcej informacji na ich temat można znaleźć w literaturze [12].

W kwestii przetwarzania danych sekwencyjnych nie można pominąć architektury Transformer [6] bazującej na mechanizmie atencji, a która w ostatnim czasie pozwoliła na znaczący rozwój wszystkich dziedzin związanych z sieciami neuronowymi, a w szczególności na wyraźny rozwój technik przetwarzania języka naturalnego. Dzięki zastosowaniu atencji, sieć ma w każdym kroku czasowym dostęp do całości poprzedniego kroku czasowego a nie tylko do tego samego kroku czasowego (jak w przypadku sieci rekurencyjnych). Dodatkowo sieć sama uczy się w trakcie treningu na które kroki czasowe z poprzedniej warstwy powinna zwracać uwagę. Oczywiście architektura Transformer ma także swoje problemy np. rosnącą kwadratowo wraz z długością sekwencji złożoność obliczeń, co powoduje problemy przy przetwarzaniu długich sekwencji. Jednak architektura ta także nie jest przedmiotem tego projektu, więc więcej informacji na jej temat można znaleźć w literaturze [13].

2.3 Trening sieci neuronowej

Podstawowym algorytmem uczącym sieci neuronowe jest algorytm wstecznej propagacji błędów (ang. backpropagation). Jest to algorytm przeszukiwania przestrzeni wag sieci stosując algorytm wzrostu. W uproszczeniu, startując z arbitralnie dobranego wektora wag, obliczany jest wektor gradientu funkcji błędu i jako nowy wektor wag przyjmowany jest wektor znajdujący się na półprostej wyprowadzonej w kierunku przeciwnym do gradientu. Do wyboru miejsca na tej półprostej wykorzystywany jest współczynnik uczenia sieci (ang. learning rate).

W praktyce stosowane są różne modyfikacje tego algorytmu pozwalające na szybszą czy też bardziej niezawodną optymalizację. Przykładem takiej modyfikacji jest dodanie do algorytmu momentu bezwładności (ang. momentum), który pozwala na branie pod uwagę nie tylko gradientu z rozważanego obecnie kroku treningu, ale także gradientu z poprzednich kroków. Przyspiesza to proces zbiegania wag do optymalnych wartości.

W treningu sieci neuronowych stosuje się różne funkcje błędu. Jednymi z najbardziej popularnych jest entropia krzyżowa (ang. cross-entropy), stosowana w problemach klasyfikacji, czy też średni błąd bezwzględny (ang. mean absolute error) lub średni błąd kwadratowy (ang. mean squeared error), stosowane często w regresji.

W treningu sieci neuronowych, tak samo jak w przypadku innych algorytmów uczenia maszynowego, mogą się pojawiać problemy związane ze zbytnim dopasowaniem do danych treningowych. Stosuje się różne metody regularyzacji w celu przeciwdziałania temu zjawisku. Można do nich zaliczyć:

– stosowanie dropoutu (losowe zerowanie pewnego procentu wyjść z poszczególnych warstw),
– dodatkowe wyrażenia w funkcji błędu wprowadzające ograniczenia na wielkość wag,
– wyżarzanie wag (ang. weight decay),
– warstwy normalizujące [14] [15].

Dzięki zastosowaniu reguły łańcuchowej do obliczeń gradientu funkcji błędu względem różnych parametrów sieci, trening kolejnych warstw (propagowanie gradientu wstecz i aktualizowanie parametrów) może być rozważany oddzielnie. W następnych podrozdziałach znajdują się wyprowadzenia równań propagacji wstecznej gradientu dla kilku przykładowych warstw wykorzystanych w projekcie.

2.4 Funkcja kosztu

Funkcja kosztu może być traktowana jako ostatnia warstwa sieci, od której rozpoczyna się propagowanie gradientu wstecz. W momencie predykcji taka warstwa oczywiście jest pomijana.

Funkcją kosztu stosowaną w tym projekcie była entropia krzyżowa, która określa różnicę między prawdziwym rozkładem prawdopodobieństwa p (etykiety w klasyfikacji) a przewidzianym rozkładem prawdopodobieństwa x (wyjście sieci neuronowej). Wyraża się ona wzorem:

$$L(p, x) = -\Sigma_i \, p_i \, log \, x_i \tag{4}$$

W przypadku klasyfikacji etykiety koduje się najczęściej przy pomocy kodowania *one-hot*, więc rozkład p:

$$p_i = \begin{cases} 1 & \text{gdy } y = i \\ 0 & \text{w p.p.} \end{cases} \tag{5}$$

gdzie y to indeks etykiety. Dzięki temu wektor gradientu względem prawdopodobieństw poszczególnych klas upraszcza się do:

$$\frac{dL}{dx_i} = \begin{cases} -\frac{1}{x_i} & \text{gdy } y = i \\ 0 & \text{w p.p.} \end{cases} \tag{6}$$

I taki wektor jest przekazywany wstecz jako początek propagacji błędów w treningu sieci neuronowej z entropią krzyżową jako funkcją kosztu.

2.5 Funkcje aktywacji

Funkcje aktywacji w przeciwieństwie do funkcji kosztu stanowią w sieci warstwy pośrednie. Oznacza to, że w trakcie wstecznej propagacji błędów propagowany jest gradient funkcji kosztu względem wyjścia funkcji aktywacji w stronę wejścia funkcji aktywacji.

Pierwszą funkcją aktywacji wykorzystywaną w projekcie jest tangens hiperboliczny:

$$y(x) = tanh(x) \tag{7}$$

W przypadku tej funkcji, pochodną odczytujemy z tablic i ponieważ każdy element wektora x przetwarzany jest niezależnie, to wektoryzacja obliczeń jest oczywista. Pochodna przedstawiona jest równaniem:

$$\frac{dy}{dx} = 1 - tanh(x)^2 \tag{8}$$

Natomiast propagacja gradientu wygląda następująco:

$$\frac{dL}{dx} = (1 - y^2) \,.\, \frac{dL}{dy} \tag{9}$$

gdzie . oznacza mnożenie element po elemencie (ang. elementwise).

Drugą wykorzystaną funkcją aktywacji jest funkcja softmax, która wyraża się wzorem:

$$y_i(x) = softmax_i(x) = \frac{e^{x_i}}{\Sigma_{j=0}^{N} e^{x_j}} \tag{10}$$

W przypadku tej funkcji, wyjście dla każdego elementu wektora wejściowego zależy od każdego elementu wektora wejściowego. Po obliczeniu pochodnej otrzymujemy:

$$\frac{dy_i}{dx_j} = \begin{cases} y_i(1 - y_i) & \text{gdy } i = j \\ -y_i y_j & \text{w p.p.} \end{cases} \tag{11}$$

Natomiast propagacja gradientu wygląda następująco:

$$\begin{aligned} \frac{dL}{dx_j} &= \Sigma_{k=1}^{N} \frac{dL}{dy_k} \times \frac{dy_k}{dx_j} \\ &= \frac{dL}{dy_j}(y_j - y_j^2) - \Sigma_{k \neq j} y_k y_j \frac{dL}{dy_k} \\ &= y_j \frac{dL}{dy_j} - \Sigma_{k=1}^{N} y_k y_j \frac{dL}{dy_k} \end{aligned} \tag{12}$$

Przy implementacji tego wzoru w postaci zwektoryzowanej należy zwrócić szczególną uwagę na wymiar, po którym odbywa się sumowanie.

Ze względu na to, że funkcję softmax stosuje się najczęściej jako warstwę poprzedzającą obliczanie entropii krzyżowej, będącej funkcją kosztu, w praktyce często wyznacza się wspólny gradient dla tych dwóch warstw. Robi się tak ze względów praktycznych, ponieważ ułatwia to obliczenia oraz implementację. W przypadku tego projektu zdecydowano się jednak traktować softmax oraz entropię krzyżową jako oddzielne obiekty.

2.6 Trening warstwy w pełni połączonej

W przypadku warstw sieci posiadających wagi, poza gradientem funkcji kosztu względem wejścia warstwy (propagowanego wstecz), należy wyznaczyć także gradient funkcji kosztu względem parametrów tej warstwy.

Wartości na wyjściu warstwy w pełni połączonej wyrażają się wzorem:

$$y = f(xW^T + b) \tag{13}$$

gdzie x to wektor zawierający wejścia do warstwy, b to wektor zawierający odpowiednie progi (ang. bias), natomiast W to macierz zawierająca wagi. Na tej podstawie i-te wyjście to:

$$y_i = \Sigma_{k=1}^{N_{in}} x_k w_{k,i} + b_i \tag{14}$$

Łatwo zauważyć, że gradient funkcji kosztu względem i-tego progu to:

$$\frac{dL}{db_i} = \frac{dL}{dy_i} \times \frac{dy_i}{db_i} = \frac{dL}{dy_i} \tag{15}$$

Natomiast względem k, i-tej wagi to:

$$\frac{dL}{dw_{k,i}} = \frac{dL}{dy_i} \times \frac{dy_i}{dw_{k,i}} = x_k \frac{dL}{dy_i} \tag{16}$$

Co można zapisać przy pomocy notacji macierzowej jako:

$$\frac{dL}{dW} = (x^T \frac{dL}{dy})^T \tag{17}$$

Przy czym odpowiednie transpozycje są podyktowane sposobem przechowywania poszczególnych danych w macierzach. Zupełnie analogicznie gradient funkcji kosztu względem wejścia do warstwy wyraża się następująco:

$$\frac{dL}{dx_k} = \Sigma_{i=1}^{N_{out}} \frac{dL}{dy_i} \times \frac{dy_i}{dx_k} = \Sigma_{i=1}^{N_{out}} \frac{dL}{dy_i} w_{k,i} \tag{18}$$

Co przy pomocy rachunku na macierzach można zapisać jako:

$$\frac{dL}{dx} = \frac{dL}{dy} W \tag{19}$$

Przy czym odpowiednie transpozycje są podyktowane sposobem przechowywania poszczególnych danych w macierzach.

Wzory wyrażone w podany sposób pozwalają na zrównoleglenie obliczeń dla całej porcji danych (ang. batch). W takim przypadku wektory wejścia x i wyjścia y zostają zastąpione przez macierze wejść X i wyjść Y zawierające odpowiednio upakowane dane. Wyprowadzenie powyższych wzorów przeprowadzone na przykładzie można znaleźć w [16].

2.7 Trening warstwy rekurencyjnej

Wsteczną propagację błędów w standardowej warstwie rekurencyjnej najłatwiej jest rozważać na poziomie pojedynczej komórki (pojedynczego kroku czasowego). W tym przypadku działanie takiej warstwy można przedstawić przy pomocy równań:

$$\begin{aligned} s_t &= x_t W_x^T + h_{t-1} W_h^T + b \\ h_t &= tanh(s_t) \\ y_t &= h_t \end{aligned} \tag{20}$$

W każdym kroku czasowym znany jest gradient funkcji kosztu wzlgędem wyjścia $\frac{dL}{dy_t} = \frac{dL}{dh_t}$, który pochodzi z dwóch źródeł:

- właściwego wyjścia przekazywanego do kolejnej warstwy,
- wyjścia przekazywanego do kolejnego kroku czasowego.

W ten sposób i-te wyjście w t-tym kroku czasowym przedstawia się następująco:

$$\begin{aligned} h_{t|i} &= tanh(s_{t|i}) \\ s_{t|i} &= \Sigma_{k=1}^{N_{in}} x_{t|k} w_{k,i}^{<x>} + \Sigma_{l=1}^{N_{out}} h_{t-1|l} w_{l,i}^{<h>} + b_i \end{aligned} \tag{21}$$

Propagacja przez tangens hiperboliczny została już wyznaczona wcześniej, więc $\frac{dL}{ds_t}$ jest znane i wygląda następująco:

$$\frac{dL}{ds_t} = (1 - h_t^2).\frac{dL}{dh_t} \tag{22}$$

Natomiast gradienty funkcji kosztu względem poszczególnych parametrów, poprzedniego kroku czasowego oraz wejścia do warstwy można wyznaczyć całkowicie analogicznie jak w przypadku warstwy liniowej. Ostatecznie otrzymujemy:

$$\frac{dL}{db} = \frac{dL}{ds_t} \tag{23}$$

$$\frac{dL}{dW_x} = (x_t^T \frac{dL}{ds_t})^T \tag{24}$$

$$\frac{dL}{dW_h} = (h_{t-1}^T \frac{dL}{ds_t})^T \tag{25}$$

$$\frac{dL}{dx_t} = \frac{dL}{ds_t} W_x \tag{26}$$

$$\frac{dL}{dh_{t-1}} = \frac{dL}{ds_t} W_h \tag{27}$$

Przy czym odpowiednie transpozycje są podyktowane sposobem przechowywania poszczególnych danych w macierzach.

W ten sposób gradient jest propagowany wstecz do poprzedniej warstwy ($\frac{dL}{dx_t}$) oraz do poprzednich kroków czasowych ($\frac{dL}{dh_{t-1}}$), natomiast poprawki do parametrów są akumulowane przez wszystkie kroki czasowe.

Analogicznie jak w przypadku warstwy liniowej, podane równania pozwalają na zrównoleglenie obliczeń dla całej porcji danych (ang. batch), poprzez zastąpienie odpowiednich wektorów macierzami.

3 Automatyczne rozpoznawanie mowy

W poniższym rozdziale wprowadzony został problem automatycznego rozpoznawania mowy. Opisano także stosowane w praktyce metody oraz metryki wykorzystywane do ewaluacji.

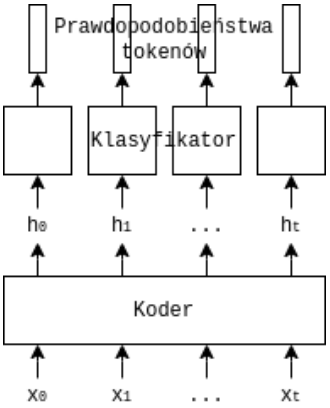
W ogólności działanie systemu automatycznego rozpoznawania mowy powinno polegać na zwróceniu najbardziej prawdopodobnej sekwencji słów na podstawie otrzymanego na wejściu sygnału mowy. Obecnie stosowane systemy można właściwie podzielić na dwie kategorie: systemy klasyczne oraz systemy bazujące na głębokich sieciach neuronowych.

3.1 Systemy automatycznego rozpoznawania mowy

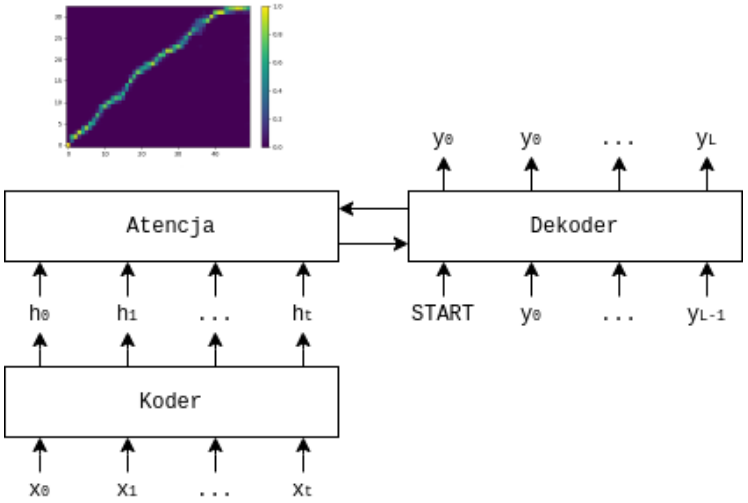
Metody klasyczne bazują na HMM (ang. Hidden Markov Model), czyli na systemach probablistycznych polegających na założeniu, że modelowany proces jest procesem Markova. Co oznacza, że rozpoznawanie mowy jest modelowane jako proces, w którym prawdopodobieństwo przewidzenia każdego kolejnego tokenu (słowa, fonemu, litery) zależy tylko od poprzedniego tokenu. Jest to oczywiście przybliżenie, jednak w praktyce sprawdza się ono bardzo dobrze i przez wiele lat pozwalało na tworzenie najlepszych systemów do automatycznego rozpoznawania mowy. Standardowo system rozpoznawania mowy bazujący na HMM jest podzielony na trzy moduły: pierwszy (akustyczny), mapujący cechy akustyczne sygnału na fonemy, drugi, mapujący fonemy na odpowiadające im litery oraz ostatni (języka), mapujący litery na słowa. [17]

W ostatnim czasie nastąpił jednak znaczący rozwój technik bazujących na głębokich sieciach neuronowych, w szczególności systemów E2E (ang. End-to-End), które pozwalają na uzyskanie systemów rozpoznawania mowy o znacząco wyższej jakości niż systemy klasyczne. Najbardziej popularne są tutaj architektury bazujące na funkcji kosztu CTC (ang. Connectionist Temporal Classification) oraz modele seq2seq wykorzystujące mechanizm atencji. [17]

Model bazujący na CTC został schematycznie przedstawiony na rysunku 4. Składa się on z warstw kodujących wejściowe cechy akustyczne (ang. encoder), które następnie są klasyfikowane. W ten sposób produkowane są prawdopodobieństwa przypisania posczególnych ramek cech akustycznych do poszczególnych elementów zbioru tokenów z dodatkowym elementem tzw. blank. Wprowadzenie tego dodatkowego elementu pozwala na wyrównywanie ze sobą sekwencji o różnych długościach (długość sekwencji wejściowej musi być większa niż długość sekwencji wyjściowej) i wykorzystywane jest w trakcie obliczania funkcji kosztu CTC. Dzięki temu, że funkcja kosztu CTC maksymalizuje wszystkie możliwe wyrównania sklasyfikowanej sekwencji wejściowej oraz sekwencji docelowej, taki model może być trenowany na danych bez przeprowadzonej wcześniej segmentacji, tzn. bez informacji o tym gdzie w nagraniu (cechach akustycznych) kończą się i zaczynają poszczególne tokeny. Pozwala to na znaczące uproszczenie procesu kontrukcji systemu automatycznego rozpoznawania mowy. Więcej informacji o architekturze bazującej na CTC można znaleźć w [18]. [17]



Rysunek 4. Schemat systemu automatycznego rozpoznawania mowy bazującego na CTC.

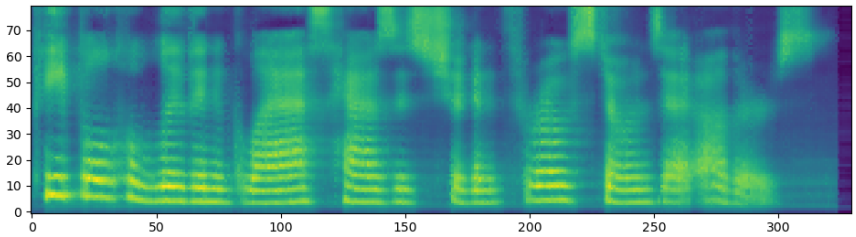


Rysunek 5. Schemat systemu automatycznego rozpoznawania mowy bazującego na architekturze seq2seq z atencją.

Atencja została zaproponowana w [19] oraz w [20], jako dodatkowy mechanizm w modelach typu koder-dekoder, umożliwiający automatyczne wyszukiwanie części reprezentacji sekwencji wejściowej, która jest najbardziej istotna w danym kroku dekodowania. Model seq2seq wykorzystujący atencję został schematycznie przedstawiony na rysunku 5. Działanie takiej architektury polega na zakodowaniu sekwencji wejściowej cech akustycznych, które następnie są przekazywane poprzez mechanizm atencji do dekodera w kolejnych krokach procesu dekodowania. Atencja jako wejście poza kontekstem pochodzącym z kodera przyjmuje także kontekst z dekodera i na ich podstawie uczy się wybierać najbardziej znaczący w danym momencie zakodowany fragment sekwencji wejściowej. Produkowane jest w ten sposób miękkie wyrównanie (ang. soft alignment) między dwiema sekwencjami, przykład takiego wyrównania jest przedstawiony na rysunku 5. [17]

3.2 Ekstrakcja cech akustycznych

Ze względu na swój charakter (wiele tysięcy jednowymiarowych próbek) sygnał dźwiękowy (mowy) jest najczęściej wstępnie przetwarzany przed podaniem go do modeli rozpoznawania mowy. W szczególności ekstrahowane są z niego cechy akustyczne. Popularne jest stosowanie MFCC (ang. Mel-frequency Cepstral Coefficients) lub spektrogramów w skali mel (ang. mel-spectrogram). Przykład cech akustycznych znajduje się na rysunku 6. W literaturze można się też spotkać ze specjalnymi modelami ekstrakcji cech z sygnału dźwiękowego przeznaczonymi do trenowania razem z docelowym modelem. Przykładem jest tutaj LEAF [21].



Rysunek 6. Przykładowe cechy akustyczne (mel-spectrogram) wyznaczony dla sygnału mowy.

3.3 Ewaluacja systemów automatycznego rozpoznawania mowy

Najpopularniejszą metryką służącą do ewaluacji systemów automatycznego rozpoznawania mowy jest WER (ang. Word Error Rate). Do obliczenia tej metryki potrzebny jest referencyjny tekst dla danego sygnału mowy. WER wyznacza się

jako liczbę modyfikacji potrzebnych do sprowadzenia wyjścia systemu do referencji podzieloną przez liczbę słów w referencji. Przy czym do modyfikacji zalicza się: podmianę słowa, usunięcie słowa oraz wstawienie słowa. Im mniejszy jest WER, tym system jest lepszej jakości.

Można także stosować zmodyfikowaną wersję WER dla pojedynczych znaków (fonemów), wtedy mamy w praktyce do czynienia z odległością Levenshteina.

W ostatnim czasie pojawiła się także propozycja metryki do ewaluacji systemów rozpoznawania mowy pod względem semantycznym [22]. Bazuje ona na kodowaniu zdania referencyjnego oraz wyjścia systemu przy pomocy popularnych ostatnio modeli językowych z rodziny BERT [23], a następnie porównaniu kodowań przy pomocy podobieństwa cosinusowego (ang. cosine similarity).

4 Rozwiązanie

Niniejszy rozdział zawiera opis rozwiązania zagadnienia projektowego. W szczególności doprecyzowana została analizowana architektura sieci neuronowej. Opisano także proces przygotowania danych, zarówno akustycznych jak i tekstowych. Omówiono metody ewaluacji rozwiązania oraz implementację od strony technicznej.

4.1 Architektura sieci neuronowej

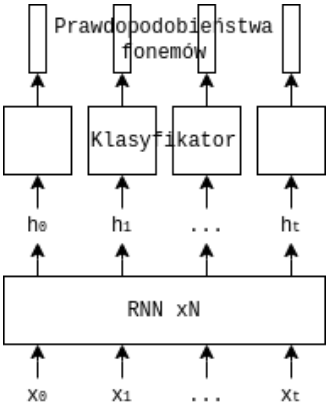
W projekcie przeanalizowane zostało działanie prostej sieci rekurencyjnej, klasyfikującej poszczególne ramki cech akustycznych do odpowiadających im fonemów. Sieć ta składała sie dwóch podstawowych modułów: kodera (warstwy rekurencyjnej) i klasyfikatora (warstwy w pełni połączonej). Sieć ta została schematycznie przedstawiona na rysunku 7. Przeanalizowane zostały różne konfiguracje tej architektury, w szczególności pod względem rozmiaru warstwy rekurencyjnej. Niestety ze względu na ograniczony czas nie udało się przeprowadzić analizy sieci składającej się z więcej niż jednej warstwy rekurencyjnej.

4.2 Zbiór danych

Sieć była trenowana na dostępnym publicznie zbiorze danych TIMIT [24], który zawiera nagrania angielskich zdań czytanych przez 630 osób w ośmiu różnych dialektach amerykańskiego angielskiego. Przy czym każda osoba przeczytała do 10 zdań. Zbiór danych został pobrany z [25].

Korpus był dostarczony w formie podzielonej na zbiór treningowy i testowy. Warto tutaj zaznaczyć, że zbiory te były rozłączne pod względem osób czytających zdania. Dostępne były także transkrypcje nagrań zarówno w formie tekstowej jak i fonetycznej oraz segmentacja dla transkrypcji fonetycznej. Nagrania zostały udostępnione w formacie *wav* z próbkowaniem $16kHz$, 16-bitową precyzją i jednym kanałem.

Zbiór danych TIMIT jest dobrze znanym i stosunkowo niewielkim zbiorem danych przeznaczonym do badania systemów automatycznego rozpoznawania



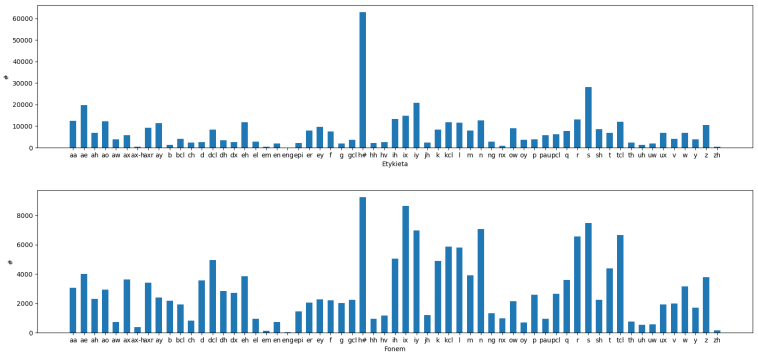
Rysunek 7. Schemat analizowanego w projekcie modelu.

mowy. Przez to w ostatnim czasie jest coraz rzadziej wykorzystywany w profesjonalnych badaniach. Obecnie bardziej popularne takie zbiory danych jak Librispeech [26] czy Common Voice [27].

Jako wejście do sieci neuronowej wykorzystane zostały spektrogramy w skali mel (ang. mel-spectrogram) wyznaczone na podstawie nagrań. Do obliczeń użyty był pakiet języka Python *librosa*. Wyekstrahowane zostały cechy o 80 wymiarach, przy szerokości okna (ang. window length) równej 1024 i długości kroku (ang. hop length) równej 480, co pozwoliło na ektrakcję jednej ramki na każde $30ms$ nagrania. Dokładne parametry oraz kod użyty do ekstrakcji cech znajduje się w skrypcie `extract_feats.py`.

Na podstawie segmentacji dostarczonej razem ze zbiorem danych do każdej ramki cech akustycznych został przyporządkowany odpowiadający jej fonem, który stanowił jej etykietę w zadaniu klasyfikacji. Na rysunku 8 przedstawiony został rozkład częstości występowania fonemów w transkrypcjach oraz w przygotowanych etykietach. Można zaobserwować, że zamiana fonemów na etykiety uwidoczniła fonemy, których wymowa trwa dłużej. Szczególnym przypadkiem jest tutaj fonem `h#`, który reprezentuje ciszę na początku i na końcu nagrania. Można zauważyć, że nagrania zaczynają się i kończą relatywnie dużymi fragmentami ciszy, które można by skrócić na etapie wstępnego przetwarzania danych. Jednak w przypadku tego projektu zdecydowano się na nieingerowanie w dane w ten sposób. Kolejną obserwacją jest fakt, że rozwiązywany tutaj problem klasyfikacji ma bardzo niezrównoważone klasy, co na pewno będzie miało wpływ na ostateczny wynik tak prostego modelu jak standardowa sieć rekurencyjna.

Następnie z danych treningowych zostały wylosowane dane do walidacji. W celu przyspieszenia treningu skonstruowane zostały porcje danych (ang. batches) o arbitralnie przyjętym rozmiarze 21 nagrań. Przy czym do pojedynczej porcji trafiały dane o podobnej długości nagrania. Następnie wykonane zostało wyrównanie danych w porcji. Cechy zostały uzupełnione ramką o najniższej średniej wartości, co odpowiada uzupełnieniu nagrania ciszą, natomiast etykiety zostały



Rysunek 8. Schemat podziału kodu na moduły z zaznaczonymi zależnościami. Ze względu na rozmiar, rysunek został także dołączony do projektu w postaci pliku PNG o lepszej rozdzielczości.

uzupełnione fonemem `h#`, który jest wykorzystywany do oznaczania ciszy na początku i na końcu nagrania. Kod wykorzystany w tym wstępnym przetwarzaniu znajduje się w module `data.py`. Ostateczne rozmiary poszczególnych zbiorów danych zostały przedstawione w tabeli 1.

Tablica 1. Rozmiar wykorzystanych zbiór danych: treningowego, walidacyjnego i testowego.

Zbiór Danych	Liczba nagrań	Liczba porcji danych
treningowy	4158	198
walidacyjny	462	22
testowy	1680	1680

4.3 Ewaluacja

Do treningu oraz ewaluacji modelu wykorzystana została standardowa metodyka podziału danych na zbiory treningowy, walidacyjny oraz testowy. Przy czym zbiór testowy był rozłączny pod względem tożsamości osób czytających zdania, natomiast zbiór walidacyjny był wylosowany spośród zbioru treningowego.

Ponieważ rozważane zadanie było klasyfikacją, ewaluacja została dokonana przy użyciu dokładności (ang. accuracy). Dodatkowo wykorzystana została także macierz pomyłek. Pomimo tego, że planowana była także ewaluacja przy pomocy TER (ang. token error rate) na zachłannie zdekodowanych sekwencjach, to ze względu na to, że osiągnięta dokładność nie była zadowalająca, zrezygnowano z tego podejścia.

Dodatkowo w celu weryfikacji poprawności rozwiązania przeprowadzone zostały proste testy polegające na klasyfikacji sztucznie zaprojektowanych prostych zbiorów danych. Wszystkie te testy zostały pomyślnie zakończone.

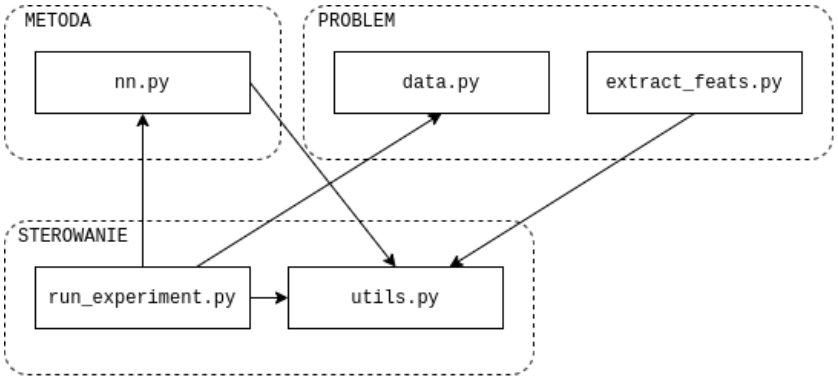
Nistety ze względu na ograniczony czas nie udało się przeprowadzić analogicznego eksperymentu, przy użyciu gotowych pakietów do implementacji sieci neuronowych. Nie udało się także zaimplementować sprawdzania gradientu (ang. gradient checking) w celu weryfikacji poprawności implementacji. Są to dalsze możliwe kierunki rozwoju tego projektu.

4.4 Implementacja

Projekt został zaimplementowany w języku Python z wykorzystaniem pakietu *NumPy* ułatwiającego działania na tablicach (macierzach, tensorach) oraz pakietów *librosa* i *scipy* do niskopoziomowego przetwarzania sygnałów mowy.

Implementacja została podzielona na moduły zgodnie z wymaganiami projektowymi, (rys. 9):

- moduł `nn.py` stanowi implementację **metody**, czyli prostej biblioteki obsługującej sieci neuronowe, którą można stosować do rozwiązywania różnych problemów klasyfikacji i regresji;
- moduł `data.py` oraz skrypt `extract_feats.py` stanowią implementację **problemu**, gdyż zapewniają one odpowiednie metody związane z przetwarzaniem danych w tym konkretnym problemie klasyfikacji cech akustycznych;
- skrypt `run_experiment.py` oraz moduł `utils.py` są implementacją **sterowania**, gdyż to przy ich pomocy przeprowadzane są eksperymenty oraz to one zawierają funkcje wejścia/wyjścia.



Rysunek 9. Schemat podziału kodu na moduły z zaznaczonymi zależnościami.

Dokładniejsza dokumentacja zaimplementowanych funkcji i obiektów znajduje się w kodzie w formie *docstringów* oraz w pliku `README.md` dołączonym do kodu źródłowego.

Moduł `nn.py` stanowi implementację prostej biblioteki obsługującej sieci neuronowe. Sposób korzystania z niej bazuje luźno na pakietach takich jak *PyTorch*, czy *Keras*. Głównym obiektem jest `Model`, który pozwala na zdefiniowanie sieci przez podanie listy warstw, które mają się w tej sieci znajdować, funkcji kosztu oraz ewentualnych metryk. Dodatkowo obiekt ten odpowiada za trening sieci oraz za wykonywanie predykcji. W pakiecie zostały zaimplementowane następujące warstwy:

- `Linear` - warstwa w pełni połączona,
- `RNN` - standardowa warstwa rekurencyjna,
- `ReLU` - warstwa reprezentująca funkcję aktywacji ReLU (ang. Rectified Linearu Unit),
- `Softmax` - warstwa reprezentująca funkcję aktywacji softmax.

Dodatkowo zaimplmentowane zostały dwie funkcje kosztu:

- `MSELoss` - średni błąd kwadratowy,
- `CrossEntropyLoss` - entropia krzyżowa.

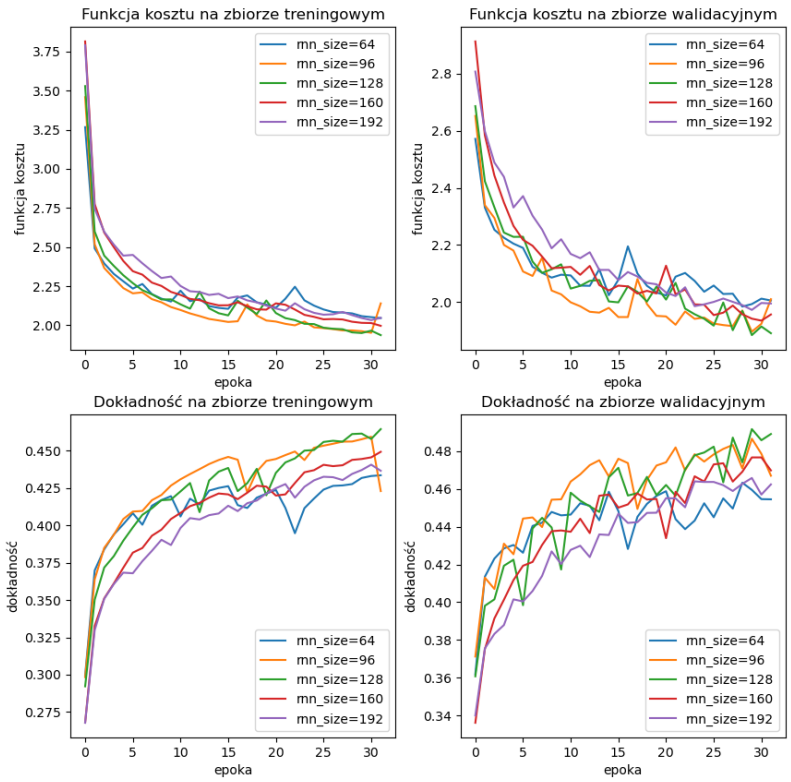
Zaimplementowano także metrykę `Accuracy`, czyli dokładność klasyfikacji. Do treningu sieci zastosowano algorytm wstecznej propagacji błędów SGD (ang. stochastic gradient descent) w wersji z momentem bezwładności (ang. momentum).

5 Wyniki

W ramach przeprowadzonych eksperymentów model został wytrenowany i zewaluaowany dla pięciu rozmiarów warstwy rekurencyjnej (64, 96, 128, 160 i 192). Rozmiary były wybrane arbitralnie na podstawie przeszłych doświadczeń autora z systemami przetwarzającymi sygnały mowy (TTS i ASR). Niestety ze względu na ograniczony czas nie udało się wykonać eksperymentów na modelach głębszych, składających się z więcej niż jednej warstwy rekurencyjnej.

Na wykresach na rysunku 10 przedstawiono przebieg funkcji kosztu oraz uzyskiwanej dokładności dla zbiorów treningowego i walidacyjnego w trakcie treningu dla wszystkich wersji modelu. Do treningu zastosowano dość standardowe wartości współczynnika uczenia $(0,0001)$ i momentu bezwładności $(0,8)$, również wynikające ze wcześniejszych doświadczeń autora w pracy z systemami przetwarzającymi sygnał mowy. Przeprowadzono kilka prób treningu z innymi parametrami, jednak nie przynosiły one lepszych rezultatów, więc zrezygnowano z opisywania ich w raporcie. Modele były trenowane przez 32 epoki, wartość tę ustalono metodą prób i błędów. Dłuższy trening skutkował wypłaszczeniem wartości funkcji kosztu oraz dokładności, a więc nie przynosił żadnych rezultatów, poza wnoszeniem ryzyka zbytniego dopasowania modelu. Należy zwrócić uwagę, że zbiór walidacyjny ma w przypadku tego problemu bardziej podobny rozkład danych do zbioru trenującego niż zbiór testowy. Z tego powodu, pomimo braku widocznego zbytniego dopasowania w dłuższym treningu, ono może występować.

Wyniki dokładności na zbiorach treningowym i walidacyjnym osiągają około $0,45$, co wydaje się raczej niską wartością. Należy jednak pamiętać, że rozważany problem jest klasyfikacją na 61 klas (liczba fonemów) o niezrównoważonym



Rysunek 10. Wykresy przedstawiające wartość funkcji kosztu oraz dokładność dla zbiorów treningowych i walidacyjnych w czasie treningu dla wszystkich wersji modelu.

rozkładzie, a do jego rozwiązywania wykorzystywany jest bardzo prosty model (standardowa jednokierunkowa sieć rekurencyjna). Biorąc te fakty pod uwagę, nie należało się spodziewać tutaj wyników na poziomie $0,9$, więc osiągane wyniki uznano za zadowalające.

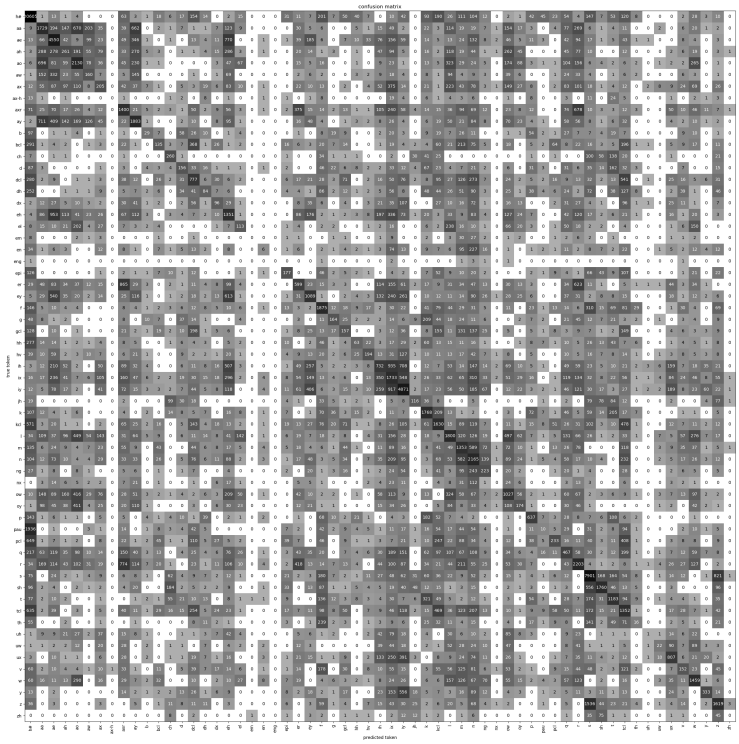
Na rysunku 10 można zaobserwować, że model z warstwą rekurencyjną o rozmiarze 128 osiąga najlepsze wyniki, różnice pomiędzy modelami nie są jednak na tyle duże żeby nie uznawać ich za wpływ losowości.

Wyniki ewaluacji wszystkich wersji modelu na zbiorze testowym zostały zebrane w tabeli 2. W tym przypadku także sieć z warstwą rekurencyjną o rozmiarze 128 osiągnęła najlepszy wynik. Jednak także w tym przypadku różnice nie są duże. Osiągane wyniki są kilka punktów procentowych niższe niż w przypadku danych treningowych i walidacyjnych. Zjawisko to można wytłumaczyć faktem, że zbiory osób występujących w danych treningowych/walidacyjnych oraz w danych testowych są rozłączne. Problem ten często pojawia się także w komercyjnych systemach rozpoznawania mowy. Aby mu przeciwdziałać stosuje się różne techniki rozszerzeania zbioru danych (ang. augmentation).

Tablica 2. Wyniki dokładności uzyskane na zbiorze testowym dla wszystkich wersji modelu.

Rozmiar sieci rekurencyjnej	Dokładność [%]
64	41,40
96	42,14
128	44,89
160	43,10
192	42,52

Ze względu na niezrównoważenie klas, wykorzystanie jako metryki dokładności (ang. accuracy) może nieść za sobą pewne niebezpieczeństwa. W szczególności przypadek, w którym model zawsze zwraca najbardziej liczną etykietę może uzyskiwać wysoką wartość dokładności. Z tego powodu zdecydowano się na wyznaczenie macierzy pomyłek (ang. confusion matrix) dla wyników na zbiorze testowym dla modelu o rozmiarze warstwy rekurencyjnej równym 128. Macierz ta została przedstawiona na rysunku 11. Można zauważyć ciemniejszy kolor wzdłuż przekątnej macierzy, co oznacza, że model zwraca różne etykiety, nie tylko tę najczęściej występującą w zbiorze treningowym. Inną ciekawą obserwacją jest fakt występowania grup fonemów, które model często myli ze sobą. Przykładami takich grup są fonemy `aa`, `ae`, `ah`, `ao`, `aw`, `ax` oraz `ih`, `ix`, `iy`, których zapis sam sugeruje, że ich wymowa może być podobna w obrębie każdej grupy. Można zuważyć także, że fonemy które rzadko występowały w zbiorze treningowym są bardzo rzadko zwracane jako wyjście modelu (białe pionowe pasy na macierzy pomyłek), przykładem są tutaj `em`, `eng` czy `nx`.



Rysunek 11. Macierz pomyłek na zbiorze testowym dla modelu $rnn_size = 128$. Kolory są przypisane w skali logarytmicznej. Ze względu na rozmiar, rysunek został dołączony do projektu także w formacie PNG jako oddzielny plik o większej rozdzielczości.

6 Podsumowanie

Celem tego projektu było rozwiązanie problemu klasyfikacji ramek z cechami akustycznymi sygnału mowy do odpowiadających im fonemów przy pomocy samodzielnie zaimplementowanej sieci rekurencyjnej. Cel projektu został zrealizowany. Została zaimplementowana prosta biblioteka pozwalająca budować, trenować i ewaluować sieci neuronowe. Wytrenowano i przenalizowano kilka wersji modelu klasyfikującego. Dodatkowo, w raporcie zostały omówione wszystkie zagadnienia potrzebne do implementacji oraz zagadnienia powiązane z projektem. W szczególności skupiono się na treningu sieci neuronowych z uwzględenieniem sieci rekurencyjnych oraz na automatycznym rozpoznawaniu mowy.

Literatura

1. Google. Google Assistant. https://assistant.google.com/.
2. Samsung. Bixby. https://www.samsung.com/global/galaxy/what-is/bixby/.
3. Apple. Siri. https://www.apple.com/siri/.
4. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
5. Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.
6. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.
7. Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.
8. Ahmed, Bapi Surampudi, V. S. Chandrasekhar Pammi, and Krishna Miyapuram. Application of multilayer perceptron network for tagging parts-of-speech. page 57, 01 2002.
9. Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.
10. Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990.
11. Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.
12. Hemanth Pedamallu. RNN vs GRU vs LSTM. https://medium.com/analytics-vidhya/rnn-vs-gru-vs-lstm-863b0b7b1573.
13. Jay Alammar. The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/.
14. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
15. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.
16. Justin Johnson. Backpropagation for a Linear Layer. http://cs231n.stanford.edu/handouts/linear-backprop.pdf.
17. Włodzimierz Kasprzak. Sieci neuronowe w percepcji mowy i obrazu. https://mgr.okno.pw.edu.pl/pluginfile.php/19400/mod_folder/intro/ITO-C6.pdf.

18. Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks. volume 2006, pages 369–376, 01 2006.
19. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.
20. Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.
21. Neil Zeghidour, Olivier Teboul, Félix de Chaumont Quitry, and Marco Tagliasacchi. LEAF: A learnable frontend for audio classification. *CoRR*, abs/2101.08596, 2021.
22. Suyoun Kim, Abhinav Arora, Duc Le, Ching-Feng Yeh, Christian Fuegen, Ozlem Kalinli, and Michael L. Seltzer. Semantic distance: A new metric for ASR performance analysis towards spoken language understanding. *CoRR*, abs/2104.02138, 2021.
23. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
24. John Garofolo, Lori Lamel, William Fisher, Jonathan Fiscus, David Pallett, Nancy Dahlgren, and Victor Zue. TIMIT Acoustic-Phonetic Continuous Speech Corpus, 1993.
25. DeepAI. TIMIT Dataset. https://deepai.org/dataset/timit.
26. Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. pages 5206–5210, 04 2015.
27. Mozilla. Common Voice. https://commonvoice.mozilla.org/.