Sieci neuronowe rozpoznające trifony w procesie analizy mowy

(Neural network recognizing triphones in speech analyze)

Aleksander Sas

Praca magisterska

Promotor: dr Paweł Rychlikowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

21 września 2017

Streszczenie

W niniejszej pracy magisterskiej przedstawiono nowe podejście do wykorzystanie splotowych sieci neuronowych przy rozpoznawaniu mowy. W pierwszej części omówiono klasyczne metody wykorzystujące wielowymiarowe mikstury Gaussowskie do estymacji prawdopodobieństwo stanów modelu Markowa. W drugiej zaproponowano i opisano podejście, w którym mikstury są zastąpione splotową siecią neuronową rozpoznającą bardzo mocno ograniczony zbiór trifonów. Przedstawiono zarówno sposób pozyskiwania danych do treningu sieci, jak i metodą budowania wektorów cech, będących wejściem sieci. W ostatniej części zaprezentowano wyniki przeprowadzonych eksperymentów. Główny nacisk położono na przebadanie wpływu liczby rozpoznawanych przez sieć stanów na skuteczność rozpoznawania mowy

---

The following paper presents a new approach to using neural networks in speech recognition. In the first part classic approach, using multidimensional Gaussian mixture to estimate probability of hidden Markov states, is described. The second part presents new method using convolutional neural network recognizing limited set of triphones, instead of Gaussian mixture. Both, method to generate training data and building of input feature vector for network are described. In the last part, experiment results are presented. The main focus in experiments is on examination impact of number of recognized states to speech recognition accuracy.

Spis treści

Rozdział 1.

Wprowadzenie - opis problemu

Automatyczne rozpoznawanie mowy, w skrócie **ARM**, polega na rozpoznaniu i zapisaniu nagranych słów. Jest to coraz częściej i powszechniej stosowana technologia, która znajduje szerokie zastosowanie w motoryzacji, urządzeniach mobilnych, administracji państwowej i medycznej. Jest szczególnie przydatna przy sporządzaniu opisów, tekstowych i protokołowaniu wszelkiego rodzaju obrad. Systemy do rozpoznawanie mowy działają na dwa sposoby:

- offline, w którym system wczytuje zbiór nagrań dźwiękowych i dla każdego z nich generuje transkrypcję,
- online, w którym system na bieżąco rejestruję dźwięk i wypisuje transkrypcję.

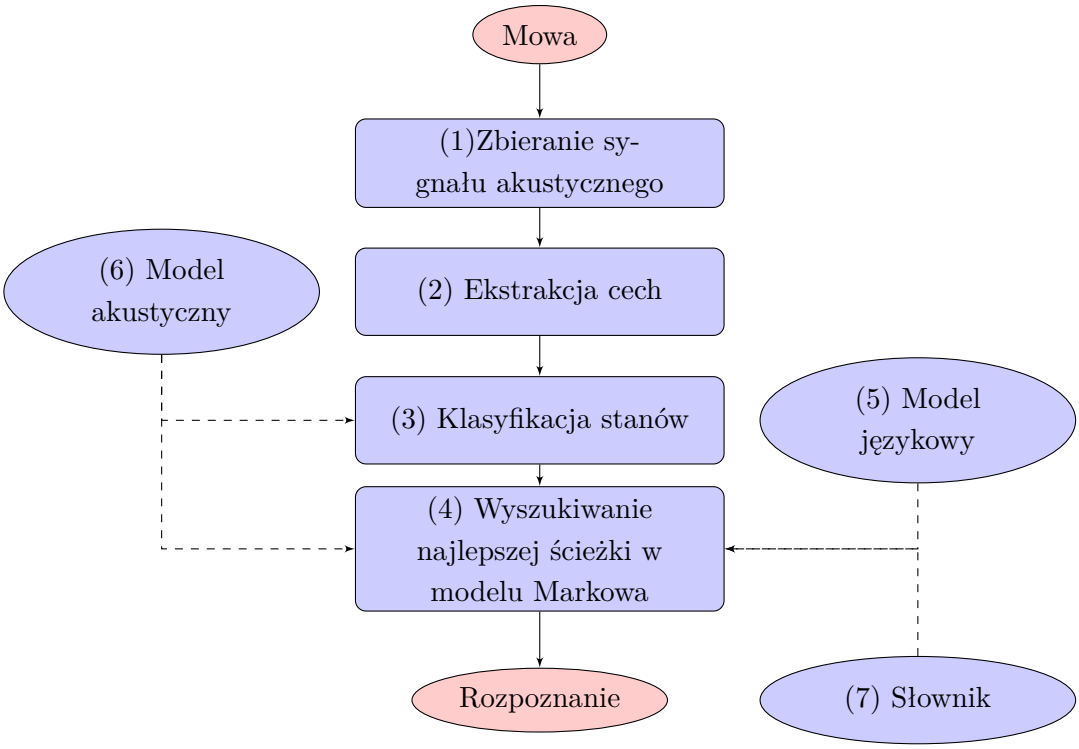
Oba tryby wymagają możliwie najlepszej skuteczności rozpoznawania, jednak tryb online jest szczególnie wymagający, gdyż dodatkowo wymaga szybkiego działania. Rozpoznawanie w trybie online powinno się odbywać w czasie zbliżony do rzeczywistego.

Rozwój kart graficznych wraz z pojawieniem się technologii CUDA oraz nowe rozwiązania z dziedziny sieci neuronowych zaowocowały w ostatnich latach pojawieniem się klasyfikatorów radzących sobie znacznie lepiej niż wcześniej stosowane algorytmy. W związku z tym coraz więcej badaczy próbuje stosować sieci neuronowe w celu poprawy jakości systemów rozpoznających mowę. Można spotkać wiele różnych konfiguracji, zarówno łączących konwencjonalne systemy, jaki i autonomiczne sieci rekurencyjne. Jedną z pierwszych skutecznych prób zastosowania sieci neuronowych przy rozpoznawaniu mowy zaprezentowano w artykule [1]. Autorzy wykorzystują w nim klasyczną sieć typy *feed-forward* do estymacji prawdopodobieństw unifonów. W kolejnych latach zaproponowano zastąpienie klasycznych sieci architekturą splotową oraz zamianę szeroko stosowanych cech *MFCC* na *MFSC*. W artykule [8] zaprezentowano właśnie takie podejście. Autorzy, inspirując się wykorzystaniem głębokich, splotowych sieci neuronowych przy rozpoznawaniu obrazów, proponują przeniesie tej samej architektury do systemów rozpoznających mowę.

Celem niniejszej pracy magisterskiej jest przetestowanie możliwości wykorzystania sieci neuronowych w procesie rozpoznawania mowy. Duży nacisk położony jest na możliwość rozpoznawanie trifonów, co jest rozwinięciem podejścia proponowanego w publikacji [1]. Przeprowadzono szereg eksperymentów w celu sprawdzenia możliwości poprawy skuteczności rozpoznawania mowy, poprzez zastosowanie sieci neuronowych rozpoznających ograniczony zbiór trifonów. Pokazano, jak liczba trifonów wpływa na skuteczność oraz porównano otrzymane wyniki z klasycznymi modelami gaussowskimi oraz sieciami neuronowymi rozpoznającymi unifony.

Rozdział 2.

Proces automatycznego rozpoznawania mowy (ARM)



Rysunek 2.1: Etapy automatycznego rozpoznawania mowy

Niniejszy rozdział bazuje na podejściu do rozpoznawania mowy przedstawionym w [4].

Automatyczne rozpoznawanie mowy możemy formalnie zdefiniować, jako znajdowanie ciągu słów $\hat{W}$ z pewnego zbioru V, o maksymalnym prawdopodobieństwie,

pod warunkiem obserwacji O. Zbiór V odpowiada blokowi (7) z rysunku 2.1.

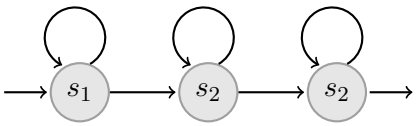
$$\hat{W} = \arg\max_{W \in V^*} P(W \mid O) \tag{2.1}$$

Niestety wyliczanie formuły 2.1 okazuję się niewygodne, ale korzystając ze wzory Bayesa możemy dojść do postaci 2.2, która jest już wygodny do obliczenia.

$$\hat{W} = \arg\max_{W \in \Sigma^*} P(W \mid O) = \arg\max_{W \in \Sigma^*} \frac{P(O \mid W)P(W)}{P(O)} = \arg\max_{W \in \Sigma^*} P(O \mid W)P(W) \tag{2.2}$$

Ostatnie przejście wynika z faktu, że $P(O)$ nie zależy od W, zatem możne je pominąć. Implementując formułę 2.2 możemy wydzielić 5 kluczowych etapów, są one zilustrowane na rysunku 2.1. Opis poszczególnych bloków ze schematu znajduje się w kolejnych podrozdziałach.
Pierwszy etap, określony jako *Zbieranie sygnału akustycznego* (blok (1) na rysunku 2.1), jest realizowany sprzętowo poprzez peryferyjne urządzenia. Obejmuje analogowe przetwarzanie sygnału, cyfryzację oraz opcjonalny post-processing wykonywany przez kartę dźwiękową. Na tym etapie ważne jest, aby dostroić poziom dźwięku, tak aby uniknąć przesterowań, dobrać częstotliwość próbkowania i zakres częstotliwości[1]. Niepoprawna konfiguracja tego etapu może skutkować zakłóceniami oraz wycięciem informacji, które mogą być istotne na kolejnych etapach przetwarzania, a w konsekwencji obniżeniem skuteczności rozpoznawania. W niniejszej pracy wykorzystany był gotowy, powszechnie dostępny korpus nagrań (patrz rozdział 4.1.), dlatego nie będę się skupiał na tym etapie.

2.1. Fonemy



Rysunek 2.2: Reprezentacja fonemów

Fonemy są podstawową koncepcją przy modelowaniu dźwięków w mowie. Typowo rozróżniają wypowiadane dźwięki, takie jak miedzy innymi głoski dźwięczne, bezdźwięczne, nosowe czy nieme. Jednak w niniejszych rozważaniach rozszerzymy definicję fonemu o dodatkowy symbol *ciszy*, oznaczony jako *sil*. W tabeli 2.3 znajduje się lista fonemów zamodelowanych w wykorzystywanych przez system *Magic Scribe*

[1] W testowym korpusie *Clarin* częstotliwość próbkowania wynosiła 16 kHz, częstotliwości powyżej 8 kHz były wycięte.

a	o~	b	c	cz	ć	d	dz
dź	dż	e	e~	f	g	gˆ	h
i	j	k	kˆ	l	ł	m	n
nn	ń	o	p	r	s	sz	ś
t	u	v	y	z	ź	ż	sil

Rysunek 2.3: Lista fonemów modelowanych w systemie *Magic Scribe*.

modelach akustycznych, łącznie jest ich 40. Należy podkreślić, że wpisy z tabeli 2.3 nie są literami, lecz identyfikatorami fonemów. Każdy zamodelowany fonem ma swój zestaw parametrów, które go opisują. Parametry te tworzą *model akustyczny* odpowiadający blokowi 6 z rysunku 2.1. Słowa, które mają być rozpoznawane przez system, muszą mieć przypisaną transkrypcję na fonemy. Lista słów wraz z transkrypcją znajduje się w *słowniku*, który odpowiada blokowi (7) na rysunku 2.1. Aby umożliwić poprawne rozpoznawanie, każda forma gramatyczne danego słowa musi się znaleźć w słowniku, gdyż jest traktowana jak niezależne słowo. Konsekwencją tego jest bardzo duża liczba wpisów w słownikach dla języków o rozbudowanej fleksji, takich jak język polski. W wykorzystanym słowniku, zawierającym 1209017 wpisów, transkrypcje zostały zbudowane na podstawie zasad sformułowanych przez Marię Steffen-Batogową w książce *Automatyzacja transkrypcji fonematycznej tekstów polskich* [13]. Zasady te nie są jednak jednoznaczne i w sytuacji, gdy dla jednego słowa można wyprowadzić wiele różnych transkrypcji, wszystkie zostają umieszczone w słowniku. Przykładem wyrazu o wielu transkrypcjach jest słowo *wystrzygł*, które zgodnie z regułami można zapisać między innymi jako *v y s t sz y k* lub *v y s t sz y k ł*.

2.2. Modelowanie fonemów, modele kontekstowe i bezkontekstowe

Przy modelowaniu fonemów wyróżnia się trzy fazy:

- początkową, podczas której aparat mowy zmienia swój kształt,
- środkową, podczas której aparat mowy jest już w ustabilizowanej pozycji,
- końcową, podczas której aparat mowy przechodzi do układu dla kolejnego fonemu, ale dźwięk jaki wydaje jest jeszcze bliższy aktualnemu fonemowi.

Uwzględniając powyższe spostrzeżenia, w systemie *HTK*, wszystkie fonemy modeluje się za pomocą trzech stanów, tak jak na rysunku 2.2. Ze stanu można przejść jedynie do następnego stanu lub powrócić do samego siebie. Dzięki przejściu zwrotnemu, możliwe jest modelowanie dźwięków o różnej długości.
Rozwinięciem fonemów są **trifony**, uwzględniają one lewy i prawy kontekst poprzez dodanie informacji o fonemach stojących obok. Typowo trifony zapisuje się zgodnie

z notacją na rysunku 2.4, gdzie a jest poprzednim fonemem, b aktualnym, natomiast c następnym. W praktyce nie występują wszystkie kombinacje trójek fonemów, takich jak przykładowo *o-o+a*, niemniej jednak prawie połowę z nich można spotkać. W modelach wykorzystanych podczas eksperymentów występuje 27005 fonemów, z czego około 26000 to trifony, a pozostałe to uni oraz bifony.

a-b+c

Rysunek 2.4: Konwencja zapisu trifonów, składających się z fonemów a, b oraz c

W języku polskim niektóre głoski różnie się wymawia w zależności od kontekstu, czyli fonemów stojących obok. Przykładowo, inaczej brzmi głoska w w słowie *wersja*, gdzie podczas wymowy usta układają się szeroko, a inaczej w słowie *wola*, gdzie usta układają się wąsko. Model trifonowy pozwala na uchwycenie takiej różnicy, dzięki czemu umożliwia osiągnięcie wyższej skuteczności niż model unifonowy. Należy zauważyć, że korzystając z *trifonów* zamiast *unifonów* wprowadzamy dodatkową informację, zarówno do modelu językowego, jak i do modelu akustycznego, ale jednocześnie znacznie zwiększamy liczbę parametrów do estymacji i czas działania silnika rozpoznającego. W modelu *trifonowym* występuje $O(n^3)$ trifonów, gdzie n to liczba fonemów w modelu unifonowym.

2.3. Ekstrakcja cech

Ekstrakcja cech, odpowiadająca blokowi (2) z rysunku 2.1, polega na zamianie sygnału akustycznego na ciąg wektorów, które mogą być następnie sklasyfikowane na kolejnym etapie. Porównanie powszechnie stosowanych metod ekstrakcji cech takich jak *MFCC* czy *PLP* zostało opisane w artykule [6].

2.3.1. Cechy *MFCC*

Przy rozpoznawaniu mowy najczyściej wykorzystuje się cechy *MFCC* zaproponowane przez *S. Davisa* i *P. Mermelsteina* [11], składające się z 5 etapów przetwarzania:

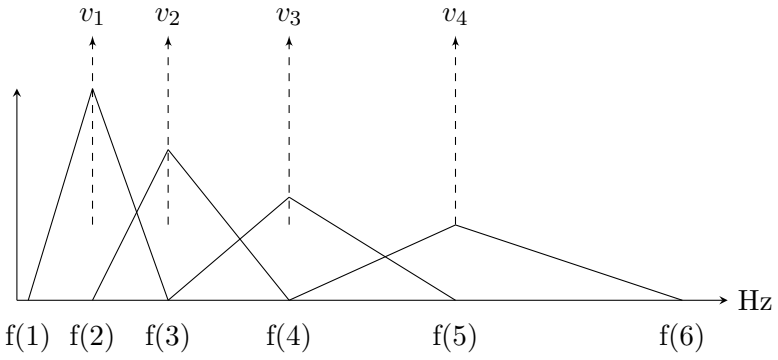
- Podział sygnału akustycznego.
- Transformata Fouriera.
- Nałożenie zestawu filtrów.
- Nałożenie logarytmu.
- Dyskretna transformata kosinusowa.

Pierwszym etapem ekstrakcji jest podzielenie sygnału na fragmenty zwane *ramkami*. Typowo ramki mają 25 ms i są przesunięte co 10 ms, przez co nachodzą na siebie. Ramki o takiej długości zawierają wystarczająco dużo informacji, a jednocześnie można założyć, że są stochastycznie stacjonarne.

W drugim etapie każdą ramkę poddaje się transformacie Fouriera. W efekcie tej operacji, otrzymuje się widmo sygnału akustycznego, opisujące ilość energii w zależności od częstotliwości.

W trzecim etapie na widmo nakłada się zestaw filtrów trójkątnych, zwanych *MEL-owym zestawem filtrów*. Filtry te są przesunięte względem siebie, każdy kolejny filtr jest coraz szerszy, co odpowiada ludzkiej percepcji. Ludzie dobrze rozróżniają sygnały o małej częstotliwości, natomiast te o wysokiej są podobnie odbierane przez ludzkie ucho. Aby określić zestaw filtrów należy podać ich liczbę, niech będzie nią N oraz zdefiniować $N+1$ częstotliwości, które wyznaczają zakresy filtrów. Załóżmy, że częstotliwości sa zapisane w ciągu $f(1), f(2), \ldots f(N+1)$, wtedy m-ty filtr jest zdefiniowany równaniem 2.3. Na rysunku 2.5 zwizualizowano 4 kolejne filtry. Efektem nałożenia filtrów jest nowy ciąg wartości $v_1, v_2, \ldots v_N$, które stanowy wejście do kolejnego etapu przetwarzania.

$$H_m(x) = \begin{cases} 0, & x < f(m-1) \\ \frac{x - f(m-1)}{\big(f(m) - f(m-1)\big)^2}, & f(m-1) \leq x \leq f(m) \\ \frac{f(m+1) - k}{\big(f(m+1) - f(m)\big)^2}, & f(m) \leq x \leq f(m+1) \\ 0, & x > f(m+1) \end{cases} \tag{2.3}$$



Rysunek 2.5: Wizualizacja 4 kolejnych filtrów trójkątnych.

Czwartym etapem przetwarzania jest nałożenie logarytmu na wektor cech otrzymany w poprzednim etapie. Ma to również związek z percepcją człowieka, gdyż ludzkie zmysły, w szczególności słuch, odbierają bodźce w skali logarytmicznej.

Piąty etap tworzenia cech *MFCC* polega na nałożeniu dyskretnej transformaty kosinusowej, zgodnie z równaniami 2.4. W efekcie jej działania otrzymamy ciąg war-

tości $W = (w_1, w_2, \ldots, w_N)$. Celem tego etapu jest dekorelacja wcześniej otrzymanych wartości, wynikająca z nakładania się na siebie ramek, filtrów z etapu trzeciego oraz samej natury sygnału akustycznego, zawierającego kolejne składowe harmoniczne dźwięków.

$$w_k = \begin{cases} \sqrt{\frac{1}{N}} \sum_{m=1}^{N} v_m, & k = 1 \\ \sqrt{\frac{2}{N}} \sum_{m=1}^{N} v_m \cos(\frac{\pi k \cdot (2m-1)}{2N}), & k = 2, 3, \ldots, N \end{cases} \tag{2.4}$$

Ostatni etap przetwarzania polega na przybliżeniu pierwszej i drugiej pochodnej, co daje informację o dynamicznej zmianie sygnału akustycznego. Pierwsza pochodna, dla t-tej ramki, może być przybliżona wzorem 2.5, natomiast druga wzorem 2.6.

$$w'_{k,t} = \frac{w_{k,t+1} - w_{k,t-1}}{2} \tag{2.5}$$

$$w''_{k,t} = \frac{2w_{k,t+2} + w_{k,t+1} - w_{k,t-1} - 2w_{k,t-2}}{10} \tag{2.6}$$

$$\epsilon = \log \sum_{i=1}^{N} v_i^2 \tag{2.7}$$

Ostatecznie, wektor cech jest składany z M pierwszych wartości ciągu W, całkowitej energii sygnału akustycznego ϵ liczonej według wzory 2.7 oraz pierwszej i drugiej pochodnej wspomnianych wartości liczonych według wzorów 2.5 oraz 2.6. Łącznie otrzymuje się wektor cech o $3(M+1)$ elementach[2]. Rysunek 2.3.1. pokazuje ostateczną formę wektora cech dla t-tej ramki.

zerowa pochodna	ϵ_t	$w_{1,t}$	$w_{2,t}$	$\ldots$	$w_{M,t}$
pierwsza pochodna	ϵ'_t	$w'_{1,t}$	$w'_{2,t}$	$\ldots$	$w'_{M,t}$
druga pochodna	ϵ''_t	$w''_{1,t}$	$w''_{2,t}$	$\ldots$	$w''_{M,t}$

Rysunek 2.6: Ostateczne forma wektora cech.

Ponadto, opcjonalnie, wektory cech mogą zostać poddane normalizacji, co uniezależnia je od niektórych czynników zewnętrznych, takich jak przykładowo głośność wypowiedzi. Normalizacji dokonuje się poprzez odjęcie średniej i podzielenie przez

[2]Typowo w przypadku cech *MFSC* $M = 12$

	MFCC	*MFSC*
liczba filtrów (N)	26	40
liczba filtrów w wektorze wynikowym (M)	12	40
logarytmowanie wartości filtrów	tak	nie
całkowita energia	tak	tak
pierwsza pochodna	tak	tak
druga pochodna	tak	tak
całkowita liczba cech	39	123

Rysunek 2.7: Zestawienie cech *MFCC* orz *MFSC*.

wariancję zgodnie ze wzorem 2.8, gdzie W jest zbiorem wektorów, a W_i i-tym wektorem ze zbioru. Każdy z parametrów wektora jest normalizowany względem odpowiadających mu parametrów w innych wektorach cech ze zbioru W. Typowo normalizacji dokonuje się dla pojedynczej frazy, co oznacza ze zbiór W zawiera jedynie wektory z jednej wypowiedzi.

$$\hat{W}_i = \frac{W_i - mean(W)}{var(W)} \tag{2.8}$$

2.3.2. Cechy *MFSC*

Cechy *MFSC* są uproszczoną wariacją cech MFCC, zaprojektowaną specjalnie na potrzeby *splotowych sieci neuronowych*. W cechach tych nie nakłada się logarytmu ani dyskretnej transformaty kosinusowej z czwartego i piątego kroku. W efekcie otrzymuje się wektory, w których kolejne elementy są pomiędzy sobą skorelowane, co jest korzystne ze względu na splotową architekturę sieci. Cechy MFSC też mogą zostać poddane normalizacji opisanej wzorem 2.8. W tabeli 2.3.2. znajduje się porównanie cech MFCC oraz MFSC wraz z parametrami, jakie zostały wykorzystane w przeprowadzonych eksperymentach.

2.4. Rozpoznawanie mowy z zastosowaniem ukrytych modeli Markowa

2.4.1. Ukryte modele Markowa - definicja

Ukryty model Markowa, w skrócie **HMM**, to automat, który przechodzi pomiędzy stanami z pewnym prawdopodobieństwem $p1$ i wraz z każdym przejściem, emituje obserwację z prawdopodobieństwem $p2$. Formalnie HMM można zdefiniować

jako krotkę 2.9.

$$HMM = (Q, O, A, B, q_0, q_F) \tag{2.9}$$

gdzie

$$\begin{aligned}
\mathbf{Q} &= \{q_1, q_2, \ldots, q_n\} && \text{Zbiór stanów automatu} \\
\mathbf{O} &= \{o_1, o_2, \ldots, o_k\} && \text{Zbiór obserwacji} \\
\mathbf{A} &= \begin{vmatrix} a_{1,1} & \ldots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \ldots & a_{n,n} \end{vmatrix} && \text{Macierz przejścia pomiędzy stanami} \\
\mathbf{B} &= \{B_1(o), \ldots, B_n(o)\}, o \in O && \text{zbiór rozkładów prawdopodobieństwa emisji} \\
& && \text{obserwacji } o \text{ w stanie } i \\
\mathbf{q_0}, \mathbf{q_F} & && \text{stany początkowy i końcowy}
\end{aligned}$$

2.4.2. Algorytmy Viterbiego

Algorytmy Viterbiego sa dynamicznymi algorytmami do znajdowania najbardziej prawdopodobnej ścieżki w automacie oraz prawdopodobieństwa stanu s_i po t krokach. Znajdują zastosowanie w algorytmie *Bauma-Welcha* służącym estymacji parametrów modelu, opisanym w rozdziale 2.4.3..

- Aby znaleźć najbardziej prawdopodobną ścieżkę kończącą się w stanie w, należy skorzystać z rekurencyjnego równania 2.10.
- Aby znaleźć prawdopodobieństwo stanu w po k krokach, należy skorzystać z rekurencyjnego równania 2.11.

$$P_t^w = \max_{q \in Q} \left(P_{t-1}^q \cdot a_{q,w} \cdot b_w(o) \right) \tag{2.10}$$

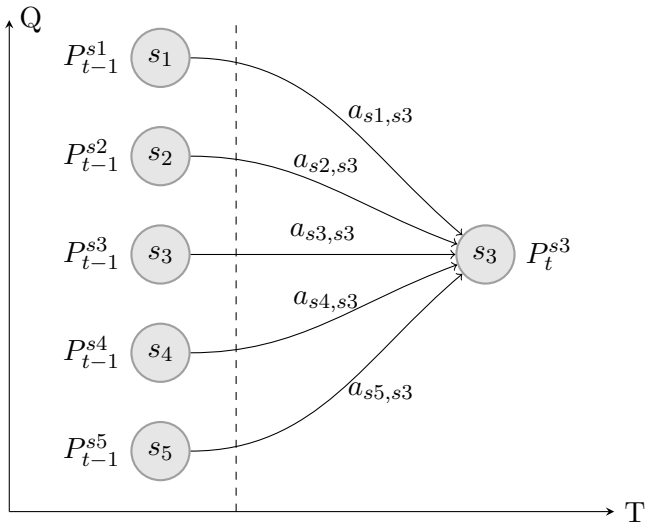
$$Q_t^w = \sum_{q \in Q} \left(P_{t-1}^q \cdot a_{q,w} \cdot b_w(o) \right) \tag{2.11}$$

Oba równania (2.10 oraz 2.11) odwołując się jedynie do wartości z poprzedniej iteracji, zatem stosując techniki programowania dynamicznego można wyznaczyć najbardziej prawdopodobną ścieżkę stosując algorytm 2.8. Algorytm ten, w zewnętrznej pętli, przechodzi przez wszystkie ramki czasowe T. Następnie dla każdego stanu $q1$, wyszukuje przejścia o największym prawdopodobieństwie, które doprowadziło ze stanu $q2$ do $q1$. Rysunek 2.10 ilustruje jedną iterację zewnętrznej pętli. Pod

Require: tablica obserwacji O o rozmiarze T
Require: macierz przejścia A o rozmiarze $|Q| \times |Q|$
1: $T_a[0 \dots |Q|] = 1$; {inicjalizacja pierwszej tablicy pomocniczej}
2: $T_b[0 \dots |Q|] = 0$; {inicjalizacja drugiej tablicy pomocniczej}
3: **for** $t = 1$ to T **do**
4: **for** $q1 = 1$ to $|Q|$ **do**
5: $p_{tmp} = 0$;
6: **for** $q2 = 1$ to $|Q|$ **do**
7: **if** $\left(T_a[q2] \cdot a_{q2,q1} \cdot b_{q1}(O_t)\right) > p_{tmp}$ **then**
8: $p_{tmp} = \left(T_a[q2] \cdot a_{q2,q1} \cdot b_{q1}(O_t)\right)$;
9: **end if**
10: **end for**
11: $T_b[q1] = p_{tmp}$
12: **end for**
13: $swap(T_a, T_b)$;
14: **end for**
15: **return** $\arg\max T_a$;

Rysunek 2.8: Algorytmy Viterbiego znajdujący najbardziej prawdopodobną ścieżkę

koniec działania algorytmu zwracany jest końcowy stan najbardziej prawdopodobnej ścieżki.
Algorytm 2.9 pokazuje analogiczny sposób znajdowania najbardziej prawdopodobnego stanu.



Rysunek 2.10: Wizualizacja jednej iteracji algorytmu Viterbiego

Require: tablica obserwacji O o rozmiarze T
Require: macierz przejścia A o rozmiarze $|Q| \times |Q|$
1: $T_a[0 \dots |Q|] = 1$; {inicjalizacja pierwszej tablicy pomocniczej}
2: $T_b[0 \dots |Q|] = 0$; {inicjalizacja drugiej tablicy pomocniczej}
3: **for** $t = 1$ to T **do**
4: **for** $q1 = 1$ to $|Q|$ **do**
5: $T_b[q1] = 0$;
6: **for** $q2 = 1$ to $|Q|$ **do**
7: $T_b[q1] += \left(T_a[q2] \cdot a_{q2,q1} \cdot b_{q1}(O_t)\right)$
8: **end for**
9: $T_b[q1] = p_{tmp}$
10: **end for**
11: $swap(T_a, T_b)$;
12: $T_b[0 \dots |Q|] = 0$;
13: **end for**
14: **return** $\arg\max T_a$;

Rysunek 2.9: Algorytmy Viterbiego znajdujący najbardziej prawdopodobną stan

2.4.3. Algorytm Bauma-Welcha

Algorytm *Bauma-Welcha* jest techniką klasy *Expectation-Maximization* służącą estymacji prawdopodobieństw przejścia pomiędzy stanami A oraz funkcji emisji b. Został on zaproponowany przez Leonarda E. Bauma, szczegółowy opis teorii dotyczącej algorytmu znajduje się w artykule [9]. Koncepcyjnie, podczas treningu wykorzystującego *algorytm Bauma-Welcha*, w $i-tej$ iteracji, rozpoznaje się zbiór treningowy z wykorzystaniem modelu M_i, otrzymując w efekcie dopasowanie stanów do obserwacji. Następnie buduje się nowy model M_{i+1}, maksymalizujący prawdopodobieństwo wcześniej dopasowanych obserwacji. Wraz z kolejnymi iteracjami parametry modelu powinny zbiegać do lokalnego maksimum.

Chcą wyznaczyć prawdopodobieństwo przejścia $q_i \to q_j$ należy policzyć:

$$a_{i,j} = \frac{\text{oczekiwana liczba przejść } q_i \to q_j}{\text{oczekiwana liczba przejść ze stanu } q_i} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \sum_{k=1}^{|Q|} \xi_t(i,k)} \tag{2.12}$$

Aby to osiągnąć zdefiniujemy następujące pomocnicze prawdopodobieństwa:

$$\xi_t(i,j) = P(q_{i,t}, q_{j,t+1} | O, M) \tag{2.13}$$

$$\beta_t(i) = P(o_{t+1}, \dots, o_T | q_{i,t}, M) \tag{2.14}$$

$$\alpha_t(i) = P(o_1, \dots, o_t, q_{i,t} | M) \tag{2.15}$$

$\alpha_t(i)$ jest dokładnie tym, co wyznacza opisany w rozdziale 2.4.2. *algorytm Viterbiego*, natomiast $\beta_t(i)$ dale się wyznaczyć za pomocą analogicznej procedury. Warto zauważyć, że:

$$P(O|M) = \sum_{i=1}^{|Q|} \alpha_t(i)\beta_t(i) \tag{2.16}$$

Dodatkowo, zdefiniujemy $\hat{\xi}$ zgodnie ze wzorem 2.17, które jest prawie tym samym co ξ. Korzystając ze wzory Bayesa w postaci 2.18, przejdziemy zgodnie z równaniem 2.19 do interesującego nas ξ.

$$\hat{\xi} = P(q_{i,t}, q_{j,t+1}, O|M) \tag{2.17}$$

$$P(X|Y,W) = \frac{P(X,Y|W)}{P(Y|W)} \tag{2.18}$$

$$\xi_t(i,j) = P(q_{i,t}, q_{j,t+1}|O,M) = \frac{P(q_{i,t}, q_{j,t+1}, O|M)}{P(O|M)} = \frac{\hat{\xi}_t(i,j)}{P(O|M)} \tag{2.19}$$

Następnie wykorzystując 2.15 oraz 2.14 zapiszemy $\hat{\xi}_t(i,j)$ w następujący sposób:

$$\hat{\xi}_t(i,j) = \alpha_t(i)a_{i,j}b_j(o_{t+1})\beta_{t+1}(j) \tag{2.20}$$

Powyższe równości pozwalają nam skutecznie wyznaczać prawdopodobieństwo przejścia $q_i \to q_j$ ze wzoru 2.12. Kolejną rzeczą potrzebną w *algorytmie Bauma-Welcha* jest prawdopodobieństwo bycia w stanie j w momencie t, opisane wzorem 2.21. Jest ono potrzebne do wyznaczenia funkcji emisji obserwacji. Na potrzeby tego rozdziału posłużymy się uproszczeniem, w którym występuje jedynie ograniczona liczba różnych obserwacji. Pozwala nam to na wyznaczenie funkcji b ze pomocą wzoru 2.22. Bardziej uogólnione podejście, w którym funkcja b nie ma ograniczeń, przedstawione jest w kolejnym rozdziale.

$$\gamma_t(j) = P(q_{t,j}|O,M) = \frac{P(q_{t,j}, O|M)}{P(O|M)} = \frac{\aleph_t(j)\beta_t(j)}{P(O|M)} \tag{2.21}$$

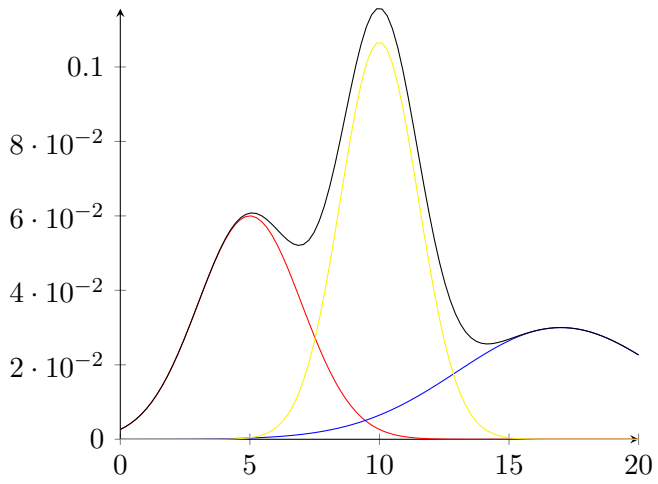
$$b_j(o) = \frac{\text{oczekiwana liczba obserwacji } o \text{ w stanie } j}{\text{oczekiwana liczba odwiedzeń stanu } j} = \frac{\sum_{t=1,o_t=o}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.22}$$

Składając razem wszystkie otrzymane równości dostajemy algorytm 2.11. W uzyskanym algorytmie linie 3, 4 odpowiadają fazie estymacji, natomiast 6, 7 odpowiadają fazie maksymalizacji. K określa liczbę epok treningu. Początkową inicjalizację można wykonać przypisując wszystkim prawdopodobieństwom równe wartości.

1: $\langle A, B \rangle \leftarrow$ zainicjalizuj
2: **for** $k = (1 \dots K)$ **do**
3: $\quad \forall_{t,j} \gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|M)}$
4: $\quad \forall_{t,i,j} \xi_t(i,j) = \frac{\alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O|M)}$
5:
6: $\quad \forall_{i,j} \xi_t(i,j) a_{i,j} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \sum_{k=1}^{|Q|} \xi_t(i,k)}$
7: $\quad \forall_{j,o} b_j(o) = \frac{\sum_{t=1, o_t=o}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$
8: **end for**
9: **return** $\langle A, B \rangle$;

Rysunek 2.11: Algorytmy Bauma-Welcha

2.4.4. Estymacja parametrów rozkładu normalnego



Rysunek 2.12: Jednowymiarowa mikstura Gaussowska składająca się z trzech komponentów.

W rozdziale 2.4.3. posłużono się uproszczeniem zakładającym, że zbiór obserwacji jest skończony i dyskretny. W rzeczywistości przy rozpoznawaniu mowy do określania $b_i(o)$ wykorzystuje się *wielowymiarowe mikstury Gaussowkie* będące sumą kilku rozkładów normalnych. Mikstura opisane jest wzorem 2.23, w którym D oznacza liczbę wymiarów wektora cech, μ jest wektorem średnich, natomiast Σ macierzą kowariancji. Rysunek 2.4.4. pokazuje przykład jednowymiarowej mikstury Gaussowskiej składającej się z trzech komponentów. Korzystając z nisko skorelowanych cech *MFCC* opisanych w rozdziale 2.3.1., można założyć, że macierz kowariancji Σ jest diagonalna. Uproszczenie to znacząco przyśpiesza obliczenia, które nie wymagają kosztownego odwracania macierzy oraz liczenie jej wyznacznika. Ponadto wzór na rozkład normalny upraszcza się do postaci 2.25 zawierającej znacznie mniej para-

metrów, które dzięki temu mogą być precyzyjniej wyznaczone.

$$b_j(o) = \sum_{m=1}^{N} c_{j,m} N_{j,m}(o) \tag{2.23}$$

$$N_{j,m}(o) = \frac{1}{(2\pi)^{\frac{D}{2}} ||\Sigma_{j,m}|^{\frac{1}{2}}} \exp\Big(-\frac{1}{2}(o-\mu_{j,m})^T \Sigma_{j,m}^{-1}(o-\mu_{j,m})\Big) \tag{2.24}$$

$$N_{j,m}(o) \simeq \frac{1}{(2\pi)^{\frac{D}{2}} (\prod_{i=1}^{D} \sigma_{j,m,i})^{\frac{1}{2}}} \exp\Big(-\frac{1}{2}\sum_{i=1}^{D}(o_i-\mu_{j,m,i})^2 \sigma_{j,m,i}\Big) \tag{2.25}$$

W celu wytrenowania parametrów mikstur Guassowkich należy zmodyfikować wzór 2.22. Teraz zamiast liczyć bezpośrednio prawdopodobieństwo obserwacji pod warunkiem stanu, wyznaczać będziemy średnią, macierze kowariancji oraz wagi komponentów tworzących miksturę Gaussowską. Do wyznaczania tych wartości posłużymy się wzorami 2.26, 2.27 oraz 2.28. Występujący w nich symbol $\delta_{t,m}(j)$ oznacza prawdopodobieństwo bycia w stanie j, w momencie t oraz emisję z komponentu Gaussowskiego m. W prostym przypadku, gdy mikstura składa się tylko z jednego komponentu, wzór na δ upraszcza się do postaci $\delta_{j,m} \equiv \gamma_j$, gdzie γ_j jest zdefiniowana wzorem 2.21. W przypadku wielu komponentów do wyznaczania δ skorzystamy z równanie 2.29, warto zauważyć, że jest to wzór bardzo podobny do końcowej definicji $\xi_t(i,j)$ wyprowadzonej w poprzednim rozdziale i zapisanej w 4 linii algorytmu 2.11. W rzeczywistości wylicza on szansę przejścia do stanu q_j wraz z emisją obserwacji przez m-ty komponent, pod warunkiem całej obserwacji O i aktualnie wyestymowanego modelu.

$$\mu_{j,m} = \frac{\sum_{t=1}^{T} \delta_{t,m}(j) o_t}{\sum_{t=1}^{T}\sum_{m=1}^{M} \delta_{t,m}(j)} \tag{2.26}$$

$$\Sigma_{j,m} = \frac{\sum_{t=1}^{T} \delta_{t,m}(j)(o_t-\mu_{j,m})(o_t-\mu_{j,m})^T}{\sum_{t=1}^{T}\sum_{m=1}^{M} \delta_{t,m}(j)} \tag{2.27}$$

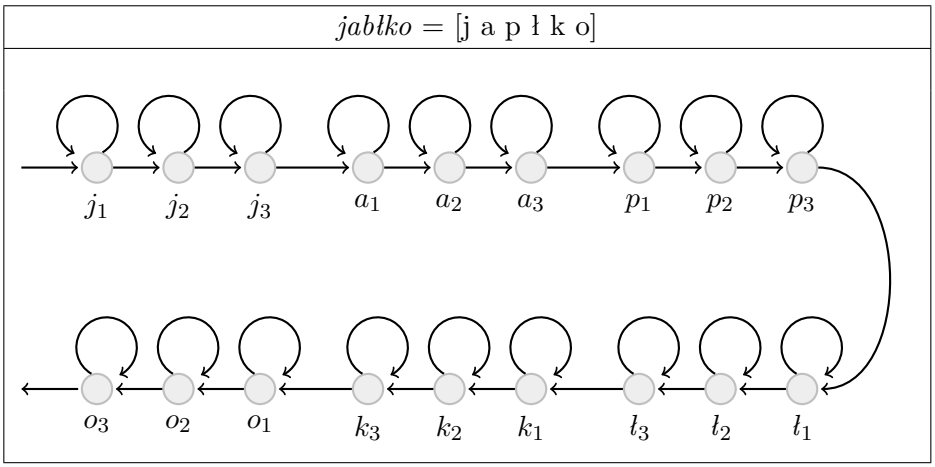
$$c_{j,m} = \frac{\sum_{t=1}^{T} \delta_{t,m}(j)}{\sum_{t=1}^{T}\sum_{m=1}^{M} \delta_{t,m}(j)} \tag{2.28}$$

$$\delta_{t,m}(j) = \frac{\sum_{i=1}^{|Q|} \alpha_{t-1}(j) a_{i,j} c_{j,m} N_{j,m}(o_t) \beta_t(j)}{P(O|M)} \tag{2.29}$$

2.4.5. Modelowanie fonetyki z wykorzystaniem HMM

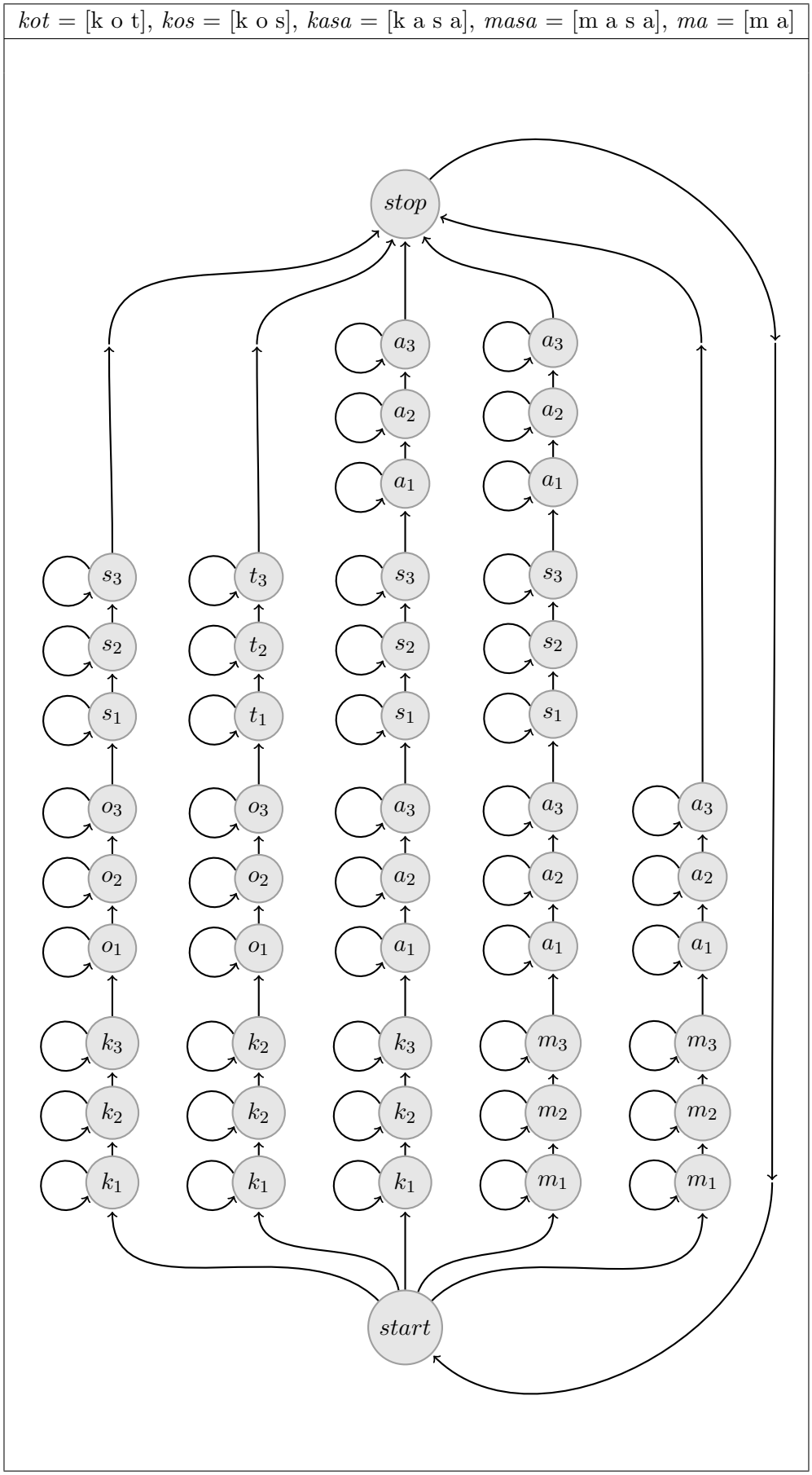
W klasycznych systemach rozpoznających mowę, wykorzystujących ukryte modele Markowa, zbiór stanów reprezentuje fonemy. Każdy fonem ma trzy stany w HMM, tak jak opisano w rozdziale 2.1.. Zbiorem obserwacji są wektory cech wyznaczane zgodnie z opisem w rozdziale 2.3.1.. Przejścia pomiędzy stanami jednego

fonemu są zgodne z opisem z rozdziału 2.1., natomiast przejścia pomiędzy różnymi fonemami są tworzone na podstawie słownika (blok (7) na rysunku 2.1). Dla każdego słowa generowany jest ciąg stanów odpowiadający kolejnym fonemom, z których składa się słowo.
Na rysunku 2.13 zamieszczono przykład automatu dla słowa *jabłko*. Zgodnie z wykorzystanymi regułami leksykalnymi słowo *jabłko* zapisuje się za pomocą fonemów jako [*japłko*].

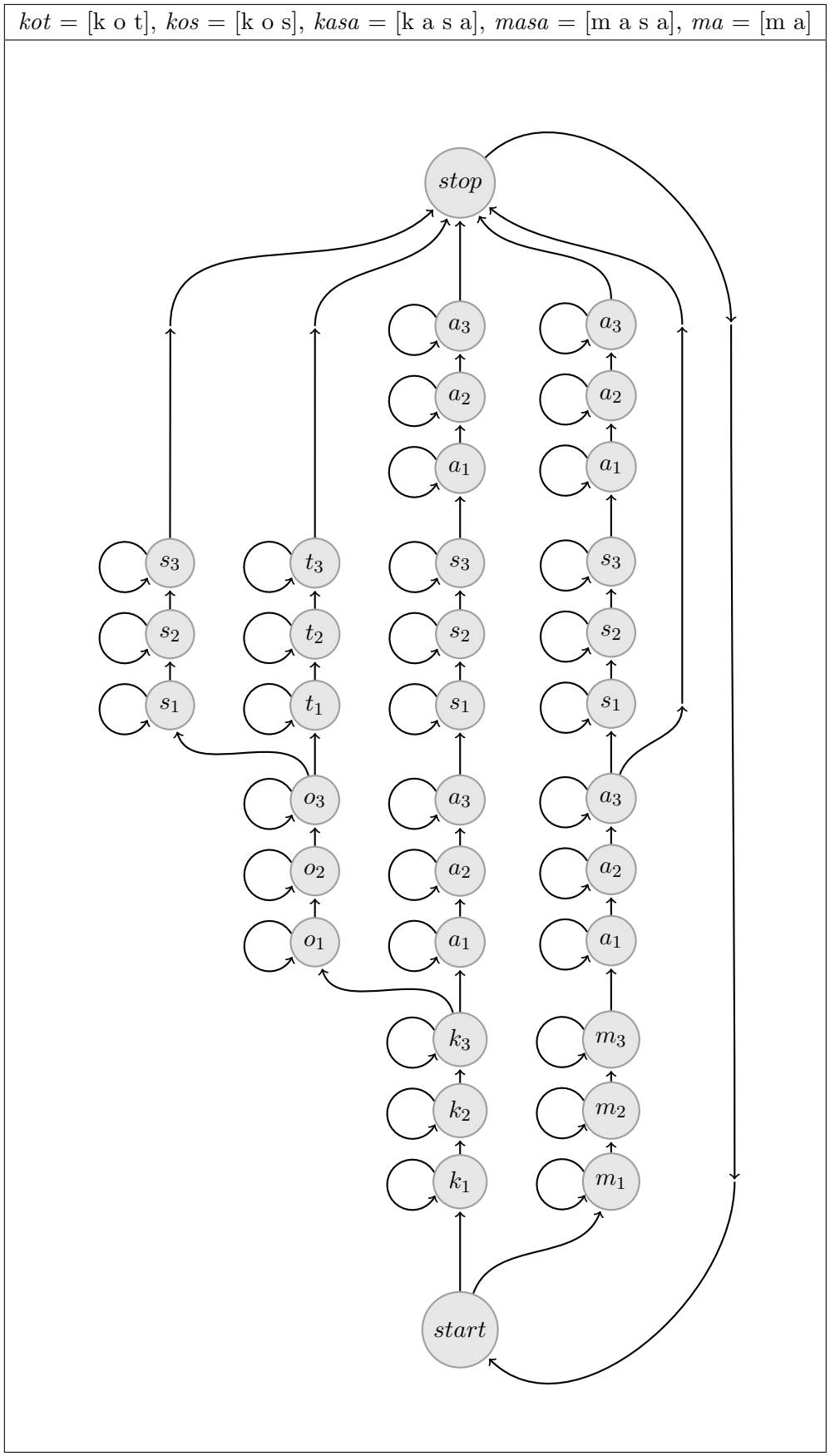


Rysunek 2.13: Automat dla słowa *jabłko*

Przy generowaniu automatu dla wielu słów, w najprostszej implementacji, dla każdego słowa ze słownika, można stworzyć osobny ciąg stanów. Rysunek 2.14 pokazuje przykład prostego grafu przejść dla słów: *kot, kos, kasa, masa, ma*. Widać na nim dwa dodatkowe stany *start* oraz *stop*, z którymi nie wiąże się emisja żadnej obserwacji. Funkcją tych stanów jest połączenie wielu łańcuchów odpowiadających słowom w jeden graf, są one równoważne stanom q_0 oraz q_F z definicji 2.9. Dodane jest również jeszcze jedno techniczne przejście, ze stanu *stop* do stanu *start*, jego zadaniem jest umożliwienie przejścia na początek automatu w celu rozpoznawania kolejnego słowa.
Opisany powyżej automat dobrze oddaje koncepcję ukrytego modelu Markowa, który jest wykorzystywany przy rozpoznawaniu mowy, jednak jest bardzo nieefektywny. Nieefektywność wynika z wielokrotnego powtarzania tych samych obliczeń, dla słów o wspólnym prefiksie. Naturalną optymalizacją jest więc połączenie wspólnych prefiksów. Rysunek 2.15 ilustruje zoptymalizowany automat 2.14. Widać, że dzięki optymalizacji otrzymujemy znacznie mniejszy graf o bardziej zwięzłej strukturze. Warto podkreślić, że nie można połączyć stanów będących sufiksami słów, gdyż dochodząc co stanu *stop* chcemy wiedzieć jakie słowo zostało rozpoznane.



Rysunek 2.14: Niezoptymalizowany automat dla słów: *kot*, *kos*, *kasa*, *masa* oraz *ma*



Rysunek 2.15: Zoptymalizowany automat dla słów: *kot*, *kos*, *kasa*, *masa* oraz *ma*

2.4.6. Rozpoznawanie

Mając zamodelowany słownik, za pomocą ukrytych modeli Markowa, zgodnie z opisem w rozdziale 2.4.5., można przejść do rozpoznawania mowy. Proces rozpoznawania jest w rzeczywistości wyszukiwaniem najbardziej prawdopodobnej ścieżki, zwanej *hipotezą*. Ze ścieżką wiąże się sekwencja wypowiedzianych słów. Ten etap rozpoznawania mowy odpowiada blokowi (4) z rysunku 2.1. W rozdziale 2.4.2. opisano algorytm 2.8, który wylicza szukaną ścieżkę. Chcąc go wykorzystać do rozpoznawania, należy zmodyfikować algorytm o uwzględnianie prawdopodobieństwa słów z modelu językowego, które wyznaczamy za pomocą n-gramów, tak jak opisano w rozdziale 2.5.. Ponadto należy dodać fragment kodu zapamiętującego słowa, przez które prowadzi ścieżka, w celu ich późniejszego wypisania.
Model językowy uwzględnia się poprzez pomnożenie prawdopodobieństwa ścieżki przez prawdopodobieństwo rozpoznanego słowa, przy przechodzeniu do stany *stop*.
W rzeczywistości wiele systemów rozpoznawania mowy, w szczególności *Julius*, wykorzystują algorytm *token passing* (2.16), który jest wariancją algorytmu Viterbiego. Czas działania algorytmu 2.10 to $O(T \cdot |Q|^2)$, co przy słowniku zawierającym kilkaset tysięcy słów przekłada się na bardzo wolny czas działania. W algorytmie *token passing* uwzględnia się jedynie stany, do których jest bezpośrednie przejście, czyli dla stanu q_i rozpatrzony będzie jedynie zbiór $\{q_j | A_{i,j} > 0\}$. Takie podejście znacząco przyśpiesza procedurę wyznaczania ścieżki.
Ponadto w algorytmie 2.16 można zastosować heurystyczną metodę przyśpieszania, polegającą na przycinaniu zbioru $Q2$ do zadanej wielkości, zwanej *szerokością wiązki*. Podczas przycinania pozostawia się jedynie n par o największym prawdopodobieństwie, gdzie n jest empirycznie wyznaczanym parametrem. W praktyce, znaczące przyśpieszenie przy jednoczesnym znikomym spadku skuteczności rozpoznawania otrzymuje się dla $n \in [500, 8000]$.

Parametry dekodowania

Wykorzystany w eksperymentach dekoder *Julius*, o którym można przeczytać w rozdziale 4.2., udostępnia szereg parametrów konfiguracyjnych wpływających na skuteczność i szybkość rozpoznawania. Przykładowo, jednym z nich jest szerokość wiązki sterująca algorytmem *token passing* opisanym w rozdziale 2.4.6.. W przeprowadzonych eksperymentach była ona ustawiona na 6500. Innymi parametrami jest *waga modelu językowego* oraz *kara za wstawienie słowa*, szczegółowy opis wszystkich dostępnym parametrów znaleźć można w [10]. Oba wspomniane parametry wpływają na ocenę hipotezy podczas procesu rozpoznawania. Załóżmy, że ich wartość to odpowiednio α dla wagi oraz β dla kary. Niech hipoteza h składa się ze słów $w_1, w_2, \cdots w_n$, którym zostały w procesie dekodowania przyporządkowane stany $s_1, s_2, \cdots, s_T$ oraz odpowiednio fragmenty obserwacji $o_1, o_2, \cdots o_T$, oczywiście $O = \overline{o_1, o_2, \cdots o_T}$, wtedy całkowita ocena h wyrażona jest wzorem 2.32. Manipulując karą za wstawianie nowych słów, można zapobiec efektowi wypisywania wielu krótkich słów, zamiast wła-

Require: tablica obserwacji O o rozmiarze T
Require: macierz przejścia A o rozmiarze $|Q| \times |Q|$
Require: zbiór krawędzi E, które występują w automacie
1: $Q1 = \{(start, 1)\}$;
2: $Q2 = \phi$;
3: **for** $t = 1$ to T **do**
4: $Q2 = \phi$
5: **for** $(q1, p1)$ in $Q1$ **do**
6: **for** $\{q2 : (q1, q2) \in E\}$ **do**
7: $p_{q_2} = \Big(p1 \cdot a_{q1,q2} \cdot b_{q2}(O_t)\Big)$;
8: $Q2 = Q2 \cup \{(q2, p_{q_2})\}$
9: **end for**
10: **end for**
11: $swap(Q1, Q2)$;
12: **end for**
13: **return** $\arg\max_{(q,p)\in Q1}(p)$;
Zbiory $Q1$, $Q2$ przechowują pary $(stan, prawdopodobieństwo)$

Rysunek 2.16: Algorytmy token passing

ściwego długiego słowa, oraz dopisywaniu krótkich słów w miejscu ciszy. Efekt ten jest związany z łatwości dopasowanie słów o małej liczbie fonemów. Wartości wagi oraz kary mogą być dobrane drogą eksperymentalną.

$$P_{acc}(h) = P(o_1|s_1) \prod_{i=2}^{T} \Big(P(o_i|s_i) \cdot a_{s_{i-1},s_i}\Big) \tag{2.30}$$

$$P_{lng}(h) = P(w_1)P(w_2|w_1) \prod_{i=3}^{n} P(w_i|w_{i-1}, w_{i-2}) \tag{2.31}$$

$$P(h) = P_{acc}(h) \cdot \Big(P_{lng}(h)\Big)^{\alpha} \cdot \beta^n \tag{2.32}$$

2.5. N-gramowy model językowy

N-gramowy model językowy, odpowiadający blokowi (5) z rysunku 2.1, zawiera prawdopodobieństwa słów ze słownika (7) . Model o stopniu n opisuje szansę wystąpienia słowa w poprzedzonego n słowami, co można zapisać jako:

$$P(w_i|w_{i-1}, w_{i-2}, \cdots, w_{i-n}).$$

Oczywiście N-gramowy model zawiera w sobie również modele o niższym stopniu, które są wykorzystywane na początku zdania. Modele językowe buduje się przez zliczanie wystąpień kombinacji słów w pewnym korpusie treningowym. Wraz ze wzrostem n znacząco rośnie ilość tekstu potrzebnego do precyzyjnej estymacji prawdopodobieństw oraz liczba kombinacji słów $w_i, w_{i-1}, \cdots, w_{i-n}$ jakie mogą wystąpić w języku naturalnym. W konsekwencji, przy ograniczonej liczbie wypowiedzi, niektóre kombinacje z warunkowego członu prawdopodobieństwa $P(w_i|w_{i-1}, w_{i-2}, \cdots, w_{i-n})$ nie występują w korpusie treningowym. Aby umożliwić rozpoznanie takiego słowa w_i, poprzedzonego słowami $w_{i-1}, w_{i-2}, \cdots, w_{i-n}$ stosuje się techniki *wygładzania* (ang: *smoothing*) takie jak przykładowo *Add-one* czy *Katz smoothing*. Ideą tych algorytmów jest przypisanie niewystępującym kombinacjom niezerowego prawdopodobieństwa. W algorytmie *Katz smoothing* rezerwuje się pewną masę prawdopodobieństwa w modelu stopnia n, które służ do przeskalowania prawdopodobieństwa modelu stopnia $n-1$ dla brakujących kombinacji słów. Procedurę przeskalowywania powtarza się rekurencyjnie, aż do napotkania pożądanej kombinacji słów. Wykorzystane w eksperymentach modele językowe miały stopień 3 i korzystały z wygładzania *Katz smoothing*.

Rozdział 3.

Neuronowy system rozpoznający mowę

3.1. Rozpoznawanie mowy z wykorzystaniem sieci neuronowych

W prezentowanym podejściu do rozpoznawania mowy z wykorzystaniem sieci neuronowych, sieć jest wykorzystywana do wyznaczania prawdopodobieństw stanów na podstawie wektora cech, co odpowiada blokowi (3) z rysunku 2.1).

3.1.1. Splotowe sieci neuronowe

Splotowe sieci neuronowe, będące rozwinięciem klasycznych sieci, zostały pierwotnie zaprojektowane na potrzeby rozpoznawania obrazów. Zyskały szerokie uznanie po prezentacji sieci, opisanej w artykule [2]. Sieć składa się zarówno ze standardowych warstw, w których wyjście jest opisane wzorem 3.1, jak i splotowych. W klasycznej warstwie wejściem jest wektor $Input$, natomiast wyjściem wektor $Output$ przemnożony przez macierz W, która powstaje w wyniku treningu sieci i przechowuje zdobytą wiedzę. Dodatkowo na każdy z osobna element wektora wyjściowego, nakłada się nieliniową funkcję f zwaną *funkcją aktywacji*. Przykładowe funkcje aktywacji to $\max(0, x)$, $\max(0.01x, x)$ lub $\tanh(x)$. Więcej funkcji aktywacji wraz ze wzorami można znaleźć w dokumentacji biblioteki wykorzystanej w eksperymentach[1].

$$Output = f(Input \cdot W) \tag{3.1}$$

Warstwa splotowa składa z zadanej parametrem n liczby filtrów, o ustalonym kształcie prostokąta o wymiarach $a \times b$. Podczas treningu wyznaczane są współczyn-

[1] http://lasagne.readthedocs.io/en/latest/modules/nonlinearities.html

niki filtrów. Wejściem warstwy jest macierzy trójwymiarowa o wymiarach $x \times y \times k$, natomiast wyjściem jest macierz trójwymiarowa o wymiarach $(x-a+1) \times (y-b+1) \times n$. Podobnie jak przy klasycznych warstwach, tutaj również na wyjście nakłada się funkcje aktywacji f. Pojedyncze wyjście $Output_{p,q,m}$ opisane jest wzorem 3.2, gdzie $Input[p : p+a, q : q+b, i]$ jest podmacierzą natomiast W_m współczynnikami m-tego filtra. Operacja splotu $*$ dla macierzy W, M o wymiarach $a \times b \times c$ zdefiniowana jest wzorem 3.3.

$$Output_{p,q,m} = f\Big(\sum_{i=1}^{k} Input[p : p+a, q : q+b, i] * W_m\Big) \tag{3.2}$$

$$W * M = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c}\Big(W[i,j,k] \cdot M[i,j,k]\Big) \tag{3.3}$$

Sieć powstaje poprzez łączenie ze sobą wielu warstw, tak aby wyjście jednej było wejściem kolejnej warstwy. Trening sieci wykonywany jest za pomocą algorytmu *back propagation* opisanego w artykule [5]. Podczas treningu pokazuje się sieci kolejno wektory wejściowe i modyfikuje współczynniki W, tak aby zminimalizować błąd wyjścia całej sieci.

3.1.2. Sieć jako estymator prawdopodobieństw stanów

Sieć neuronowa wykorzystywana jako klasyfikator, dostaje na wejściu k-elementowy wektor cech $v \in R^k$, natomiast na wyjściu zwraca n-elementowy wektor $w \in R^n_+$, gdzie n jest liczbą rozpoznawanych klas. W idealnym klasyfikatorze $w \in \{0,1\}^n$ i dokładnie jeden element ma wartość 1, a pozostałe 0. W praktyce, klasyfikatory nie udzielają tak idealnych odpowiedzi, dlatego uznajemy, że klasyfikator wybiera klasę o największej wartości w wektorze wyjściowym. Aby nadać wyjściu klasyfikatora charakter prawdopodobieństwa, należy nałożyć na niego funkcję *softmax* opisaną równaniem 3.4. Po złożeniu z funkcją 3.4 klasyfikator będzie wyznaczał $P(w_i|v)$, czyli prawdopodobieństwo i-tej klasy pod warunkiem wektora wejściowego v.

$$\sigma(w)_k = \frac{\mathrm{e}^{w_k}}{\sum_{i=1}^{n} \mathrm{e}^{w_i}}, \text{ gdzie } n \text{ to szerokości wektora } w \tag{3.4}$$

Zgodnie z równaniem 2.2 do rozpoznawania mowy potrzeba $P(O|W)$, co oznacza, że klasyfikator musi wyznaczać $P(v|w_i)$. Korzystając ze wzoru Bayesa i obserwacji, że $P(v)$ nie ma wpływu na rozpoznawanie, gdyż jest takie samo dla wszystkich hipotez, otrzymujemy:

$$P(v|w_i) = \frac{P(w_i|v) \cdot P(w_i)}{P(v)} \simeq P(w_i|v) \cdot P(w_i) \tag{3.5}$$

W proponowanym podejściu zastosowania sieci neuronowej przy rozpoznawaniu mowy, wejściem sieci jest k 123-elementowych wektorów cech, wyznaczanych zgodnie z opisem w rozdziale 2.3.2.. Parametr k określa liczbę kolejnych ramek dostarczanych sieci w wektorze wejściowym. Poprzez *szerokość kontekstu* rozumieć będziemy liczbę ramek na prawo i lewo od rozpoznawanej obserwacji, zatem $k = 2 \cdot kontekst + 1$. Zwiększanie kontekstu wiąże się ze zmniejszeniem liczby obserwacji, gdyż pierwsza obserwacja może wystąpić dopiero po liczbie ramek równej $kontekst + 1$. Rysunek 3.1 pokazuje, jak kolejne ramki są agregowane przy tworzeniu obserwacji. W przykładzie z rysunku szerokość kontekstu wynosi 2, zatem każda obserwacja jest tworzona z 5 kolejnych ramek. Wyjściem sieci jest wektor o szerokości równej liczbie stanów modelu Markowa. Uwzględniając równanie 3.5, wartość funkcji $b_q(o_t)$ z 7. linii algorytmu 2.16 odpowiada wyrażeniu:

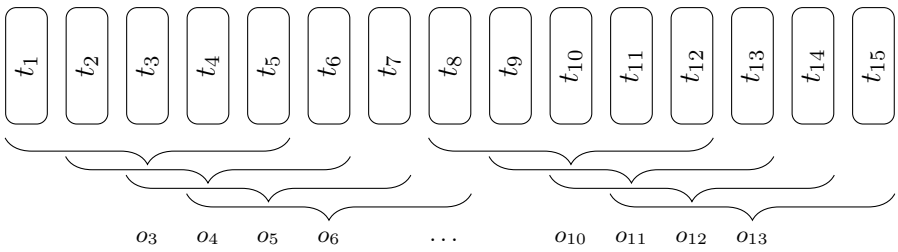
$$b_q(O_t) \simeq NN(O_t) \cdot P(w_q), \tag{3.6}$$

gdzie NN to sieć neuronowa rozumiana jako funkcja.

3.1.3. Prawdopodobieństwo stanów *a priori*

W rozdziale 3.1.2. opisano, jak wykorzystać sieć neuronową do wyznaczania prawdopodobieństw stanów, jednak wymaga to wcześniejszego wyznaczenia prawdopodobieństw stanów a priori, czyli $P(w_i)$ ze wzoru 3.5. $P(w_i)$ może być łatwo wyznaczone podczas przygotowywani danych do uczenia sieci neuronowej. Na etapie generowania wektorów wyjściowych[2] do treningu, możliwe jest zliczanie wystąpień stanu q_i. Następnie wystarczy dla każdego stanu obliczyć:

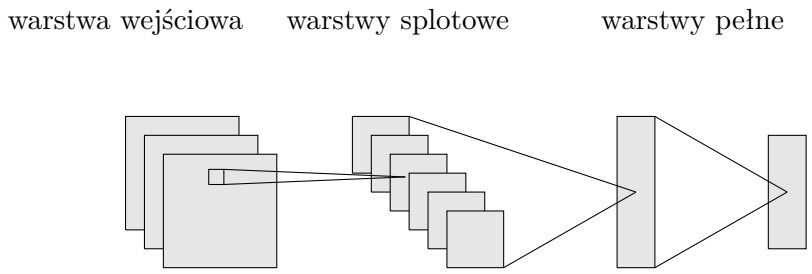
$$P(w_k) = \frac{c_k}{\sum_{i=1}^{|Q|} c_i}, \text{ gdzie } c_i \text{ to liczba wystąpień stanu } q_i \tag{3.7}$$



Rysunek 3.1: Tworzenie obserwacji dla sieci neuronowej przy kontekście równym 2.

[2]Wektory wyjściowe kodują numer stanu, jaki ma zostać rozpoznany, stosują kodowanie *1-hot*.

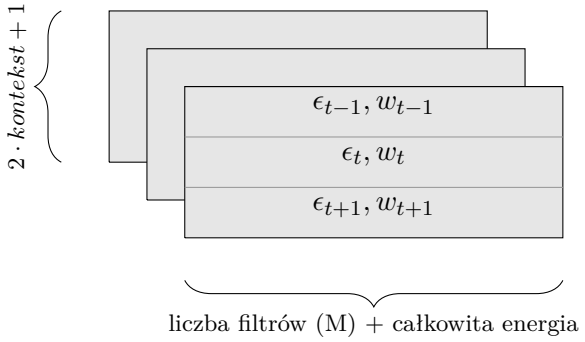
3.1.4. Architektura sieci i układ danych



Rysunek 3.2: Architektura splotowej sieci neuronowej.

Do wyznaczanie prawdopodobieństw stanów modelu Markowa wykorzystano splotową sieć neuronową. Sieć składa się z odpowiednio sformatowanej warstwy wejściowej, warstw splotowych oraz klasycznych, pełnych warstw na samym końcu. Rysunek 3.1.4. wizualizuje ogólną architekturę wykorzystanych sieci. Szczegółowy opis konfiguracji warstw znajduje się w rozdziale 4.. Jako funkcję błędu zastosowano *entropię skrośną*. W celu zapobieżenia efektowi przeuczania dodano *dropout* ze współczynnikiem odrzucania równym 0.5 do pierwszej warstwy klasycznej oraz *regularyzacje* parametrów ze współczynnikiem 10^{-5}. *Dropout* jest techniką zapobiegającą przeuczaniu sieci polegającą na zerowaniu wejścia neuronu z zadanym jako parametr prawdopodobieństwem. Początkowy współczynnik uczenia wynosił $5 \cdot 10^{-3}$ i w każdej epoce, w której nie nastąpiła poprawo na zbiorze walidacyjnym, był mnożony przez 0.94, trening trwał 160 epok.

Wejście sieci jest podzielone na trzy warstwy odpowiadające zerowej (ϵ, w), pierwszej (ϵ', w') i drugiej pochodnej $\epsilon'', w'')$, co odzwierciedla charakter wektorów cech wyznaczanych zgodnie z opisem w rozdziale 2.3.2.. Dzięki oddzieleniu pochodnych warstwami, dane pochodzące z różnych pochodnych nie są przetwarzane przez jeden filtr. Szerokości jednej warstwy wejściowej odpowiada liczbie uwzględnionych, przy ekstrakcji cech, filtrów MEL-owych wraz z całkowitą energią, łącznie $M + 1$ liczb. Wysokości jest równa liczbie ramek z jakich składna się pojedyncza obserwacja, czyli $2 \cdot kontekst + 1$. Na rysunku 3.1.4. zwizualizowano podział wektora wejściowego.



Rysunek 3.3: Wejście splotowej sieci neuronowej.

3.2. Zastosowanie modelowania fonemów z ograniczonym kontekstem w ASR

Wykorzystując ukryte modele Markowa, tak jak opisano w rozdziale 2.4., wymaga się policzenia dla każdej ramki prawdopodobieństwa obserwacji pod warunkiem każdego ze zdefiniowanych stanów. Wykorzystując trifony przy rozpoznawaniu mowy, trzeba zdefiniować $O(\sigma^3)$ stanów. Jednak tylko niektóre z trifonów o wspólnym środkowym fonemie różnią się w istotny sposób. Oczywiście wraz ze wzrostem liczby fonemów, rośnie liczba obliczeń, które trzeba wykonać, co ma negatywny wpływ na szybkość rozpoznawania. Ponadto, przy wykorzystaniu sieci neuronowych zgodnie z podejściem opisanym w rozdziale 3.1. trzeba wytrenować więcej parametrów, co z kolei utrudnia lub wręcz uniemożliwia proces uczenia. Aby ograniczyć negatywny wpływ dużej liczby trifonów, zastosowano technikę opisaną w rozdziale 3.2.1.. Dzięki niej możliwe jest znaczne ograniczenie liczby stanów, a w konsekwencji szerokości warstwy wyjściowej w sieci neuronowej.

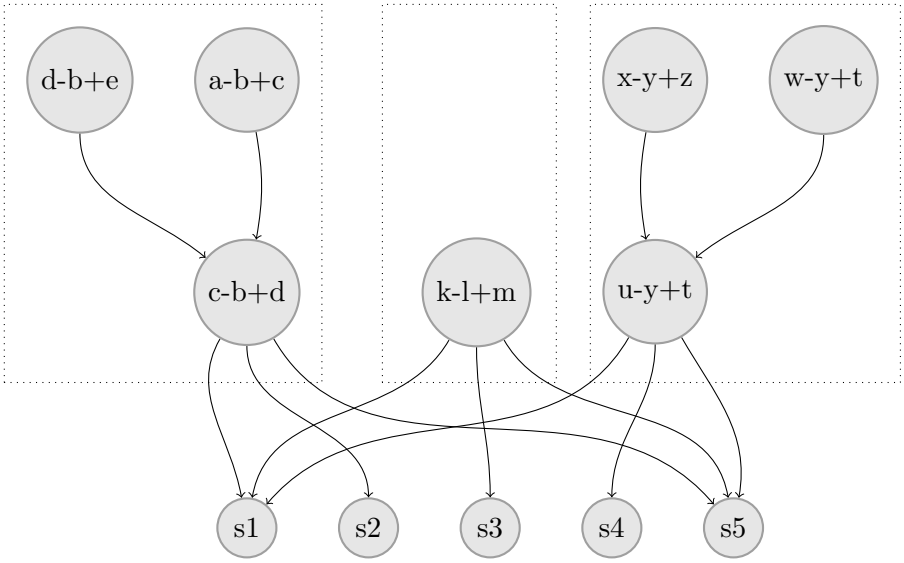
3.2.1. Definicja modelu z ograniczonym kontekstem

Korzystając ze spostrzeżenia, że tylko niektóre trifony o wspólnym środkowym fonemie istotnie się różnią, można zastosować mechanizm współdzielenia modeli przez wiele trifonów, tworząc tym samym *klasy abstrakcji*. Grupa trifonów należeć będzie do tej samej klasy abstrakcji, jeśli opisana jest takimi samymi parametrami, w skład których wchodzą stany tworzące fonem oraz prawdopodobieństwa przejść pomiędzy nimi. Ponadto trifony w obrębie jednej klasy abstrakcji będą mieć taki sam środkowy fonem. Z przyczyn implementacyjno-technicznych każda klasa abstrakcji będzie miała swojego reprezentanta, zwanego *trifonem fizycznym*. Rysunek 3.4 pokazuje przykładowe klasy abstrakcji. Oznaczenia trifonów z rysunku są zgodne z konwencją przedstawioną w rozdziale 2.2.. Schemat przedstawia trzy klasy abstrakcji

- $(d-b+e) \equiv (a-b+c) \equiv (c-b+d)$
- $(k-l+m)$
- $(x-y+z) \equiv (w-y+t) \equiv (u-y+t)$

Każda z nich jest zaznaczona ramką. Reprezentantem jest trifon z którego wychodzą bezpośrednie przejścia do stanów. Stany z których zbudowani są reprezentanci klas abstrakcji mogą być współdzielone. Oba mechanizmy, współdzielenie pojedynczych stanów oraz całych konfiguracji, umożliwia bardzo skuteczne ograniczenie liczby parametrów, które trzeba wyliczyć podczas procesu rozpoznawania i treningu modelu akustycznego. W tabeli 4. umieszczono zestawienie modeli akustycznych, wytrenowanych na potrzeby eksperymentu, w kolumnie *Liczba stanów* wpisano liczbę stanów, do jakiej ograniczono modele trifonowe, korzystając z opisanego mechanizmu.

Poprzez *Model z ograniczonym kontekstem* będziemy rozumieć akustyczny model trifonowy, w którym ograniczono liczbę stanów modelu Markowa poprzez oba opisane powyżej sposoby.



Rysunek 3.4: Współdzielenie modeli przez klasy abstrakcji.

3.2.2. Metody ograniczania kontekstu

Wykorzystany do budowy modeli akustycznych pakiet *HTK* oferuje dwie metody ograniczania liczby stanów. Pierwsza metoda o nazwie *Data driven clustering* zakłada, że na wejściu dostaje się zbiór trifonów. Każdy z trifonów ma przypisane trzy stany, zgodnie z opisem w rozdziale 2.1.. Następnie porównuje się zróżnicowanie w obrębie pierwszych, drugich i trzecich stanów. Jeśli dwa stany są podobne, łączy się je. W efekcie, wraz z każdą iteracją algorytmu, otrzymuje się mniejszy zbiór stanów wykorzystywanych przez zbiór trifonów. Jako miarę podobieństwa pomiędzy stanami wykorzystuje się odległość Euklidesową średnich z rozkładów normalnych modelujących stan. Wadą tej metody jest niemożność obsłużenia trifonów, które nie wystąpiły w zbiorze treningowym.

Drugą metodą z pakietu *HTK*, która została wykorzystana podczas eksperymentów, jest *Tree- based clustering*. Twórcy *Tree- based clustering* opisali swój algorytm w artykule [7]. W przeciwieństwie do *Data driven clustering*, ta metoda obsługuje również trifony, które nie pojawiły się w zbiorze treningowym. Metoda ta wymaga wczytania zbioru *pytań*, opisujących zależności pomiędzy fonemami. *Pytania* powinny odzwierciedlać fizyczne podobieństwa w wymowie fonemów. Zapisuje się je poprzez zbiory, odpowiedz na *pytanie* jest twierdząca jeśli fonem należy do zbiory, negatywna w przeciwnym wypadku. Dzięki temu każde z pytań dzieli cały zbiór zdefiniowanych fonemów na dwa podzbiory. Przykładowymi pytaniami są:

- Czy fonem x jest nosowy?
- Czy fonem x jest spółgłoską zwartą?
- Czy fonem x jest dźwięczny?
- Czy fonem x jest spółgłoska miękką?
- Czy fonem x jest spółgłoską twardą?
- Czy fonem x jest głoską ustną?
- Czy fonem x jest spółgłoską sonorną?

Zbiór wykorzystanych pytań został zbudowany na podstawie zaleceń z *HTK Book*[14] oraz ogólnodostępnych zasad polskiej fonetyki. Po wczytaniu zbioru trifonów współdzielących lewy lub prawy stan, wszystkie zdefiniowane pytania są uruchamiane. W efekcie dostaje się zbiór hipotetycznych podziałów, z których wybierany jest jeden. Aby zdecydować, który podział jest najlepszy, dla każdego z nich, liczy się prawdopodobieństwo zbioru treningowego. Pytanie maksymalizujące prawdopodobieństwo jest wybierana.
Algorytm *Tree-based clustering* wykorzystywany w systemie *HTK* został zapisany w pseudokodzie na rysunku 3.5. Kod uwzględnia tylko podziały lewego stanu spośród trójki definiującej trifon. Linie od 2 do 20 opisują jedną iterację, w wyniku której wybierana jest najkorzystniejsza reguła. Wraz z każdą iteracją, zwiększa się liczba stanów w zbiorze Q, a jednocześnie zmniejsza się liczba trifonów współdzielących ten sam stan. Najbardziej zewnętrzna pętla wykonywana jest dopóki istnieje jakiś podział spełniający wymagane kryteria. Pętla w linii 3 przechodzi po wszystkich stanach, które są aktualnie zdefiniowane w modelu akustycznym. Następnie wyliczany jest zbiór P zawierający trifony współdzielące pierwszy stan. W kolejnej pętli z linii 5 przechodzi się przez zdefiniowane reguły podziału. Przy pomocy wybranej reguły r zbiór P dzieli się na dwa podzbiory P_l, P_r. W liniach 9 oraz 10 wybierane są ramki, które podczas rozpoznawania zostały przyporządkowane do stanu q_l i q_r. Warunek w linii 11 sprawdza, czy oba zbiory są odpowiednio liczne. Następnie korzystając z wyznaczonych zbiorów trenuje się parametry rozkładów opisujących dwa nowe stany q_l, q_r. Ostatecznie wylicza się w linii 16 prawdopodobieństwo zbiorów f_l, f_r, które stanowią ocenę reguły podziału. Wyliczona ocena jest dodawana do zbioru X. Po przejściu przez wszystkie stany z pętli w linii 3 wybiera się regułę o najlepszej ocenie. Reguła ta jest następnie wykorzystania do rozbudowy modelu akustycznego o nowy stan w liniach 23 oraz 24. Na końcu działania, w linii 26, algorytm zwraca rozbudowany zbiór stanów Q oraz definicji trifonów *Phones*.
Na rysunku 3.7 znajduje się przykład działania opisanego algorytmu. Ilustruje on, jak podziały wpływają na współdzielenie stanów. Początkowo wszystkie trzy trifony dzielą te same stany. Odpowiada to sytuacji, gdy mamy tylko jedną klasę abstrakcji. W pierwszej iteracji wybrana została reguła dla pierwszego stanu, dzieląca fonemy

na dwie grupy $\{a, w\}, \{n\}$. W efekcie stan $s1$ został podzielony na dwa $s0, s1$. W kolejnej iteracji wybrano regułę dla stanu trzeciego dzielącą fonemy na grupy $\{z\}, \{b\}$. Stan $s3$ został podzielony na dwa nowe $s3, s4$. W wyniku działania algorytmu powstały trzy klasy abstrakcji, każda z nich ma inny zbiór stanów, które ją modelują. Proces podziału w algorytmie *tree-based clustering* jest iterowany tak długo, dopóki nie osiągnie się kryterium stopu. Jednym z możliwych kryteriów jest *minimalna liczba obserwacji*. W pseudokodzie na rysunku 3.5 warunek z linii 11 jest związany właśnie tym kryterium. Stosując ten warunek wymaga się, aby przy podziale na dwa nowe stany, oba z nich miały co najmniej n obserwacji, gdzie n jest zadanym parametrem. Drugim kryterium jest *poprawa prawdopodobieństwa obserwacji*, zapisana w linii 14. Wymaga się przy nim, aby po podziale prawdopodobieństwo obserwacji przypisanych do dzielonego stanu zwiększyło się o podany współczynnik λ. Wymóg ten jest opisany wzorem 3.8, ze względów numerycznych stosuje się tu skalę logarytmiczną. Oznaczenia we wzorze są takie same, jak w pseudokodzie. Prawdopodobieństwo $P(o|q)$ jest wyliczane zgodnie z opisem w rozdziale 2.4.4..

$$\log P(f_l|q_l) + \log P(f_r|q_r) - \log P(f_l|q) - \log P(f_r|q) > \lambda \tag{3.8}$$

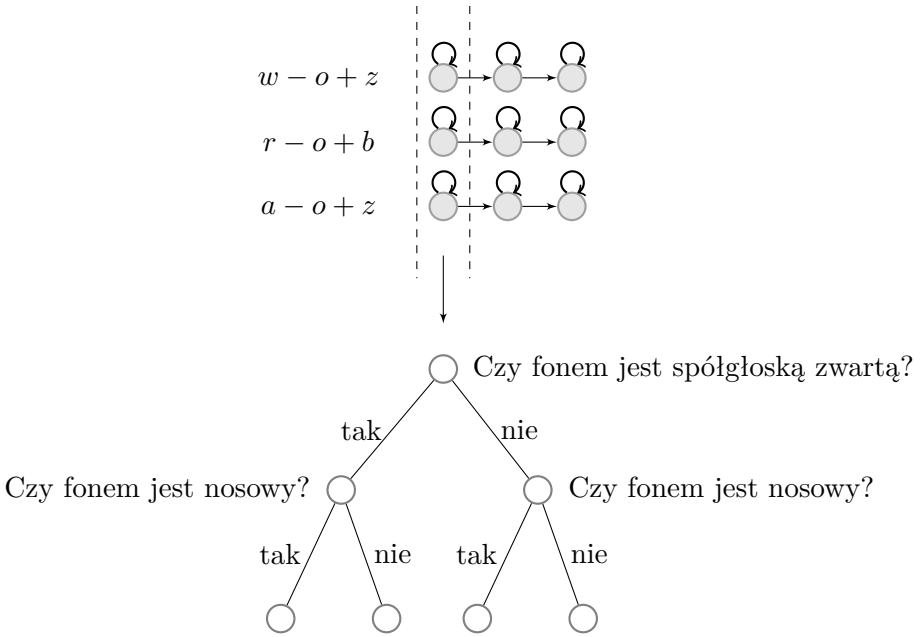
Ustawiając wysoką liczbę wymaganych obserwacji oraz duży współczynnik poprawy można zdławić proces dzielenia się trifonów i otrzymać model o małej liczbie stanów. Analogicznie przesuwając parametry w druga stronę można zbudować bardzo duży model. Warto zauważyć, że w przypadku tego algorytmu, nie ma sensu poddawać podziałowi środkowego stanu.
Rysunek 3.6 pokazuje zagłębianie się reguł nałożonych na pierwszy stan wraz z kolejnymi iteracjami. Początkowo wybierana jest reguła pytająca o spółgłoski zwarte, która dzieli trifony na dwie grupy. Następnie obie grupy dzielone są regułą odpytująca o fonemy nosowe.
Dodatkowo po wykonaniu podziałów stanów algorytmem *Tree-based clustering*, można zastosować algorytm podobny do opisanego *Data driven clustering* w celu połączenia podobnych stanów pochodzących z trifonów o różnym fonemie centralnym. Proces ten nazywa się *Parameter Tying*.

Require: Zbiór reguł $Rules$
Require: Zbiór stanów Q
Require: Zbiór trifonów $Phones$
Require: Zbiór ramek F

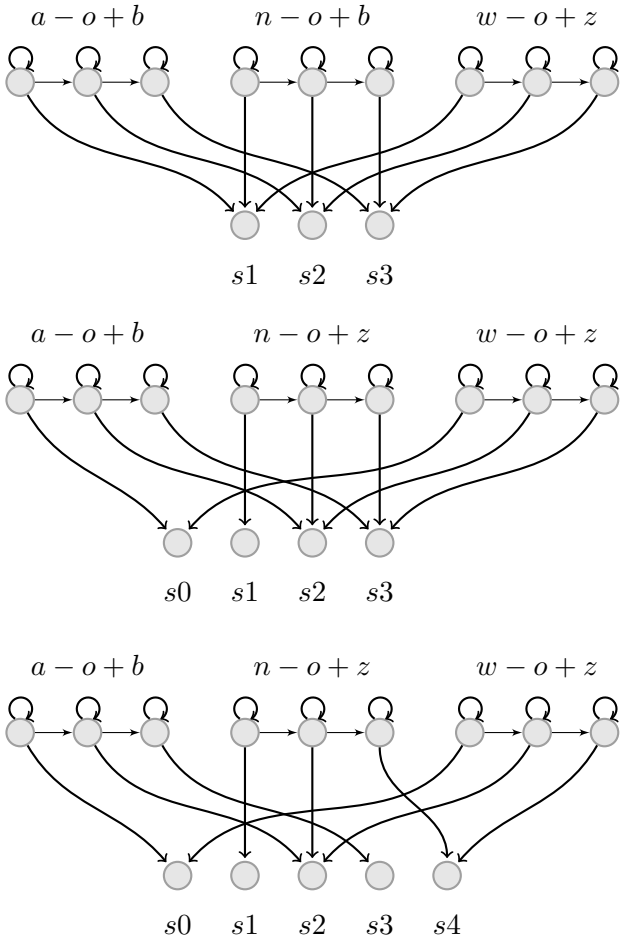
while $TRUE$ **do**
  $X = \emptyset$
  for $q \in Q$ **do**
    $P = \left\{ \langle q_1, q_2, q_3 \rangle \in Phones : q_1 = q, q_2 \in Q, q_3 \in Q \right\}$
    for $r \in Rules$ **do**
      $\langle P_l, P_r \rangle = r(P)$
      $q_l = P_{l,1}$
      $q_r = P_{r,1}$
      $f_l = F(q_l)$
      $f_r = F(q_r)$
      if $|f_l| > n$**and**$|f_r| > n$ **then**
        $q_l.params = estimate(f_l)$
        $q_r.params = estimate(f_r)$
        if $\log P(f_l|q_l) + \log P(f_r|q_r) - \log P(f_l|q) - \log P(f_r|q) > \lambda$ **then**
          $prob = P(f_l|q_l) \cdot P(f_r|q_r)$
          $X = X \cup \{\langle prob, r, q, r_l, q_r \rangle\}$
        end if
      end if
    end for
  end for
  if $X \neq \emptyset$ **then**
    $\langle prob, r, q, q_l, q_r \rangle = \arg\max_{\langle prob,r,q,r_l,q_r \rangle \in X}(prob)$;
    $Q = Q \setminus q \cup \{q_l, q_r\}$
    $Phones = rebuild(Phones, Q)$
  else
    return $\langle Q, Phones \rangle$
  end if
end while

Zbiory X przechowują pary $(prawdopodobieństwo, reguła, dzielonystan)$

Rysunek 3.5: Algorytmy tree-based clustering zastosowany w pakiecie HTK



Rysunek 3.6: Tree-based clustering - nakładanie pytań



Rysunek 3.7: Tree-based clustering - dzielenie stanów

Rozdział 4.

Eksperymenty

Wiele badań i publikacji pokazało, że stosując *trifonowe modele akustyczne* można znacząco poprawić skuteczność rozpoznawania. Niestety przejście z modeli *unifonowych* do *trifonowych* wiąże się ze zwiększeniem stopnia skomplikowania całego systemu, procesu uczenia, jak i wydłużeniem samego czasu rozpoznawania. Ponadto, w przypadku zastosowania sieci neuronowej, duża liczba stanów modelu trifonowego wymaga jednoczesnej estymacji wielu prawdopodobieństw, co z kolei może utrudnić proces trenowania sieć. Kompromisem pomiędzy modelami czysto trifonowymi, które modelują wszystkie występujące trifony, a unifonowymi oferującymi przeciętną jakość rozpoznawania, są właśnie *modele trifonowe z ograniczonym kontekstem* budowane zgodnie z koncepcją opisane w rozdziale 3.2.2.. Modele te łączą zwiększoną zdolność gromadzenia wiedzy klasycznych modeli trifonowych ze zmniejszoną liczbą parametrów, które trzeba wyuczyć w klasycznym modelu unifonowym. Ponadto w przeciągu kilku ostatnich lat, pojawiło się wiele publikacji na temat zastosowania sieci neuronowych, jako estymatorów prawdopodobieństwa stanów model Markowa, tak jak w artykule [8]. Chcąc sprawdzić możliwości połączenia obu wspomnianych podejść, *modeli z ograniczonym kontekstem* oraz *sieci*

Nazwa modelu	Min liczba obserwacji	Próg poprawy	Liczba stanów	Skuteczność
M_TRI_OPT	100	350	4192	83.44
M_TRI	100	350	4192	81.58
M_25000	25000	350	304	76.90
M_29000	29000	350	269	77.75
M_33000	33000	350	231	77.23
M_37000	37000	350	214	77.13
M_70000	70000	350	147	75.91
M_180000	18000	350	122	75.17
M_UNI	N/A	N/A	120	76.86

Rysunek 4.1: Wytrenowane modele akustyczne

neuronowych estymujących prawdopodobieństwa stanów, zgodnie z opisem w rozdziale 3.1., przeprowadzono szereg eksperymentów, mających sprawdzić wpływ ograniczania kontekstu na skuteczność sieci neuronowej. Dodatkowo porównano skuteczność klasycznego podejścia opisanego w rozdziale 2.4. z proponowanym podejściem. Przeprowadzone eksperymenty dały możliwość zweryfikowania głównej tezy niniejszej pracy, zgodnie z którą: **Trifonowe, neuronowe modele o ograniczonym kontekście umożliwiają poprawę skuteczności rozpoznawania, względem klasycznych Gaussowkich modeli akustycznych.**

4.1. Opis danych

W celu przetestowania proponowanych metod wykorzystano studyjny korpus Clarin[1]. Składa się on z 56 godzin polskich nagrań o różnej tematyce, nagranych przez 554 różnych mówców. Każdy mówca nagrał 20 lub 30 wypowiedzi o długości od 6 do 22 sekund. Korpus nie jest zbalansowany pod względem płci. Wypowiedzi mają przypisane transkrypcje, jednak znajduje się w nich wiele błędów i artefaktów, takich jak zająknięcia (4. wypowiedz 191. mówcy), seplenienie (200. mówca) czy błędne odczytanie tekstu (18. wypowiedz 294. mówcy) skutkujące różnicą pomiędzy transkrypcją a faktycznie wypowiedzianymi słowami. Artefakty utrudniają lub wręcz uniemożliwiają wykorzystanie niektórych nagrań do uczenia modelu akustycznego[2]. Dźwięk został nagrany z próbkowaniem 16 kHz i jakością 16 bitów na próbkę, a następnie zapisany w niekompresowanym formacie wav. Wypowiedzi zaczynają się i kończą ciszą.
W fazie wstępnego przetwarzania danych, na potrzeby przeprowadzonych eksperymentów, nagrania poddano następującym procesom:

- automatyczne wyrównanie poziomu nagrań to stałego poziomu -6dB (1/2 maksymalnego możliwego do osiągnięcia poziomu) dla najgłośniejszych miejsc całego nagrania
- odrzucenie fragmentów ciszy/szumów tła o długości przekraczającej 0.7 sek,
- odrzucenie tych nagrań, dla których faktycznie wypowiedziana fraza różniła się od frazy zadeklarowanej w dostarczonej transkrypcji.

W celu wykonania ostatniego kroku, służącego automatycznemu oczyszczeniu nagrań, zbudowano klasyczny model akustyczny i językowy na wszystkich nagraniach z korpusu Clarin. Tak otrzymane modele dają bardzo wysoką skuteczność rozpoznawania zbioru treningowego, na podstawie którego je zbudowano. Następnie wykorzystując zbudowany model, rozpoznano nagrania z korpusu. Wypowiedzi o idealnym

[1] http://mowa.clarin-pl.eu/korpusy/
[2] Niektóre nagrania zawierają powtórzenia, które nie są zadeklarowane w transkrypcji.

	Liczba wypowiedzi	Udział wypowiedzi [%]	Czas wypowiedzi [s]	Udział czasu [%]
przed czyszczeniem	11998	100	144895	100
po czyszczeniu	9193	76.6	101957	70.4

Rysunek 4.2: Wpływ procesu automatycznego czyszczenia na rozmiar zbioru treningowego.

rozpoznaniu uznano za poprawne, natomiast te o niepoprawnym odrzucono. Odrzucone nagrania nie były wykorzystywane do budowy żadnych późniejszych modeli. Ostatecznie, oczyszczony korpus został podzielony na trzy zbiory: treningowy, walidacyjny oraz testowy w stosunku 27:2:1. Zbiory są rozłączne pod względem mówców. Tabele na rysunku 4.1. pokazuje jak proces automatycznego czyszczenia wpłynął na rozmiar zbiory treningowego. W przypadku większości z odrzuconych wypowiedzi deklarowana transkrypcja nie pokrywa się z rzeczywiście wypowiedzianą frazą.

4.2. Środowisko sprzętowo-programisytyczne

W celu przeprowadzenia eksperymentów wykorzystany został szereg ogólnodostępnych pakietów i bibliotek oraz komercyjny system do rozpoznawania mowy *Magic Scribe* firmy *Radcomp Integral* (`http://radcomp.eu`) z Wrocławia. System *Magic Scribe* wykorzystano do zarządzania korpusem *Clarin* oraz do przeprowadzenia testów skuteczności. W skład ogólnodostępnych, wykorzystanych pakietów wchodzi *HTK* (`http://htk.eng.cam.ac.uk/`) oferujący zestaw narzędzi do budowy klasycznych modeli akustycznych oraz językowych. Kolejnym wykorzystanym narzędziem jest *Julius* (`http://julius.osdn.jp/en_index.php`) będący dekoderem w pełni kompatybilnym z pakietem HTK. Julius jest jednym z kilku open-sourcowych dekoderów. Pomimo, że nie oferuje on najwyższej skuteczności rozpoznawania, o czym można przeczytać w artykule [3], zdecydowałem się na to narzędzie, ponieważ mam doświadczenie w pracy z nim. Ponadto Julius uchodzi za szybki dekoder, działający w czasie zbliżonym do rzeczywistego, co też jest istotne z punktu widzenia użytkownika systemu rozpoznającego mowę. Stanowi to dużą zaletę względem dekodera z pakietu *HTK*, który jest relatywnie wolny i ze względu na dwubajtowe adresowanie *słownika* nie jest w stanie wczytać wszystkich słów z modelu językowego. W artykule [12] twórcy *Juliusa* opisują swój system oraz zastosowane algorytmy i struktury danych umożliwiające osiągnięcie wysokiej szybkości rozpoznawania. Do budowy sieci neuronowych wykorzystano bibliotekę *Lasagne* `http://lasagne.readthedocs.io/en/latest/`, napisaną w języku Python. Zapewnia ona wygodny interfejs do środowiska *Theano*. Lasagne oferuje zestaw metod do budowy sieci składających się z warstw klasycznych, splotowych, dropout, max-pull itd. *Theano* (`http://deeplearning.net/software/theano/`) jest z kolei

biblioteką umożliwiającą definiowanie oraz optymalizację równań matematycznych, które mogą być następnie skompilowane i wykonane na CPU lub wielowątkowo na karcie graficznej. Z wykorzystaniem tej biblioteki oraz GPU udało się znacząco przyśpieszyć trening sieci neuronowych w stosunku do treningu na CPU. Schemat zależności pomiędzy użytymi komponentami oraz budowanymi w trakcie eksperymentów modelami, znajduje się na rysunku 4.3. Poniżej zamieszczono opis kolejnych kroków wykonanych przy eksperymentach.

- Punktem wyjścia stworzonej architektury jest korpus językowy *Clarin* przedstawiony w rozdziale 4.1., na schemacie 4.3 odpowiada mu blok (8) .
- Na podstawie transkrypcji zawartych w korpusie, zbudowano model akustyczny oraz językowy wraz ze słownikiem (10) , wykorzystując do tego pakiet HTK oraz system Magic Scribe z bloku (9) . Do budowy obu modeli oraz słownika (10) wykorzystano wszystkie wypowiedzi ze zbioru treningowego, walidacyjnego i testowego.
- Powstałe w poprzednim kroku modele zostały wykorzystane do rozpoznania wypowiedzi z korpusu (8) i stworzenia plików *mfsc*[3] oraz *out*[4] tworzących korpus (15) . Każdej z oryginalnych wypowiedzi znajdujących się w korpusie (8) przypisano w zbiorze (15) , plik *mfc* zawierający kolejne wektory cech oraz plik *out* z zapisanym mapowaniem ramek na trifony wraz z numerami stanu. Pliki *out* wygenerował dekoder (14) podczas procesu rozpoznawania. Uzyskano bardzo wysoką skuteczność rozpoznawania wynoszącą 99.3%, dzięki czemu dopasowanie ramek do stanów w korpusie (15) jest bardzo precyzyjne. Wysoka skuteczność jest efektem budowy modelu akustycznego oraz językowego (10) na zbiorze, który został nimi później rozpoznany. Słownik z modelu (10) zawiera jedynie słowa, które mają się pojawić w wyniku rozpoznawania.
- Na podstawie korpusu (15) przygotowano za pomocą narzędzia (16) dane do treningu sieci neuronowej. Efektem działania narzędzia (16) jest plik z mapowaniem *wektor cech* $\longrightarrow$ *stan modelu Markowa*. Podczas tego etapu dokonana została normalizacja wektorów cech dla każdej frazy zgodnie z opisem w rozdziale 2.3.1..
- Wykorzystując stworzone mapowanie oraz wspomnianą powyżej bibliotekę *Lasagne* i *Theano* w kroku (17) wytrenowano szereg sieci neuronowych (18) .
- Uruchomiono rozpoznawanie *zbioru testowego* dekoderem (19) korzystającym z sieci neuronowych (18) . W efekcie otrzymano skuteczności rozpoznawania (20) . Należy dodać, że dekoder (19) jest oryginalnym dekoderem *Julius* rozbudowanym o własnoręcznie dopisany moduł pobierający prawdopodobieństwa stanów modelu Markowa z sieci neuronowej, zamiast z klasycznego modelu Gaussowskiego.

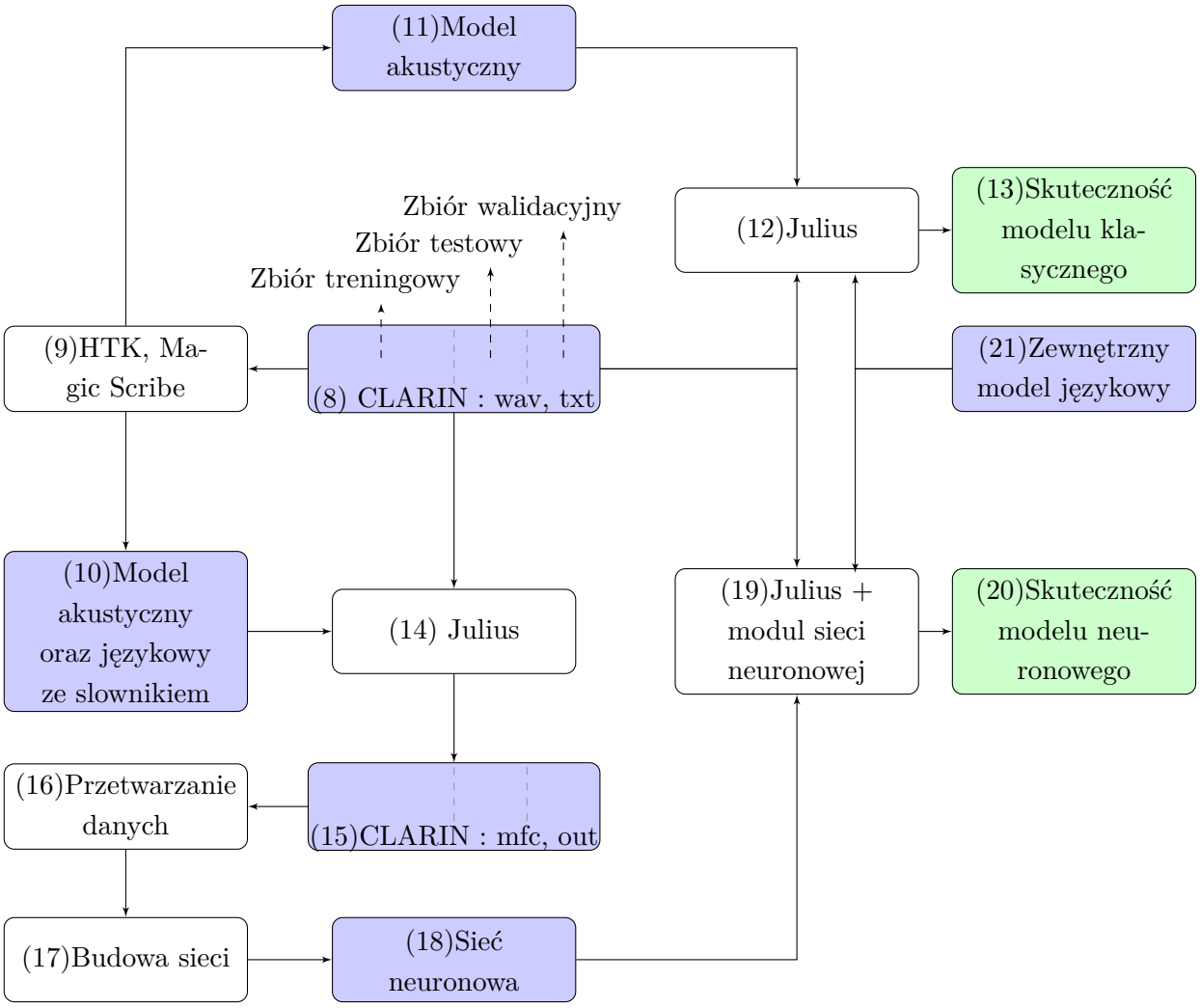
[3]Pliki zawierające wektory cech *MFSC*, zbudowane zgodnie z opisem w rozdziale 2.3.2..
[4]Pliki zawierające dopasowanie stanów do ramek.

- Korzystając z pakietu HTK i systemu Magic Scribe (9) wyuczono szereg klasycznych modeli akustycznych (11) , modele te zostały wytrenowane jedynie na *zbiorze treningowym* z korpusu (8)

- Następnie wykorzystując modele (11) dokonano testów skuteczności rozpoznawania *zbioru testowego* z korpusu (8) dekoderem (12) . W efekcie otrzymano skuteczność (13) .

Jako miarę skuteczności zastosowano metrykę *WER* (*Word Error Rate*), zdefiniowaną wzorem 4.1, gdzie S jest liczbą zamienionych słów, D liczbą usuniętych słów, I liczbą dodanych słów, natomiast N całkowita liczbą słów z referencyjnego zdania. Jest to metryka analogiczna do *odległości Levenshteina*, ale jest zdefiniowana na poziomie całych słów, a nie liter.

$$WER = \frac{S + D + I}{N} \tag{4.1}$$

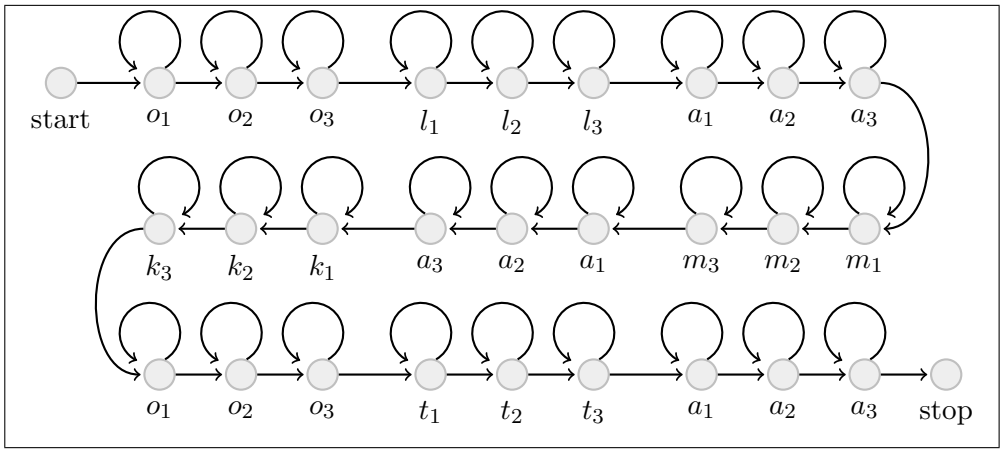


Rysunek 4.3: Zależności pomiędzy wykorzystanymi komponentami. Na fioletowo zaznaczono dane oraz modele, białe są programy i skrypty, kolorem zielonym oznaczono pomiary skuteczności.

4.3. Budowa modeli klasycznych

Klasyczne modele Gaussowkie, zaznaczone w bloku (11) na schemacie 4.3, zostały zbudowane z wykorzystaniem pakietu *HTK* oraz opakowującego go systemu *Magic Scribe*. Do treningu użyto zbioru treningowego z korpusu *Clarin*. Model akustyczny uczony był zgodnie z zaleceniami opisanymi w *HTK Book*[14], procedurę uczenia zapisano w pseudokodzie na rysunku 4.5. Pierwszym krokiem procedury jest zainicjalizowanie parametrów prostego modelu unifonowego, w którym mikstury gaussowskie, zgodne z opisem w rozdziale 2.4.4., składają się z jednego komponentu. Następnie w pierwszej pętli wykonuje się po trzy iteracje re-estymacji parametrów, która przebiega zgodnie z algorytmem Bauma-Welcha opisanym w rozdziale 2.4.3.

oraz 2.4.4.. Podczas re-estymacji przechodzi się przez wszystkie dane treningowe ze zbioru S. Podczas przechodzenia wyliczany jest *Force alignment*, a następnie na jego podstawie estymowane są nowe parametry. *Force alignment* polega na dopasowaniu ramek do zadanego ciągu stanów. Ciąg ten odpowiada transkrypcji, którą dostaje się podczas treningu razem z nagraniem. W praktyce *Force alignment* polega na wymuszeniu przejścia, w fazie E algorytmu *Buma-Welcha*, przez automat odpowiadający transkrypcji nagrania. Schemat 4.4 pokazuje przykład automatu dla zdania *Ola ma kota* wymuszającego przejście przez odpowiednie stany. Automat ten jest jednym łańcuchem i w przeciwieństwie do automatu ze schematu 2.15, wykorzystywanego przez dekoder, uniemożliwia swobodny wybór kolejnych słów. Kolejne re-estymacje powodują zbieganie modelu do wytrenowanej postaci. Po trzech iteracjach, kiedy model ma już pewną wiedzę, rozbudowuje się mikstury Gaussowskie o nowe komponenty. Po zakończeniu pierwszej pętli następuje przekształcenie modelu unifonowego w trifonowy. Początkowo trifony o wspólnym centralnym fonemie współdzielą te same stany. Dopiero w następnym kroku wykonuje się algorytm *Tree-based clustering* z rozdziału 3.2.2., w wyniku którego zwiększa się liczba stanów modelu akustycznego. W kolejnych krokach ponownie re-estymuje się parametry i rozbudowuje mikstury Gaussowskie. Łącznie wykonuje się 36 iteracji reestymacji parametrów. Na koniec zwracany jest wytrenowany model. Wszystkie modele trifonowe z tabeli 4. zostały zbudowane zgodnie z procedurą 4.5. Wyjątek stanowi *M_UNI*, które jest modelem unifonowym, budowanym według uproszczonej procedury. *M_TRI* oraz *M_TRI_OPT* to dokładnie te same modele, ale *M_TRI_OPT* odpowiada wynikowi otrzymanemu dla najlepszych, znalezionych parametrów dekodowania, które wynoszą 20 -15 20 -15. Opis parametrów dekodowania znajduje się w rozdziale 2.4.6..



Rysunek 4.4: Automat dla wypowiedzi treningowej z transkrypcją *Ola ma kota*, stworzony na potrzeby *Force alignmentu*

Require: S = zbiór par $(wypowiedz, transkrypcja)$
1: $M_{acc} = Init(S)$
2: **for** $i = 1$ to 3 **do**
3: $M_{acc} = ReEstimate(M_{acc}, S)$
4: $M_{acc} = ReEstimate(M_{acc}, S)$
5: $M_{acc} = ReEstimate(M_{acc}, S)$
6: $M_{acc} = ExtendGMM(M_{acc})$
7: **end for**
8: $M_{acc} = CreateTriFones(M_{acc})$
9: $M_{acc} = TreeBasedClustering(M_{acc})$
10: **for** $i = 4$ to 12 **do**
11: $M_{acc} = ReEstimate(M_{acc}, S)$
12: $M_{acc} = ReEstimate(M_{acc}, S)$
13: $M_{acc} = ReEstimate(M_{acc}, S)$
14: $M_{acc} = ExtendGMM(M_{acc})$
15: **end for**
16: $M_{acc} = ReEstimate(M_{acc}, S)$
17: $M_{acc} = ReEstimate(M_{acc}, S)$
18: $M_{acc} = ReEstimate(M_{acc}, S)$
19: **return** M_{acc};

Rysunek 4.5: Procedura uczenia trifonowych modeli akustycznych.

4.4. Szczegóły budowy modeli neuronowych

Wspomniane w rozdziale 4.2. modele neuronowe (18) ze schematu 4.3 składają się z dwóch części. Pierwszą jest zbiór stanów Q oraz macierz A, opisująca prawdopodobieństwa przejść pomiędzy stanami. Drugą częścią jest natomiast sama sieć neuronowa, odpowiadająca rodzinie funkcji B wyznaczającej prawdopodobieństwa obserwacji. Każdy model neuronowy jest budowany w oparciu o wcześniej wytrenowany klasyczny model Gaussowski. Z klasycznego modelu brane sa, beż żadnych modyfikacji, stany Q oraz macierz A. Następnie w oparciu o zbiór Q definiuje się liczbę neuronów w warstwie wyjściowej sieci. Sieć ma pełnić rolę estymatora prawdopodobieństw obserwacji, zgodnie z opisem w rozdziale 3.1.. Podczas uczenia sieci, pokazuje się jej pary wartości *wektor cech* $\longrightarrow$ *stan modelu Markowa*, które tworzą zbiór treningowy sieci, odpowiadający blokowi (15) ze schematu 4.3. Każdy model neuronowy musi mieć swój zbiór treningowy, gdyż jest on uzależniony od modelu klasycznego, na podstawie, którego uczona jest sieć. Wynika to z faktu, że każdy model Gaussowski ma inne mapowanie trifonów na stany, co definiuje zbiór treningowy sieci. Opis mapowania trifonów na stany oraz jego wpływ na model akustyczny został opisany w rozdziale 3.2.1..

4.5. Optymalizacja parametrów procesu dekodowania

Celem pierwszego eksperymentu było znalezienie wartości *wagi modelu językowego* oraz *kary za wstawienie słowa*, które dają dobrą skuteczność rozpoznawania mowy. Wpływ tych parametrów na proces dekodowania został opisany w rozdziale 2.4.6.. W trakcie eksperymentu sprawdzono skuteczność rozpoznawania na zbiorze testowym z korpusu *Clarin* dla różnych wartości *wagi* oraz *kary*. Eksperymentowi poddano sieci neuronowe wykorzystane w testach z rozdziału 4.6.. Wyniki eksperymentu zostały zamieszczone w tabeli 4.5.. Bazując na otrzymanych rezultatach uznano parametry $12, -10, 15, -10$ za optymalne. Dały one najlepszą skuteczność w ujęciu wszystkich pomiarów, w szczególności dla sieci o 231 oraz 147 stanach.

4.6. Wpływ liczby stanów na skuteczność rozpoznawania

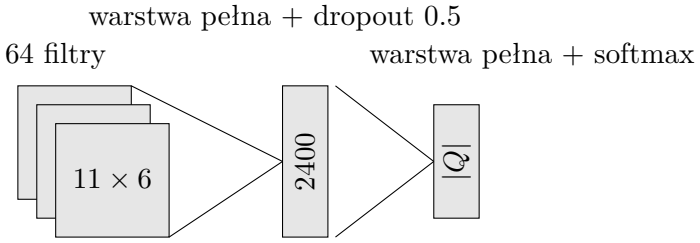
W celu zweryfikowania głównej tezy niniejszej pracy przeprowadzono porównanie modeli o różnym stopniu zdławienia, dokonanym zgodnie z opisem w rozdziale 3.2.2.. Manipulując liczbą obserwacji wymaganych do podziału trifonu, zbudowano serię klasycznych modeli akustycznych o różnej liczbie stanów. Następnie dla każdego z nich wytrenowano sieć neuronową zgodnie z filozofią opisaną w rozdziale 3.1.. Wszystkie wykorzystane w tym eksperymencie siec neuronowe miały taką samą architekturę. Składały się z jednej warstwy splotowej zbudowanej z 64 filtrów, filtry były trzykanałowe i miały wymiary 11×6, co było związane w szerokością kontekstu i odpowiada wartości $kontekst \cdot 2 + 1 \times 6$. Następnie dodano jedną klasyczną warstwę ukrytą, składającą się z 2400 neuronów oraz warstwę wyjściową, której szerokość była równa liczbie stanów danego modelu akustycznego. Warstwa środkowa miała dodatkowo dropout w współczynniku 0.5. Funkcją aktywacji było *rectify* opisane wzorem $\max(x, 0)$, ostatnia warstwa miała funkcję *softmax*, aby nadać wyjściu sieci charakter prawdopodobieństwa. Każda ze zbudowanych sieci neuronowych jest kompatybilna z jednym modelem klasycznym, zgodnie z opisem w rozdziale 4.4.. Schemat 4.6. obrazuje architekturę sieci. Powstałe pary modeli akustycznych przebadano na testowym podzbiorze korpusu językowego *Clarin*. Podczas procesu dekodowania wykorzystano wartości *wagi* oraz *kary* równe optymalnym parametrom z eksperymentu opisanego w rozdziale 4.5.. Otrzymane wyniki zamieszczono w tabeli 4.6.. Wykres 4.6. obrazuje, w sposób graficzny, wpływ liczby stanów na skuteczność rozpoznawania dla modeli neuronowych oraz klasycznych.

Przeprowadzony eksperyment pokazał, że rzeczywiście zwiększenie liczby wyjść sieci neuronowej pozytywnie wpływa na skuteczność rozpoznawania. Zaskakujące jednak okazało się, że dla modeli o ponad 231 neuronach obserwuje się gwałtowny spadek skuteczności. Niemniej jednak, model o 231 neuronach uzyskał 83.48% skuteczności, co daje 1.9% punktu procentowego poprawy względem najlepszego stan-

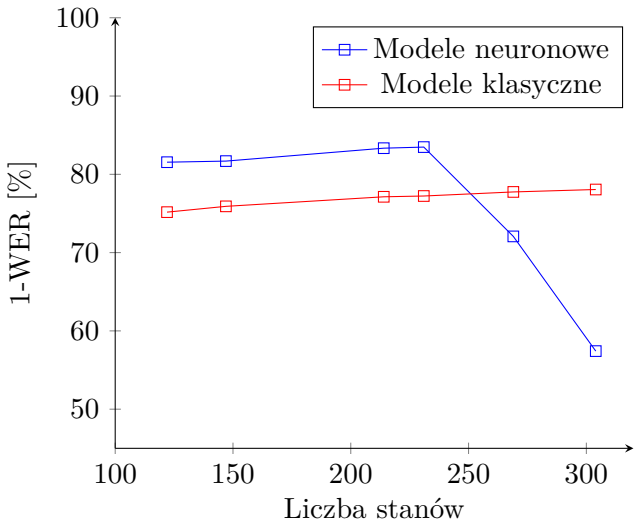
Nazwa modelu (zgodna z tabelą na rysunku 4.)	Liczba stanów	Parametry pierwszego przejścia	Parametry drugiego przejścia	Skuteczność [%]
M_UNI	-	13 -8	13 -9	81.09
		12 -7	**15 -9**	**81.55**
		12 -7	12 -7	81.32
		11 -7	11 -7	81.03
		10 -7	10 -7	80.72
		7 -8	7 -8	78.19
		5 -8	5 -8	73.37
M_180000	122	**13 -8**	**13 -9**	**81.69**
		12 -7	15 -9	81.48
		12 -7	12 -7	81.61
		11 -7	11 -7	81.50
		10 -7	10 -7	81.23
		7 -8	7 -8	78.47
		5 -8	5 -8	73.9
		18 -7	16 -12	79.04
M_37000	214	13 -8	13 -9	81.44
		12 -7	**15 -9**	**83.39**
		12 -7.5	15 -9.5	83.33
		12 -7 16	-9.5	83.04
		12 -7 12	-7	81.09
		11 -7 11	-7	80.61
		18 -7 16	-12	79.76
M_25000	304	13 -8	13 -9	54.30
		12 -7	**15 -9**	**57.42**
		12 -7	12 -7	52.58
		7 -8	7 -8	39.58
		18 -7	16 -12	55.14
M_33000	231	18 -7	16 -12	79.90
		12 -7	15 -9	83.37
		12 -7	15 -9	83.40
		12 -10	**15 -10**	**83.48***
		13 -10	15 -10	83.37
		7 -8	7 -8	74.63
		12 -7	12 -7	80.68
		13 -11	15 -11	83.34
		14 -12	15 -12	82.19
M_70000	147	18 -7	16 -12	79.64
		12 -7	15 -9	83.18
		12 -10	**15 -10**	**83.24**
		7 -8	7 -8	78.59
		11 -10	11 -10	81.12
		11 -9	11 -9	81.32
		12 -9	12 -9	81.55

Rysunek 4.6: Wpływ parametrów dekodowania na skuteczność rozpoznawania

dardowego modelu (M_TRI) i 0.05% punktu procentowego poprawy względem modelu o zoptymalizowanych parametrach dekodowania (M_TRI_OPT).



Rysunek 4.7: Architektura sieci przy badaniu wpływu liczby stanów na skuteczność rozpoznawania mowy.



Rysunek 4.8: Wpływ liczby stanów na skuteczność rozpoznawania.

Liczba stanów	Min liczba obserwacji	Typ	Skuteczność modelu GMM	Skuteczność modelu CNN
120	-	UNI	71.86	81.55
122	180000	TRI	75.17	81.69
147	70000	TRI	75.91	83.24
214	37000	TRI	77.13	83.39
231	**33000**	**TRI**	**77.23**	**83.48**
269	29000	TRI	77.75	72.07
304	25000	TRI	78.06	57.42

Rysunek 4.9: Wpływ liczby stanów na skuteczność rozpoznawania mowy.

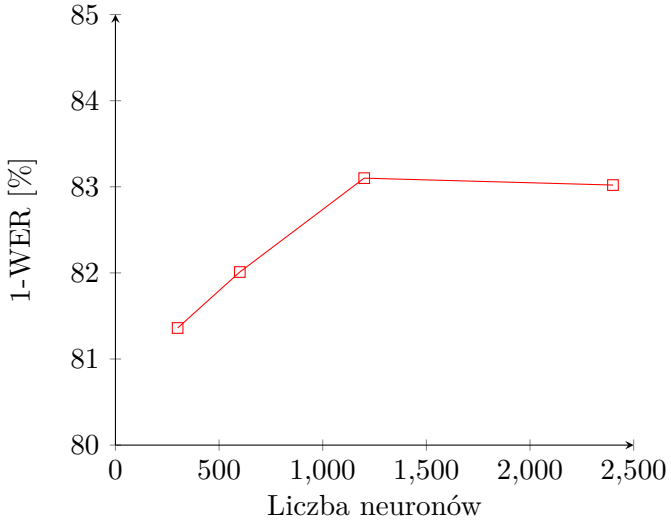
Liczba neuronów w warstwie ukrytej	Liczba filtrów	Skuteczność
2400	32	83.02
1200	**32**	**83.10**
600	32	82.01
300	32	81.36

Rysunek 4.10: Wpływ liczby neuronów na skuteczność rozpoznawania.

4.7. Wpływ głębokości/architektury sieci na skuteczność rozpoznawania

Celem tego eksperymentu jest sprawdzenie, jak architektura sieci, w tym liczba neuronów wpływa na skuteczność rozpoznawania. W pierwszej części eksperymentu zbudowano szereg modeli neuronowych bazujących na jednym modelu Gaussowskim o 231 stanach. Modele miały taki sam kontekst równy 5 oraz tę samą liczbę i kształt filtrów. Sieci miały taką sama architekturę jak w rozdziale 4.6.. Jedynym parametrem, jaki uległ zmianie, była liczba neuronów w środkowej, pełnej warstwie.

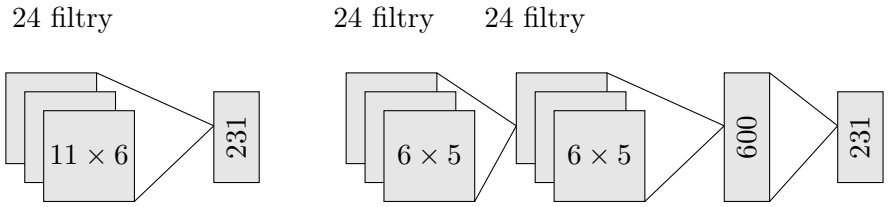
Otrzymane wyniki zamieszczono w tabeli 4.10 i zilustrowano wykresem 4.7.. Przeprowadzony test pokazał, że liczba neuronów w warstwie ukrytej ma wpływ na skuteczność. Optymalną wartością wydaje się 1200 neuronów, dalsze rozszerzanie warstwy jedynie utrudnia trening sieci.



Rysunek 4.11: Wpływ liczby neuronów warstwy ukrytej na skuteczność rozpoznawania.

W drugiej części zbudowano dwie kolejne sieci o innej architekturze. Pierwsza z nich była uproszczona, nie zawierała ukrytej warstwy pełnej, miała 24 filtry. Druga sieć miała dodatkową warstwę splotową, obie warstwy splotowe miały po 24 filtry,

warstwa pełna miała 600 neuronów. Zmniejszenie szerokości warstwo podyktowane było czasem treningu. Schemat 4.7. przedstawia obie architektury. W przeprowadzonym eksperymencie uproszczona siec uzyskała 77.38% skuteczności, natomiast rozbudowana 82.35%. Oba wyniki są gorsze od najlepszej wartości z pierwszego eksperymentu, wynoszącej 83.10%, co sugeruje że przyjęta w poprzednich eksperymentach architektura ze schematu 4.6., wydaje się być optymalną. Sieci o mniejszej liczbie warstw nie dysponują wystarczająca pojemnością wiedzy, a modele bardziej rozbudowane są zbyt trudne do wyuczenia.

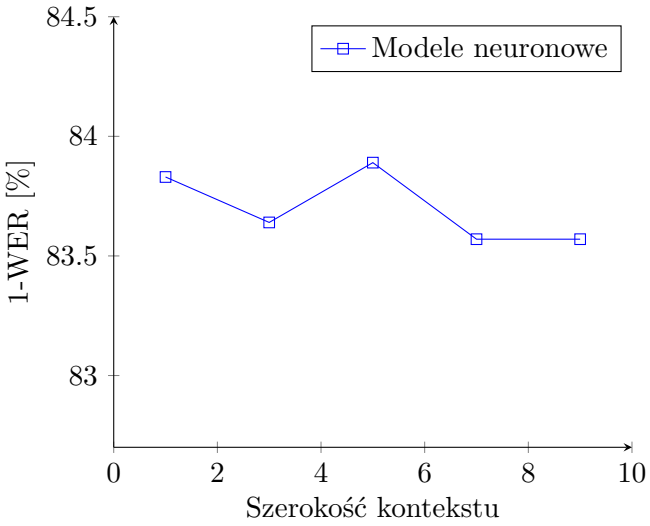


Rysunek 4.12: Architektura sieci przy badaniu wpływu architektury na skuteczność rozpoznawania mowy.

4.8. Wpływ szerokości kontekstu na skuteczność rozpoznawania

Typowa szybkość mówienia wynosi $12 - 14$ głosek na sekundę, w konsekwencji przeciętna głoska zajmuje $7 - 8$ ramek. W związku z tym, uzasadnione wydaje się przekazywanie do wektora cech wielu kolejnych ramek. W tym eksperymencie sprawdzono wpływ liczby ramek, z których budowany jest wektor cech na skuteczność rozpoznawania mowy z wykorzystaniem sieci neuronowych. Architektura wykorzystanej siec była podobna do architektury z eksperyment opisanego w rozdziale 4.6.. Jednak teraz liczba neuronów w warstwie wyjściowej była ustalona, gdyż bazowano na jednym modelu akustycznym o 231 stanach. Liczba filtrów pozostała taka sama i wyniosła 64. Zmieniał się natomiast kształt filtrów, który wynosił $kontekst \cdot 2 + 1 \times 6$. Ponadto zamiast funkcji nieliniowości *rectify* wykorzystano funkcję *leaky rectify* opisana wzorem $\max(x, 0.01x)$. Wykres 4.8. przedstawia wyniki przeprowadzonych eksperymentów.

Na podstawie otrzymanych wyników ciężko jednoznacznie określić wpływ szerokości kontekstu na skuteczność rozpoznawania. Najlepszy wynik otrzymano dla kontekstu 5, co odpowiada 11 ramkom przekazywanym do wektora cech.



Rysunek 4.13: Wpływ szerokości kontekstu na skuteczność rozpoznawania mowy.

4.9. Inne czynniki wpływające na skuteczność rozpoznawania

Oprócz parametrów wspomnianych we wcześniejszych rozdziałach, sprawdzono jaki wpływ na skuteczność rozpoznawania ma funkcja aktywacji, pomieszanie danych uczących oraz uwzględnianie prawdopodobieństwo a priori opisanego w rozdziale 3.1.3..

W eksperymencie opisanym w rozdziale 4.6. wykorzystano funkcję aktywacji opisaną wzorem $\max(x, 0)$. Celem tego eksperymentu jest sprawdzenie czy funkcja *leaky rectify* opisana wzorem $\max(x, 0.01x)$ umożliwia wytrenowanie skuteczniejszej sieci. W założeniach czynniki $0.01x$ daje sieci informację z neuronów o negatywnym pobudzeniu. Jest to różnica w stosunku do funkcji *rectify*, która zwraca wtedy zawsze 0 i nie propaguje żadnej informacji.

W wcześniejszych eksperymentach dane treningowe były ułożone w uporządkowany sposób. Wektory cech pochodzące od jednego mówcy następowały po sobie. Podobnie, wektory dla kolejnych ramek pochodzących z jednej wypowiedzi, też ułożone było kolejno po sobie. Celem eksperymentu jest sprawdzenie czy po przemieszaniu wektorów cech można uzyskać lepszą skuteczność. Efektem przemieszania, kolejne, podobne wektory nie są razem pokazywane sieci podczas treningu.

Oba czynniki testowane były przy takich samych architekturach, ale innych liczbach neuronów. Tabele 4.9. pokazuje wyniki eksperymentu. Okazało się, że zarówno stosowanie funkcji *leaky rectify*, jak i mieszanie danych treningowych poprawia skuteczność. Funkcja *leaky rectify* dała względną poprawę o 2.97%, natomiast mieszanie danych dało 3.2% poprawy. Na skutek pozytywnych wyników eksperymentu, do ostatniej części testu zbudowano sieć łączącą w sobie oba badane elementy. Po-

wstały model agreguje wszystkie obserwacje uzyskane z przeprowadzonych testów i stanowi najlepszą uzyskaną konfigurację. Ma 64 filtry, 2400 neuronów w warstwie ukrytej, funkcję aktywacji *leaky rectify* oraz architekturę z rysunku 4.6.. W celu sprawdzenia wpływu prawdopodobieństwa a priori uruchomiono rozpoznawanie korzystające ze zbudowanej sieci z uwzględnieniem członu a priori oraz bez niego. Uzyskano odpowiednio 84.71% oraz 85.75% skuteczności. Oba wyniki sa lepsze niż najlepszy klasyczny model *M_TRI_OPT* o kolejno 1.27 i 2.31 punktu procentowego. Zaskakującym jest, że uwzględnianie członu a priori pogarsza skuteczność.

Funkcja nieliniowości	Dane przemieszane	Ppb a priori	Liczba filtrów	Liczba neuronów	Skuteczność
recify	nie	tak	64	2400	83.50
leaky rectify	nie	tak	64	2400	83.89
leaky rectify	nie	tak	32	1200	83.10
leaky rectify	tak	tak	32	1200	83.64
leaky rectify	tak	tak	64	2400	84.71
leaky rectify	tak	nie	64	2400	85.75

Rysunek 4.14: Wpływ liczby stanów na skuteczność rozpoznawania mowy.

4.10. Podsumowanie

Na podstawie przeprowadzonych eksperymentów, można jednoznacznie potwierdzić tezę niniejszej pracy magisterskiej. **Wykorzystując splotowe sieci neuronowe rozpoznające *trifony*, jako estymator prawdopodobieństw stanów modelu Markowa, można uzyskać poprawę skuteczności, względem analogicznego podejścia wykorzystującego *unifony*.** W eksperymencie 4.6. uzyskano 1.93% punktu procentowego poprawy względem sieci neuronowej unifonowej, co odpowiada 10.46% względnej redukcji błędu. Ponadto, w eksperymencie 4.9. udało się zbudować sieć uzyskującą 2.31 punktu procentowego poprawy (13.95% względnej redukcji błędu) względem najlepszego klasycznego modelu *M_TRI_OPT*. Nieoczekiwany okazał się gwałtowny spadek skuteczności dla modeli neuronowych o więcej niż 231 stanach. Efekt ten można uzasadnić tym, że mając ograniczony i stosunkowo mały zbiór treningowy, sieć nie uzyskuje zdolności uogólniania wiedzy powyżej pewnej liczby wyjść. Zaskakujące również się okazało, że skuteczność pogarsza uwzględnianie prawdopodobieństwa a priori, dzięki któremu sieć zwracała takie prawdopodobieństwo, jakiego spodziewa się ukryty model Markowa. W przeprowadzonych eksperymentach nie uzyskano poprawy, przy zwiększaniu liczby warstw względem proponowanej architektury ze schematu 4.6..

Wyniki przeprowadzonych eksperymentów dały pozytywne rezultaty zachęcająca do dalszych badań. W przyszłości warto by było skupić się na budowie sieci uczonej na znacznie większym korpusie, jednak wymaga to odpowiednich zasobów

sprzętowych. Uczenia jednego modelu neuronowego na korpusie *Clarin* trwało 2-3 dni. W przypadku większych zbiorów treningowych czas ten może się parokrotnie wydłużyć. Jednak mają więcej danych treningowych można powtórzyć próbę budowy sieci o głębszej architekturze lub sieci o większej liczbie wyjść. Można tez spróbować budowy modeli neuronowych na sztucznie powiększonym zbiorze, stosując metody augmentacji.

Rozdział 5.

Podziękowania

Autor wyraża podziękowania Panu Piotrowi Radlińskiemu, prezesowi firmy *Radcomp Integral Sp z o.o.*, za udostępnienie oprogramowania *Magic Scribe* przy pomocy, którego zgromadzono dane do wykonania eksperymentów przedstawionych w niniejszej pracy.

Bibliografia

[1] George E. Dahl Abdel-rahman Mohamed and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE*, 2012.

[2] Geoffrey E. Hinton Alex Krizhevsky, Ilya Sutskever. Imagenet classification with deep convolutional, neural networks. 2012.

[3] Rico Petrick Patrick Proba Ahmed Malatawy Christian Gaida, Patrick Lange and David Suendermann-Oeft. Comparing open-source speech recognition toolkits. 2014.

[4] James H. Martin Daniel Jurafsjy. Speech and language processing. 2009.

[5] Ronald J. Williams David E. Rumelhart, Geoffrey E. Hinton. Learning representations by back-propagation error. 1986.

[6] Mihalis Siafarikas Nikos Fakotakis Iosif Mporas, Todor Ganchev. Comparison of speech features on the speech recognition task. *Journal of Computer Science*, 2007.

[7] P. C. Woodland J. J. Odell and S. J. Young. Tree-based state clustering for large vocabulary speech recognition. *IEEE*, 1994.

[8] Rich Caruana Gregor Urban ShengjieWang Özlem Aslan Matthai Philipose Matthew Richardson Charles Sutton Krzysztof J. Geras1, Abdel-rahman Mohamed. Blending lstms into cnns. 2016.

[9] Rabiner L. A tutorial on hidden markov models and selected applications in speach recognition. *IEEE*, 1989.

[10] Akinobu Lee. The julius book.

[11] P. Mermelstein S. Davis. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE*, 1980.

[12] Akinobu Lee; Tatsuya Kawahara; Kiyohiro Shikano. Julius — an open source real-time large vocabulary recognition engine. 2001.

[13] Maria Steffen-Batogowa. Automatyzacja transkrypcji fonematycznej tekstów polskich. 1975.

[14] Mark Gales Thomas Hain Dan Kershaw Xunying Liu Gareth Moore Julian Odell Dave Ollason Dan Povey Valtcho Valtchev Phil Woodland Steve Young, Gunnar Evermann. The htk book.