POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Praca dyplomowa magisterska

Model akustyczny języka angielskiego na potrzeby automatycznego rozpoznawania mowy

Autor: Łukasz Olczak
Kierujący pracą: dr inż. Krzysztof Cyran

Gliwice, wrzesień 2009

Spis treści

1 Wstęp

Celem niniejszej pracy jest budowa modelu akustycznego języka angielskiego na potrzeby automatycznego rozpoznawania mowy. Praca zawiera zarówno aspekty teoretyczne nt. budowy modelu akustycznego, jak i techniczne szczegóły zaimplementowanych algorytmów.

W ramach pracy dyplomowej zaimplementowałem kompletny system ASR (ang. *automatic speech recognition*). Implementacja zawiera moduły przeznaczone do budowy różnorodnych modeli akustycznych w oparciu o ukryte modele Markowa oraz moduł odpowiedzialny za rozpoznawanie mowy. Testy poprawności zrealizowanych algorytmów zostały przeprowadzone na podstawie korpusu mowy języka angielskiego.

W rozdziale drugim przedstawiono podstawowe zagadnienia teoretyczne z dziedziny rozpoznawania obiektów. Rozdział trzeci zawiera szczegółowy opis ukrytych modeli Markowa, będących fundamentem współczesnych systemów rozpoznawania mowy. Rozszerzenia ukrytych modeli Markowa wprowadzone w zadaniu rozpoznawania mowy zostały opisane w rozdziale czwartym. W rozdziale piątym omówiono ideę ekstrakcji cech akustycznych z sygnału mowy. Rozdział szósty omawia trening dyskryminujący. Najważniejsze zagadnienia specyfikacji technicznej zaimplementowanego systemu ASR znajdują się w rozdziale siódmym. Budowa i parametryzacja modelu akustycznego została opisana w rozdziale ósmym. Rezultaty testów systemu umieszczono w rozdziale dziewiątym.

2 Preliminaria

2.1 Wprowadzenie

Termin rozpoznawanie obiektów (ang. *pattern recognition*) pojawił się po raz pierwszy na początku lat 50. Wraz z rozwojem pierwszych maszyn cyfrowych, umożliwiających szybkie przetwarzanie informacji, podjęto pierwsze próby automatyzacji problemu rozpoznawania obiektów. W systematycznie rosnącej mocy obliczeniowej pierwszych komputerów zaczęto upatrywać kres poglądu głoszącego, że rozpoznawanie obiektów jest wyłącznie domeną umysłu ludzkiego. Pojęcie rozpoznawania obiektów zawiera wiele różnorodnych problemów poczynając od diagnostyki medycznej kończąc na rozpoznawaniu twarzy, czy mowy. Większość z tych problemów człowiek rozwiązuję całkowicie podświadomie. Umiejętność rozpoznawania pozwala nam na podstawie wyselekcjonowanych (często podświadomie), charakterystycznych właściwości obiektów przypisać im konkretne znaczenie, a więc przyporządkować obiekt do pewnej predefiniowanej klasy. Jednakże w przypadku podejścia algorytmicznego, stosowanego w maszynach cyfrowych, problem nie jest już tak trywialny. Projektant systemu musi zatem odnaleźć formalną metodę rozwiązania zadania rozpoznawania obiektów możliwą do implementacji na maszynie cyfrowej. Należy pamiętać jednak, że istota komputerowych metod rozpoznawania obiektów bazuje na ogromnej mocy obliczeniowej tych maszyn oraz zdolności przetwarzania i pamiętania wielowymiarowych danych.

W rozdziale tym przedstawiam kluczowe koncepcje rozpoznawania obiektów. Definiuję problem rozpoznawania w sposób formalny, objaśniam podstawowe terminy stosowane w pracy oraz przedstawiam ogólny opis metod statystycznych stosowanych w rozwiązywaniu problemów rozpoznawania obiektów. Rozdział kończę formalną definicją problemu rozpoznawania mowy, któremu poświęcona jest niniejsza praca magisterska.

2.2 Definicja problemu rozpoznawania

Pod pojęciem rozpoznawania rozumiemy przypisanie obiektu do jednej z wcześniej zdefiniowanych klas. Jednak pojęcie rozpoznawania jest terminem dość ogólnym rozumianym jako całokształt procedury polegającej na zaklasyfikowaniu obiektu do jednej z klas. A więc rozpoznawanie obiektów obejmuje ekstrakcję cech (ang. *feature extraction*), normalizację danych wejściowych, wstępne przetwarzanie (ang. *pre-processing*) oraz klasyfikację. Natomiast przez

klasyfikację będziemy rozumieć jedynie czynność decyzyjną polegającą na przyporządkowaniu rozpoznawanemu obiektowi odpowiedniej klasy.

Abstrahując od istoty konkretnego problemu rozpoznawania zakładam, że każdy obiekt podlegający klasyfikacji reprezentowany jest przez wektor atrybutów mierzalnych bezpośrednio lub pośrednio nazywanych dalej cechami obiektu. Jednakże w większości przypadków liczba składników wektora atrybutów może sięgać setek tysięcy, wydłużając tym samym czas podejmowania decyzji podczas klasyfikacji obiektu. Z tego powodu konieczna jest redukcja liczby wymiarów wektora opisującego obiekt. W systemach rozpoznawania obiektów redukcję taką uzyskujemy przez selekcję cech (ang. *feature selection*) lub ekstrakcję cech (ang. *feature extraction*). Pierwszy proces polega na selekcji tych cech, które są istotne dla procesu podejmowania decyzji podczas klasyfikacji (często dokonywany na podstawie wiedzy eksperta dziedzinowego). Kryterium wyboru może tutaj być również łatwość obserwacji czy koszt wykonania pomiaru. Druga metoda polega na transformacji wektora wejściowego za pomocą zdefiniowanego przekształcenia matematycznego. Przekształcenie to redukuje liczbę wymiarów oraz usuwa pewną porcję informacji nieistotną z punktu rozpoznawania obiektu. Ekstrakcji cech dla sygnału mowy został poświęcony rozdział 5. Warto w tym miejscu zaznaczyć, że algorytm ekstrakcji (lub selekcji) cech jest często jedyną wiedzą *a priori* wykorzystywaną podczas rozpoznawania, dlatego etap ten należy przeprowadzić starannie, gdyż determinuje on często rezultat procesu rozpoznawania. Wektor cech będziemy od tego momentu definiowali w następujący sposób:

$$\bar{x} = [x_1, x_2, \dots, x_d]^T \tag{2.1}$$

gdzie d jest liczbą wymiarów, T jest symbolem transpozycji, $\bar{x}$ jest wektorem kolumnowym przyjmującym wartości z przestrzeni cech $\mathcal{X}$. Ze względu na ogólność rozważań przyjmujemy $\mathcal{X} \subseteq R^d$.

Załóżmy, że podczas klasyfikacji rozpatrujemy *k* klas, wtedy

$$\mathcal{C} = \{C_1, C_2, \dots, C_k\} \tag{2.2}$$

jest zbiorem klas. Zadanie klasyfikacji polega zatem na przyporządkowaniu obiektowi reprezentowanemu przez wektor $\bar{x}$ klasy ze zbioru $\mathcal{C}$. Procedurę podejmującą decyzję o przypisaniu obiektowi odpowiedniej klasy będziemy nazywać regułą decyzyjną lub algorytmem klasyfikacji. Implementację takiego algorytmu nazwiemy klasyfikatorem lub recognizer'em.

2.3 Regiony decyzyjne

Reguła decyzyjna przyporządkowuje każdemu wektorowi cech klasę ze zbioru $\mathcal{C}$, a więc reguła decyzyjna jest funkcją odwzorowującą przestrzeń wektora cech $\mathcal{X}$ w zbiór predefiniowanych klas $\mathcal{C}$:

$$\rho: X \to C \tag{2.3}$$

Zatem funkcja ρ dzieli przestrzeń $\mathcal{X}$ na tak zwane regiony decyzyjne (ang. *decision regions*):

$$D_x^i = \{x \in X: \rho(x) = C_i\} \tag{2.4}$$

Regiony te są parami rozłączne i pokrywają całą przestrzeń $\mathcal{X}$. Regiony decyzyjne są rozdzielone granicami decyzyjnymi (ang. *decision boundaries*). A więc regiony decyzyjne w pełni opisują wykorzystywany podczas klasyfikacji algorytm rozpoznawania.

2.4 Funkcje dyskryminujące

Ze względów praktycznych podczas klasyfikacji korzystamy zazwyczaj ze zbioru funkcji dyskryminujących.

$$y_i: \mathcal{X} \rightarrow R, i \in C \tag{2.5}$$

Funkcje dyskryminujące przypisują wektorowi cech obiektu pewną wartość liczbową, którą możemy zinterpretować jako ocenę (ang. *score*) przynależności obiektu do danej klasy. Wartość funkcji dla poprawnej klasy powinna wyróżniać się na tle pozostałych funkcji. Przykładem może być tutaj klasyfikator skonstruowany w myśl reguły Bayes'a, którego funkcje dyskryminujące estymują prawdopodobieństwa *a posteriori*.

Możemy teraz zredefiniować proces klasyfikacji wykorzystując pojęcie funkcji dyskryminujących. Zakładamy, że istnieją funkcje dyskryminujące, po jednej dla każdej klasy $y_1(x), \dots, y_k(x)$, wtedy wektor cech $\mathcal{X}$ przyporządkujemy do klasy C_i jeśli:

$$y_i(x) > y_j(x), dla\ wszystkich\ j \neq i \tag{2.6}$$

Zatem funkcje dyskryminujące dzielą przestrzeń cech na regiony decyzyjne takie że:

$$D_x^i = \{x \in X: y_i(x) = max_{j \in C}\, y_j(x)\} \tag{2.7}$$

Natomiast granicę decyzyjną rozdzielającą regiony decyzyjne klas i oraz j definiujemy następująco:

$$B_x(i,j) = \{x \in X: y_i(x) = y_j(x)\}, dla\ wszyskich\ i,j \in C \tag{2.8}$$

Granice decyzyjne posiadają kształt płaszczyzn. W zależności od typu funkcji dyskryminujących oraz liczby wymiarów przestrzeni cech są to hiperpłaszczyzny, płaszczyzny pierwszego, drugiego rzędu itd.

2.4.1 Liniowe funkcje dyskryminujące

Najbardziej podstawowym typem funkcji dyskryminujących jest postać liniowa:

$$y_i(x) = w_0^i + w_1^i x_1 + w_2^i x_2 + \cdots + w_d^i x_d = \sum_{j=1}^{d} w_j^i x_j + w_0^i \tag{2.9}$$

Wyznaczone obszary decyzyjne przez liniowe funkcje dyskryminujące są rozdzielone fragmentami prostych (dla przypadku dwuwymiarowego), płaszczyzn (dla trzech wymiarów) lub hiperpłaszczyzn (dla czterech i więcej wymiarów). A więc klasyfikator ten rozwiązuje jedynie problemy separowalne liniowo. Otrzymane w ten sposób regiony decyzyjne są wypukłe, oznacza to że dwa dowolne punkty należące do jednego obszaru decyzyjnego można połączyć prostą w całości leżącą w tym obszarze decyzyjnym. Dodatkowo struktura klasyfikatora bazującego na liniowych funkcjach dyskryminujących odpowiada schematowi neuronu.

2.4.2 Ogólna postać funkcji dyskryminujących

W zależności od typu funkcji dyskryminującej mamy do czynienia z zupełnie innym klasyfikatorem. Dla przykładu klasyfikator kwadratowy jest opisany przez funkcje:

$$y_i(x) = \sum_{l=1}^{d}\left(w_0^l x_l + w_1^l x_l x_1 + \cdots + w_d^l x_l x_d\right) + w_0 \tag{2.10}$$

którego powierzchnie rozdzielające regiony decyzyjne są powierzchniami drugiego rzędu.

Jedynym sposobem odseparowania za pomocą funkcji dyskryminujących dowolnych regionów decyzyjnych, często o nieliniowych granicach, jest zdefiniowanie ogólnej postaci funkcji dyskryminującej sprzężonej z nieliniową funkcją $\phi_i(x)$ transformującą przestrzeń cech w przestrzeń separowalną liniowo. Zatem ogólna postać funkcji dyskryminującej wygląda następująco:

$$y_i(x) = \sum_{j=1}^{l} w_j^i \phi_j^i(x) + w_0^i \tag{2.11}$$

Ponieważ funkcje ϕ, zwane często funkcjami bazowymi, są dowolnej postaci toteż powierzchnie rozdzielające regiony decyzyjne możemy modelować w dowolny sposób. Wzór (2.11) jest jedynie uogólnieniem klasyfikatora w wersji liniowej.

Z powodów praktycznych najczęściej stosowanymi funkcjami bazowymi są:

Funkcja progowa

$$\phi(x) = \begin{cases} 0, gdy\ x < 0 \\ 1, gdy\ x \geq 0 \end{cases} \tag{2.12}$$

Funkcja sigmoidalna

$$\phi(x) = \frac{1}{1+e^{-x}} \tag{2.13}$$

Funkcja Gaussa

$$\phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.14}$$

2.5 Klasyfikator probabilistyczny

Klasyfikatory omówione do tej pory zakładały deterministyczny związek pomiędzy wartościami cech obiektów, a klasą do której należy rozpoznawany obiekt. Gdyby związki te były w praktycznych zadaniach klasyfikacji tak jednoznaczne, to algorytmiczne rozwiązanie problemu rozpoznawania nie powodowałoby większych trudności projektantowi. Niestety w większości problemów rozpoznawania, w szczególności w zadaniu rozpoznawania mowy, wartości wektora cech dla obiektów należących do różnych klas mogą przyjmować te same wartości. A więc wartości wektora cech niejednoznacznie określają klasę, do której należy obiekt. Zatem mamy sytuację pokrywających się nawzajem przestrzeni klas (ang. *overlapping class*). Dlatego tak ważna jest selekcja odpowiednich cech rozpoznawanego obiektu, tak aby przestrzeń wektora cech była wyraźnie odseparowana dla poszczególnych klas.

Tym samym reguła decyzyjna (2.3) przyporządkowująca wektorowi cech rozpoznawanego obiektu odpowiednią klasę powinna reprezentować informację niepewną takiego odwzorowania. W tym celu w zdecydowanej większości systemów rozpoznawania wykorzystuje się model probabilistyczny opisujący niepewność związków pomiędzy wektorem cech, a klasą do której należy obiekt. Konstruując model probabilistyczny (statystyczny) zakładamy, że zarówno wektor cech, jak i numer klasy do której należy rozpoznawany obiekt są realizacją zmiennej losowej. Zmienna losowa to – intuicyjnie – zmienna przyjmująca wartości z pewnym prawdopodobieństwem, a więc modeluje niepewność przypisania zmiennej określonej wartości. Podczas budowy modelu statystycznego na potrzeby rozpoznawania obiektów naszą jedyną wiedzą *a priori* jest zazwyczaj rozkład prawdopodobieństwa (tudzież funkcja gęstości rozkładu) lub jedynie kształt rozkładu prawdopodobieństwa zmiennych losowych. W zależności od trafności dobory kształtów rozkładów oraz związków między zmiennymi losowymi otrzymujemy model probabilistyczny, który dobrze lub źle modeluje zadanie rozpoznawania. Warto w tym momencie zwrócić uwagę na fakt, że budowany model powinien dobrze reprezentować proces generowania wektora cech, a nie dokładnie modelować zbiór uczący.

W modelu statystycznym rozpoznawania obiektów zakładamy, że wektor wartości cech opisujący rozpoznawany obiekt oraz klasa do której należy obiekt są realizacją pary zmiennych losowych (X,M). Zmienna losowa X przyjmuje wartości z przestrzeni cech $\mathcal{X}$, natomiast zmienna losowa M przyjmuje wartości ze zbioru klas C.

Zakładamy również, że proces generowania obiektów podlegających rozpoznaniu wygląda następująco: zdarzenie X_l (polegające na wygenerowaniu wektora cech obiektu) może zajść, jeśli zajdzie jedno z wykluczających się zdarzeń $M_1,M_2,\ldots, M_k$. Zdarzenie M_i polega na wyborze klasy i generującej obiekty należące do niej. Zdarzenia $M_1,M_2,\ldots, M_k$ wyłączają się parami oraz

zajście jednego ze zdarzeń M_i jest zdarzeniem pewnym. Zatem prawdopodobieństwo zajścia zdarzenia X_l opisuje następujący wzór:

$$P(X_l) = P(M_1)P(X_l|M_1) + P(M_2)P(X_l|M_2) + \cdots + P(M_k)P(X_l|M_k) \quad (2.15)$$

Doświadczenie polegające na obserwacji (pomiarze) wektora X_l może zajść tylko wtedy, gdy zajdzie jedno z wyłączających się zdarzeń $M_1, M_2, \ldots, M_k$, ponieważ nie wiemy które z tych zdarzeń zaszło, dlatego zdarzenia M_i nazywamy hipotezami. Natomiast prawdopodobieństwa $P(M_i)$ nazywamy prawdopodobieństwami *a priori*. Prawdopodobieństwo to możemy zinterpretować następująco: określa ono udział poszczególnych klas w procesie generowania obiektów i nie zawiera w sobie żadnej informacji pomiarowej o rozpoznawanym obiekcie. A więc jedynie ilościowo opisuje statystyczne właściwości poszczególnych klas.

Stosując twierdzenie Bayes'a otrzymujemy:

$$P(M_i|X_l) = \frac{P(M_i)P(X_l|M_i)}{P(X_l)} \quad (2.16)$$

Prawdopodobieństwo $P(M_i|X_l)$ nazywamy prawdopodobieństwem *a priori*. Zatem miara ta określa prawdopodobieństwo, że obiekt opisany wektorem cech X_l należy do klasy *i*-tej.

Doniosłość twierdzenia Bayes'a polega na fakcie, że prawdopodobieństwo *a posteriori* (lewa cześć równania) jest wyrażone przez wielkości często o wiele łatwiejsze do obserwacji (pomiaru). Zazwyczaj wielkości te można wyliczyć na podstawie danych treningowych.

2.5.1 Algorytm rozpoznawania za pomocą reguły Bayes'a

Chcemy zaklasyfikować nowy obiekt do jednej z predefiniowanych klas. Celem algorytmu jest minimalizacja prawdopodobieństwa błędnej klasyfikacji (ang. *misclassification*). Wiemy jedynie, że przynależności do klas obiektów opisanych wektorem cech nie są stałe, a wykazują niepewny charakter przyporządkowania. Pomimo, że charakter związku przyporządkowania jest losowy, to wiemy, że przyporządkowanie to jest określone pewną statystyczną prawidłowością opisaną przez prawdopodobieństwa.

Zakładamy na chwilę, że wektor cech jest niemierzalny (nie podlegający obserwacji). Musimy więc dokonać klasyfikacji jedynie na podstawie rozkładu prawdopodobieństwa klas. Jedynym optymalnym rozwiązaniem jest wybór klasy posiadającej największe prawdopodobieństwo:

$$P(C_i) > P(C_j), dla\ wszystkich\ j \neq i \quad (2.17)$$

Taka reguła decyzyjna za każdym razem przyporządkuje rozpoznawanemu obiektowi klasę posiadającą największe prawdopodobieństwo *a priori*.

Jeśli wektor cech opisujący obiekt jest obserwowalny, wtedy możemy skorzystać z dodatkowej informacji w celu minimalizacji przypadku błędnej

klasyfikacji. W tym celu szukamy takiego formalizmu, który pozwoli nam połączyć informację na temat rozkładu klas z informacją na temat wartości wektora cech obiektu. Wprowadźmy zatem prawdopodobieństwo, że obiekt posiada wektor cech X_l oraz należy do klasy C_i. Prawdopodobieństwo to będziemy nazywali prawdopodobieństwem łącznym (ang. *joint probability*) i oznaczali symbolem $P(X_l,C_i)$. Intuicyjnie prawdopodobieństwo łączne to prawdopodobieństwo wylosowania klasy C_i i prawdopodobieństwo wygenerowania X_l wiedząc, że w procesie generowania X_l bierze udział klasa C_i. Powyższą interpretacje prawdopodobieństwa łącznego możemy zapisać następująco:

$$P(X_l, C_i) = P(X_l|C_i)P(C_i) \tag{2.18}$$

Zauważmy, że równanie (2.18) jest licznikiem prawej strony równania Bayes'a, natomiast mianownik w równaniu (2.16) pełni rolę normalizacyjną gwarantując, że wszystkie prawdopodobieństwa *a posteriori* sumują się do 1.

Tym samym Bayes'owski algorytm rozpoznawania sprowadza się do wyboru klasy C_i dla której rozpoznawany obiekt opisany wektorem X_l posiada największe prawdopodobieństwo *a posteriori*.

$$\rho(X_l) = C_i \Leftrightarrow P(C_i|X_l) = max_{C_j \in C}\, P(C_j|X_l) \tag{2.19}$$

Znając prawdopodobieństwo *a priori* oraz prawdopodobieństwo warunkowe $P(X_l|C_i)$ możemy zminimalizować prawdopodobieństwo błędnej klasyfikacji wybierając dla obiektu klasę z najwyższym prawdopodobieństwem *a posteriori*. Zauważmy, że algorytm ten abstrahuje od istoty konkretnego problemu rozpoznawania. Jedynym założeniem jest znajomość pełnej informacji probabilistycznej, a więc znajomość rozkładów prawdopodobieństw występujących w równaniu (2.18).

2.6 Budowa modelu

Niestety zazwyczaj podczas budowy systemu rozpoznawania nie znamy rozkładów prawdopodobieństw klas oraz prawdopodobieństwa warunkowego klas. Nasza aprioryczna wiedza o informacji probabilistycznej rozpoznawanych obiektów nie jest pełna. Wobec nieznajomości rozkładów prawdopodobieństw zmiennych losowych występujących w równaniu (2.16), zastanówmy się czy istnieje metoda pozwalająca nam oszacować wspomniane rozkłady i czy błąd oszacowania możemy zrekompensować za pomocą innej wiedzy *a priori* np. o charakterze związków pomiędzy klasami i cechami obiektów.

Źródłem informacji o rozkładach prawdopodobieństw cech i klas może być zbiór obiektów będący próbą losową populacji składającej się z pary (x, m). A więc budowa modelu polega na kolekcjonowaniu danych liczbowych opisujących pewne związki pomiędzy cechami obiektów i klas do których należą, oraz wnioskowanie na ich podstawie – polegające na estymacji parametrów rozkładów lub estymacji kształtu rozkładu. Zbiór taki, który pozwala nam odkryć

prawidłowości statystyczne pomiędzy cechami i klasami w zadaniu rozpoznawania nazywamy zbiorem uczącym (ang. *training set, learning set*). Każdy element zbioru treningowego składa się z wektora cech opisującego obiekt oraz numeru klasy do której należy obiekt:

$$T = \{(x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)\} \tag{2.20}$$

Pod pojęciem nauki rozumieć będziemy budowę modelu probabilistycznego na podstawie zbioru uczącego będącego źródłem informacji statystycznej. Proces budowy modelu polega na estymacji rozkładu prawdopodobieństwa $P(C_i)$ oraz gęstości prawdopodobieństwa $P(X_l|C_i)$. Regułę decyzyjną korzystającą z informacji probabilistycznej zawartej w zbiorze uczącym T oznaczać będziemy $\rho_T(x)$.

2.6.1 Model parametryczny

Model parametryczny jest zbiorem rozkładów prawdopodobieństw takich, że każdy element zbioru jest opisany przez wektor parametrów θ o skończonej liczbie elementów. Przestrzeń możliwych wartości wektora parametrów oznaczamy $\Theta \subseteq R^k$, gdzie k jest wymiarem wektora parametrów. Innymi słowy budując model parametryczny posiadamy pewną wiedzę *a priori* dotyczącą kształtu rozkładów prawdopodobieństw modelu. Informacja ta jest wynikiem naszej wiedzy dziedzinowej na temat zadania rozpoznawania lub rezultatem wcześniejszych badań statystycznych.

Model taki posiada pewien istotny walor implementacyjny. Dzięki sprowadzeniu algorytmu uczenia do estymacji zadanej liczby parametrów (zazwyczaj dobieranej przez projektanta) redukujemy wymaganą liczebność zbioru treningowego. Tym samym znacznie redukując czas uczenia oraz koszt pozyskania odpowiedniej liczby próbek uczących.

Najszerzej stosowaną metodą estymacji parametrów rozkładu jest metoda największej wiarygodności (ang. *maximum likelihood estimation*). Dla próby losowej definiujemy funkcję wiarygodności L próby określaną jako łączne prawdopodobieństwo (gęstość prawdopodobieństwa) wszystkich elementów próby losowej. Tak zdefiniowana wiarygodność jest funkcją elementów próby losowej oraz szacowanego parametru θ:

$$L(X, \theta) = \prod_{i=1}^{n} p(x_i, \theta) \tag{2.21}$$

gdzie p jest rozkładem prawdopodobieństwa dla zmiennych dyskretnych lub gęstością prawdopodobieństwa dla zmiennych ciągłych.

Poszukiwanie estymatora wektora parametrów θ polega na znalezieniu takiej wartości parametru θ^E:

$$L(X, \theta^E) = max_{\theta \in \Theta} L(X, \theta) \tag{2.22}$$

Zatem budowa modelu sprowadza się do zadania optymalizacji funkcji L względem parametru θ.

2.6.2 Model nieparametryczny

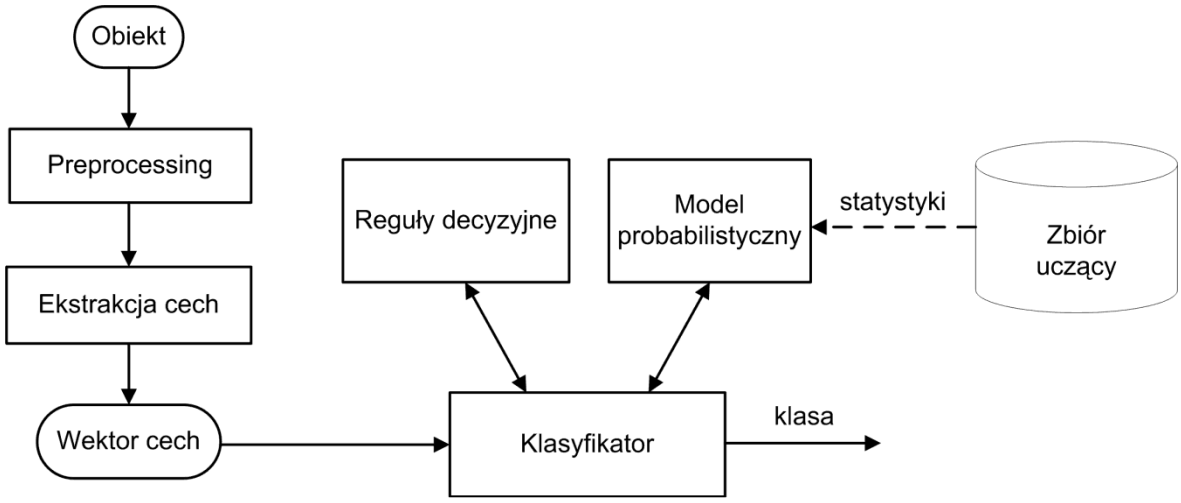
O wiele ciekawsza, z algorytmicznego punktu widzenia, jest sytuacja, gdy nie posiadamy wiedzy *a priori* na temat postaci funkcyjnej gęstości prawdopodobieństw warunkowych cech w klasach. W przypadku nieparametrycznym zakładamy jedynie, że zbiór treningowy jest rezultatem procesu generującego obiekty o rozkładzie prawdopodobieństwa, który chcemy estymować podczas budowy modelu. A zatem zadanie budowy modelu nieparametrycznego sprowadza się do zadania estymacji nieparametrycznej gęstości prawdopodobieństw warunkowych z próbki losowej, którą w naszym przypadku jest zbiór treningowy. Mówiąc obrazowo w przeciwieństwie do modelu parametrycznego struktura modelu nieparametrycznego nie jest zadana z góry, ale kształtowana na podstawie danych treningowych.

Formalizując, ponownie dla próby losowej określamy funkcję wiarygodności *L* rozumianą jako łączne prawdopodobieństwo wszystkich elementów próby. Jednak w tym przypadku funkcja ta jest zależna jedynie od próby losowej.

$$L(X) = \prod_{i=1}^{n} p(x_i) \tag{1.23}$$

Następnie poszukujemy takiego estymatora rozkładu prawdopodobieństwa *p* (lub funkcji gęstości prawdopodobieństwa) który maksymalizuje funkcję (1.23).

2.7 System rozpoznawania obiektów



Rys. 2.1 Schemat systemu rozpoznawania obiektów

Schematycznie architekturę systemu rozpoznawania obiektów przedstawia rysunek 2.1. Na podstawie zbioru uczącego pozyskujemy pewne statystyczne prawidłowości związków pomiędzy wektorem cech rozpoznawanych obiektów, a klasą do której należy obiekt. Ponieważ związki te nie są deterministyczne, dlatego budujemy model probabilistyczny. Opieramy się tutaj na założeniu, że wielkości wektora cech obiektów nie są stałe dla tej samej klasy, lecz wykazują

one pewną statystyczną prawidłowość scharakteryzowaną rozkładami prawdopodobieństwa lub gęstościami prawdopodobieństwa.

Budując model pamiętamy o tym, aby modelował on proces generowania danych dla poszczególnych klas, a nie dopasowywał się dokładnie do zbioru uczącego (ang. *over-fitting*). Celem zadania rozpoznawania obiektów jest klasyfikacja nowych wcześniej nieanalizowanych obiektów. Zatem poprawnie zbudowany model powinien charakteryzować się wysokim stopniem generalizacji. Warto dodać, że model nie powinien być zbyt złożony. To znaczy nie powinien zawierać zbyt dużej liczby parametrów. Zbyt złożony model podczas nauki zapamiętuje często szumy towarzyszące procesowi generowania danych, a więc charakteryzuje się słabą reprezentacją prawdziwego rozkładu prawdopodobieństwa.

Każdy rozpoznawany obiekt pojawiający się na wejściu recognizer'a jest wstępnie przetwarzany w celu normalizacji atrybutów opisujących obiekt. Następnie wektor atrybutów podlega transformacji, zwanej ekstrakcją cech. Ekstrakcja cech ma na celu pozbycie się informacji nadmiarowej, zbytecznej podczas klasyfikacji oraz redukcję liczby wymiarów wektora opisującego rozpoznawany obiekt. Algorytm ekstrakcji cech często stanowi jedyną wiedzą *a priori* na temat natury konkretnego problemu rozpoznawania, dlatego powinien być dobrany starannie przez eksperta dziedzinowego.

Wektor cech następnie trafia do klasyfikatora, który przy pomocy reguł decyzyjnych i wielkości statystycznych modelu probabilistycznego przyporządkowuje wektorowi odpowiedni numer klasy.

2.8 Definicja problemu rozpoznawania mowy

Mowa jest podstawowym środkiem komunikacji stosowanym przez ludzi. Problem automatycznego rozpoznawania mowy (ang. *automatic speech recognition*) intrygował naukowców od dziesięcioleci. Pierwsze kroki w kierunku matematycznego opisu sygnału mowy zostały postawione w latach trzydziestych w laboratorium Bell'a. Jednakże podwaliny pod współczesne systemy rozpoznawania mowy zostały położone w latach siedemdziesiątych przez Baker'a i Jelinek'a pracowników IBM. Badacze ci sformułowali problem rozpoznawania mowy stosując aparat pojęciowy wykorzystywany w metodach statystycznego rozpoznawania obiektów. Poczynając od lat osiemdziesiątych obserwujemy coraz większy udział systemów ASR w komunikacji człowieka z maszyną.

W rozdziale tym postaram się w sposób formalny opisać problem automatycznego rozpoznawania mowy. Natomiast rozdziały następne zawierają szczegóły techniczne systemu ASR zrealizowanego w ramach pracy magisterskiej.

Zadanie automatycznego rozpoznawania mowy polega na przyporządkowaniu fali akustycznej sygnału mowy transkrypcji wypowiedzianych słów. Ponieważ sygnał mowy jest zazwyczaj zaszumiony oraz struktura akustyczna wypowiedzi nie jest stała, proste porównywanie sygnału wejściowego z zapamiętanymi wzorcami nie daje zadowalających rezultatów. Dane wejściowe charakteryzują się dużą zmiennością oraz zawierają wiele informacji niepewnej. Tym samym jedyną metryką reprezentującą przypadkowy rozrzut cech sygnału mowy jest rozkład prawdopodobieństwa dla przypadku dyskretnego (lub gęstość prawdopodobieństwa dla przypadku ciągłego). A więc zadanie rozpoznawania mowy polega na oszacowaniu nieznanych rozkładów sygnału akustycznego mowy. Stosując metody statystyczne budujemy model probabilistyczny procesu powstawania mowy i tworzymy przestrzeń poszukiwań, którą przeszukujemy podczas dekodowania w celu odnalezienia sentencji najbardziej prawdopodobnej. Zatem problem automatycznego rozpoznawania mowy jest szczególnym przypadkiem Bayes'owskiego algorytmu rozpoznawania opisanego w punkcie 2.5.

Problem rozpoznawania mowy można w skrócie streścić do:

Jaki jest najbardziej prawdopodobny ciąg słów W z danego języka L (przestrzeni słów) dla wektora cech sygnału akustycznego O, który chcemy rozpoznać?

Przy czym ciąg W traktujemy jako ciąg słów w postaci tekstowej:

$$W = w_1, w_2, \dots, w_n \tag{2.24}$$

Natomiast dane wejściowe *O* są kolejnymi wektorami cech wyekstrahowanymi z fali akustycznej sygnału mowy. Zazwyczaj elementy tego ciągu reprezentują ramki 10 milisekundowe złożone ze składowych reprezentujących energię w poszczególnych kanałach analizowanego pasma.

$$O = o_1, o_2, \dots, o_t \tag{2.25}$$

Zdefiniujmy proces pozyskania ciągu *O*. Zakładamy, że ciąg *O* jest zdarzeniem losowym obserwowanym w doświadczeniu polegającym na wypowiadaniu słów. Zdarzenie *O* zachodzi tylko wtedy, gdy zachodzi jedno z wykluczających się zdarzeń $W_1, W_2, \dots, W_n$ – polegających na wylosowaniu wymawianego słowa. Skoro proces automatycznego rozpoznawania mowy polega na wyborze słowa W_i takiego że:

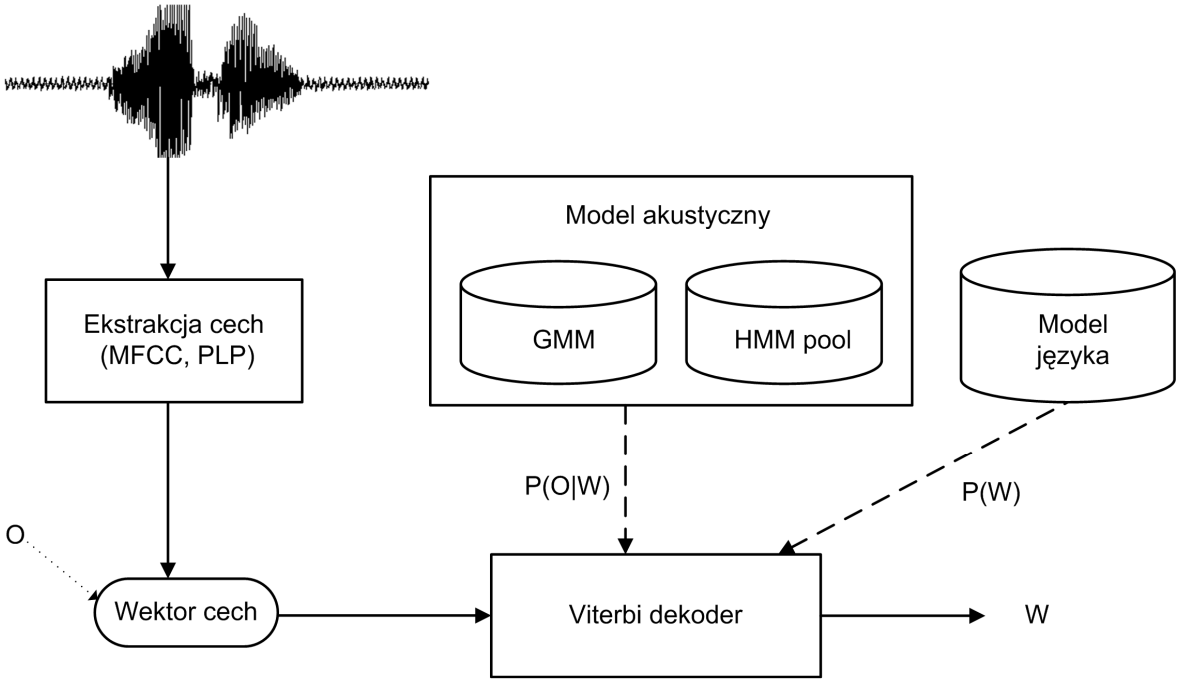
$$P(W_i|O) = max_{W \in L}\, P(W|O) \tag{2.26}$$

Stosując twierdzenie Bayes'a, otrzymujemy formalną definicję problemu automatycznego rozpoznawania mowy:

$$W_i = argmax_{W \in L}\, P(W|O) = argmax_{W \in L} \frac{P(O|W)P(W)}{P(O)} \tag{2.27}$$

Zatem dekodowanie odbywa się na podstawie kryterium MAP (ang. *maximum a posteriori*). Prawdopodobieństwo *a posteriori* obliczamy w prosty sposób na podstawie równania (2.27). Prawdopodobieństwo *a priori P(W)*

obliczamy na podstawie modelu języka, natomiast prawdopodobieństwo *P(O|W)* szacujemy z modelu akustycznego. Najtrudniejsze w oszacowaniu jest prawdopodobieństwo *P(O)*. Ponieważ jest ono takie same dla każdego słowa z przestrzeni poszukiwań możemy je pominąć podczas procesu rozpoznawania.



Rys. 2.2 Schemat architektury systemu ASR

Na rysunku 2.2 przedstawiono uproszczony diagram architektury zaimplementowanego systemu rozpoznawania mowy. W pierwszym etapie sygnał mowy jest przekształcany za pomocą algorytmu ekstrakcji cech akustycznych (np. MFCC, PLP). Skonstruowany wektor następnie trafia do dekodera, który implementuje algorytm Vitetrbi'ego. Podczas przeszukiwania przestrzeni słów wykorzystujemy model akustyczny do estymacji gęstości prawdopodobieństwa cech w klasie *P(O|W)* oraz model języka do estymacji prawdopodobieństwa *a priori* słowa *W*. Na podstawie wymienionych prawdopodobieństw szacujemy prawdopodobieństwo *a posteriori* analizowanych słów. W ostatniej fazie algorytmu wybieramy najbardziej prawdopodobną hipotezę – a więc słowo z najwyższym prawdopodobieństwem *a posteriori*.

Najczęściej oszacowanie *P(O|W)* w systemach ASR odbywa się za pomocą estymacji parametrycznej. Przyjmujemy, że rozkład cech akustycznych jest opisany wielowymiarowym rozkładem Gaussa składającym się z kilku komponentów (ang. *Gaussian mixture model*). Na podstawie zbioru treningowego szacujemy wartości średnich i macierzy kowariancji rozkładów. Z kolei wymiar czasu sygnału mowy modelujemy za pomocą ukrytych modeli Markowa HMM.

3 Ukryte modele Markowa

Generalizując można powiedzieć, że zadanie rozpoznawania mowy polega na wybraniu dla każdego wypowiedzianego wyrażenia najlepiej pasującego wzorca z przestrzeni poszukiwań. Jednak nawet dla tego samego człowieka struktura akustyczna tej samej wypowiedzi, a w szczególności struktura czasowa, może się zmieniać. Porównanie wypowiedzi ze wzorcem staje się skomplikowanym zadaniem. Po pierwsze, czas trwania poszczególnych fonemów (elementarnych części mowy) różni się u poszczególnych osób. Jednym z rozwiązań tego problemu jest normalizacja wzorca i rozpoznawanej wypowiedzi. Jednakże pojawia się drugi problem, tempo wymowy poszczególnych fonemów nie jest stałe w danym słowie. Tym samym algorytm optymalnego wyrównania fonemów wzorca i wypowiedzi nie jest liniowy, co komplikuje problem skutecznego rozpoznawania mowy. Problem optymalnego wyrównania fonemów znany jest w literaturze pod pojęciem *dynamic time warping* [4][5]. Jednakże podejście to nie znalazło szerszego zastosowania w nowoczesnych systemach automatycznego rozpoznawania mowy.

Znacznie częściej stosowanym rozwiązaniem są ukryte modele Markowa. Przyczyną tego faktu jest szkielet statystyczny zastosowany w ukrytych modelach Markowa, pozwalający łatwo modelować proces powstawania mowy. Ukryte modele Markowa poza tym pozwalają łatwo konfigurować system ASR (np. liczba stanów, typ połączeń, architektura itp.), tym samym umożliwiając dostosowanie konfiguracji recognizer'a to konkretnego problemu oraz konkretnego zbioru treningowego. Dodatkowo HMM-y posiadają łatwy w implementacji algorytm nauki estymujący parametry modelu oraz wydajny algorytm dekodowania sygnału mowy.

W dalszej części rozdziału przedstawiam szczegółowo teoretyczne aspekty ukrytych modeli Markowa. Materiał tego rozdziału abstrahuje od problemu rozpoznawania mowy. W rozdziale czwartym natomiast omawiam zastosowanie HMM w systemach ASR.

3.1 Proces stochastyczny

Procesem stochastycznym nazywamy rodzinę zmiennych losowych Z_t przyjmujących wartości z przestrzeni *X*, indeksowanych za pomocą wartości ze zbioru *T*. Zmienne losowe Z_t są określone na przestrzeni probabilistycznej (Ω, F, P). Zatem proces stochastyczny *Z* jest zbiorem:

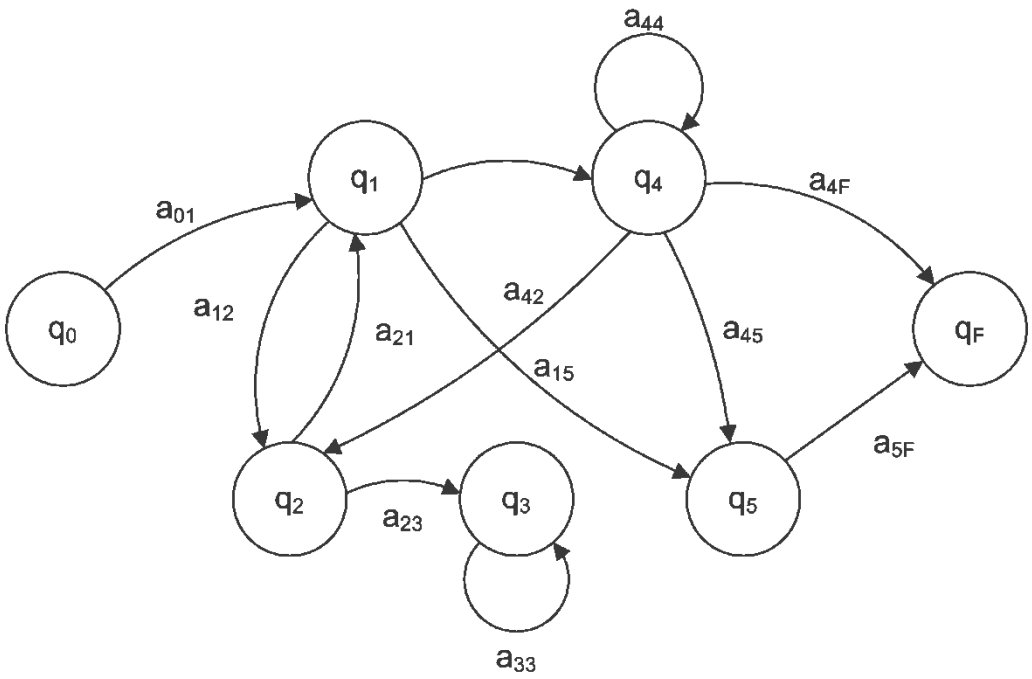
$$\{Z_t : t \in T\} \tag{3.1}$$

gdzie Z_t są zmiennymi losowymi przyjmującymi wartości z przestrzeni *X*.

Procesy stochastyczne, których zbiór indeksów jest przeliczalny nazywamy łańcuchami.

3.2 Proces Markowa

Proces Markowa to proces stochastyczny, w którym warunkowe rozkłady prawdopodobieństwa następnych stanów są zdeterminowane jedynie przez bieżący stan procesu i są niezależne od stanów przeszłych. Innymi słowy informacja potrzebna do ewolucji procesu jest zawarta jedynie w obecnym stanie, natomiast przejścia pomiędzy kolejnymi stanami są opisane prawdopodobieństwami.



Rys. 3.1 Łańcuch Markowa

Na rysunku 3.1 przedstawiono schemat graficzny przykładowego łańcucha Markowa. Wierzchołki reprezentują możliwe stany łańcucha Markowa, natomiast

przejścia pomiędzy stanami zamodelowane są za pomocą łuków między wierzchołkami. Łańcuch Markowa jest opisany następującymi komponentami:

$Q = \{q_1, q_2, \dots, q_N\}$	Zbiór stanów realizujących łańcuch Markowa
$A = \begin{bmatrix} a_{00} & a_{10} & \dots & a_{N0} & a_{F0} \\ a_{01} & a_{11} & & a_{N1} & a_{F1} \\ \vdots & & \ddots & \vdots & \\ a_{0N} & a_{1N} & & a_{NN} & a_{FN} \\ a_{0F} & a_{1F} & \dots & a_{NF} & a_{FF} \end{bmatrix}$	Macierz prawdopodobieństw przejść pomiędzy stanami. a_{ij} reprezentuje prawdopodobieństwo przejścia ze stanu *i* do stanu *j*. $a_{ij} = P(q_t = j \mid q_{t-1} = i)$
q_0, q_F	Sztucznie stworzone stany specjalne reprezentujące odpowiednio stan początkowy i stan końcowy.

Możemy teraz zapisać własność Markowa bardziej formalnie:

$$P(q_t|q_1 q_2 \dots q_{t-1}) = P(q_t|q_{t-1}) \tag{3.2}$$

Prawdopodobieństwo, że łańcuch Markowa znajduje się w stanie *q* w chwili *t* zależy jedynie od stanu procesu w chwili *t*-1. Ponieważ wartości a_{ij} reprezentują prawdopodobieństwa to suma wychodzących krawędzi powinna sumować się do 1.

$$\sum_{j=1}^{N} a_{ij} = 1 \,, \forall_i \tag{3.3}$$

3.3 Ukryty model Markowa

W wielu przypadkach sekwencja kolejnych stanów realizujących łańcuch Markowa jest nieobserwowalna. Podczas rozpoznawania mowy procesy myślowe odpowiedzialne za generowanie sygnału mowy są niewidoczne. Możemy jedynie zaobserwować zdarzenia związane ze zmiana fali akustycznej, natomiast słowa będące źródłem tych zmian są ukryte. Zatem realizacja stanów tworzących proces generowania mowy jest zmienną ukrytą. Jednak pomimo tego, że nie jest bezpośrednio obserwowalna wpływa w pewien sposób na zdarzenia podlegające obserwacji. A więc musimy na podstawie zmiennych obserwowanych (ang. *observed events*) wywnioskować stan zmiennych ukrytych (ang. *hidden variable*). Warto zaznaczyć, że wiele zjawisk występujących w przyrodzie posiada właśnie wyżej przedstawione właściwości. W tym celu potrzebujemy zupełnie nowy aparat matematyczny modelujący przestawiony proces stochastyczny.

W zastosowaniach inżynierskich zjawiska takie modelujemy za pomocą ukrytych modeli Markowa (ang. *hidden Markov model, HMM*). Po raz pierwszy szersze zastosowanie ukrytych modeli Markowa nastąpiło w laboratoriach IBM. Szybko jednak ukryte modele Markowa znalazły zastosowanie w innych

dyscyplinach naukowych. Od początku lat 80 HMM zdominowały wręcz praktyczne implementacje zadania rozpoznawania mowy.

Ukryty model Markowa jest parą procesów stochastycznych: ukrytego łańcucha Markowa i obserwowalnego procesu opisanego rozkładem prawdopodobieństw (lub gęstościami prawdopodobieństw) dla każdego stanu ukrytego. Innymi słowy: obserwowalne zdarzenia (np. cechy akustyczne sygnału mowy) są modelowane za pomocą rozkładu prawdopodobieństwa emisji osobno dla każdego ukrytego stanu, natomiast stany ukryte realizują łańcuch Markowa rzędu pierwszego.

Formalna definicji ukrytego modelu Markowa wygląda następująco:

$Q = (q_1, q_2, \dots, q_N)$ — Ciąg stanów ukrytych realizujących łańcuch Markowa – proces ten jest nieobserwowalny

$$A = \begin{bmatrix} a_{00} & a_{10} & \cdots & a_{N0} & a_{F0} \\ a_{01} & a_{11} & & a_{N1} & a_{F1} \\ \vdots & & \ddots & \vdots & \\ a_{0N} & a_{1N} & & a_{NN} & a_{FN} \\ a_{0F} & a_{1F} & \cdots & a_{NF} & a_{FF} \end{bmatrix}$$

Macierz prawdopodobieństw przejść pomiędzy stanami ukrytymi. a_{ij} reprezentuje prawdopodobieństwo przejścia ze stanu *i* do stanu *j*. $a_{ij} = P(q_t = j | q_{t-1} = i)$

$$\sum_{j=1}^{N} a_{ij} = 1 \,, \underset{i}{\forall}$$

$O = (o_1, o_2, \dots, o_T)$ — Ciąg zdarzeń obserwowalnych podczas procesu.

$B = \{b_i(o_t) : i = 1..N, t = 1..T\}$ — Prawdopodobieństwa emisji obserwacji o_t przez stan *i*. Dla przypadku dyskretnego prawdopodobieństwa emisji opisane są rozkładem prawdopodobieństwa, dla przypadku ciągłego funkcją gęstości prawdopodobieństwa

q_0, q_F — Sztucznie stworzone stany specjalne reprezentujące odpowiednio stan początkowy i stan końcowy.

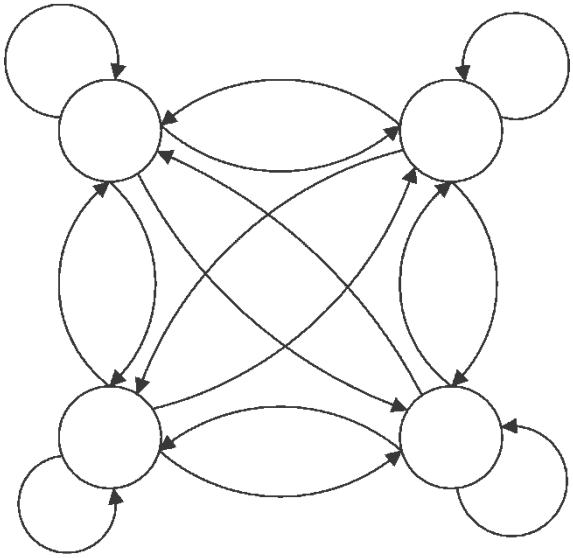
Ponownie zakładamy, że prawdopodobieństwo obecnego stanu zależy jedynie od stanu poprzedniego, zatem równość (3.2) jest spełniona. Dodatkowo prawdopodobieństwo emisji symbolu o_t zależy jedynie od obecnego stanu q_t, co możemy formalnie zdefiniować za pomocą:

$$P(o_t | q_1 \dots q_T, o_1 \dots o_T) = P(o_t | q_t) \tag{3.4}$$

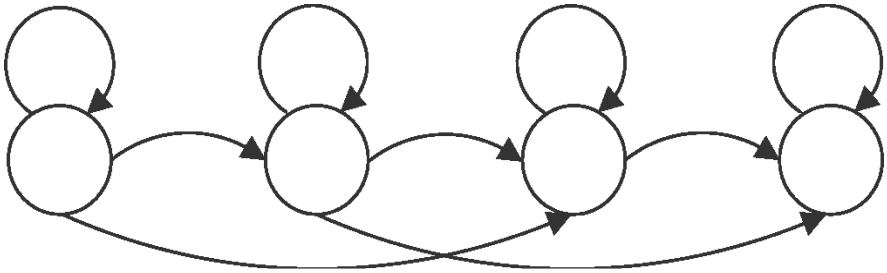
Dodatkowo ukryty model Markowa opisany macierzą przejść *A* między stanami oraz prawdopodobieństwami emisji *B* oznaczać będziemy symbolem λ

$$\lambda = (A, B) \tag{3.5}$$

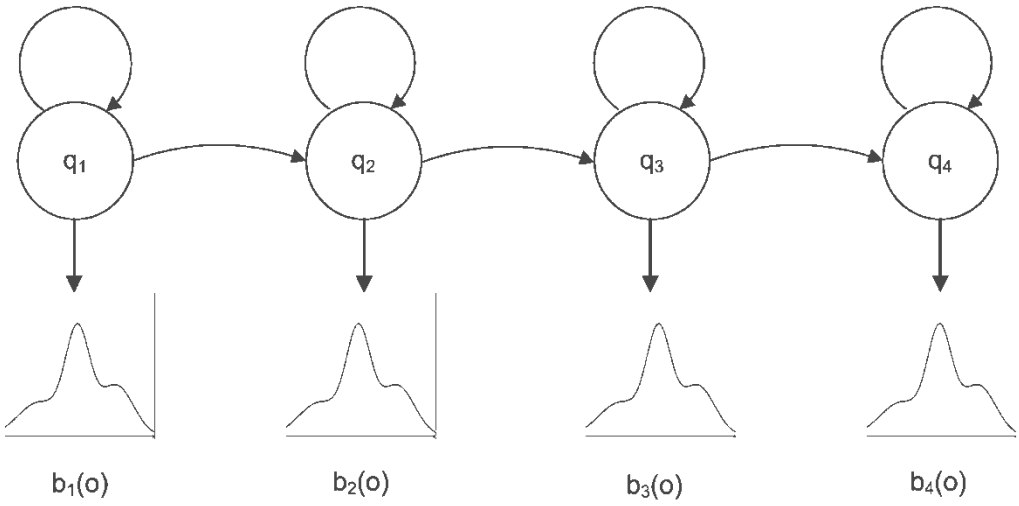
Kolejne diagramy przedstawiają najbardziej typowe schematy połączeń pomiędzy stanami ukrytymi HMM wykorzystywanymi w zastosowaniach inżynierskich.



Rys. 3.2 Ergodic HMM – reprezentuje pełne połączenia pomiędzy stanami



Rys. 3.3 Bakis HMM



Rys. 3.4 – Schemat liniowy - najczęsciej stosowany schemat HMM w systemach ASR. Na diagramie zaznaczono funkcje gęstości prawdopodobieństwa emisji obserwacji o

Ukryty model Markowa opisany za pomocą Q, A, B może posłużyć nam jako generator ciągu obserwacji:

$$O = (o_1, o_2, \dots, o_T)$$

Algorytm generatora zrealizowany jest za pomocą następujących kroków:

1) Ustaw stan początkowy na q_0. Stan ten nie emituje obserwacji.

2) Ustaw t=1

3) Na podstawie wewnętrznych połączeń pomiędzy stanami oraz rozkładowi prawdopodobieństwa przejść *A* wybierz kolejny stan $q_t=i$

3) Dla nowego stanu zgodnie z funkcją gęstości prawdopodobieństwa emisji $b_i(o)$ wybierz symbol zdarzenia obserwowalnego $o_t=k$

4) Ustaw $t=t+1$

5) Jeśli $t<T$ to idź do kroku 3); w przeciwnym wypadku zakończ proces generacji

3.3.1 Trzy podstawowe problemy HMM

Rabiner w pracy [27] zdefiniował trzy fundamentalne problemy algorytmiczne związane z HMM, które należy rozwiązać, aby HMM znalazły zastosowanie w pracy inżynierskiej:

1) Problem obliczania prawdopodobieństwa:

Dany jest ciąg obserwacji $O = (o_1, o_2, \dots, o_T)$ *oraz model HMM* $\lambda = (A, B)$ *w jaki sposób można wydajnie obliczyć prawdopodobieństwo P(O|λ) wygenerowania ciągu O przez model λ?*

2) Problem dekodowania:

Dany jest ciąg obserwacji $O = (o_1, o_2, \dots, o_T)$ *oraz model HMM* $\lambda = (A, B)$ *w jaki sposób znaleźć ciąg stanów ukrytych* $Q = (q_1 q_2 \dots q_T)$*, który najlepiej modeluje zaobserwowany ciąg O?*

3) Problem nauki:

Dany jest ciąg obserwacji $O = (o_1, o_2, \dots, o_T)$ *oraz model HMM* $\lambda = (A, B)$ *w jaki sposób dobrać parametry A i B aby zmaksymalizować prawdopodobieństwo P(O|λ)?*

W kolejnych punktach rozdziału przedstawiam szczegółowe rozwiązania zarysowanych problemów.

3.4 Algorytm Forward

Pierwszym problemem, który musimy rozwiązać jest obliczenie prawdopodobieństwa wygenerowania zadanego ciągu obserwacji przez ukryty model Markowa. Nie znamy ciągu stanów ukrytych realizujących proces, znamy jedynie parametry *A* i *B* modelu HMM. Możemy problem ten również ująć trochę bardziej obrazowo: przypisz ocenę jak bardzo model HMM pasuje do ciągu obserwacji. Takie zdefiniowanie problemu jest niesamowicie użyteczne, gdy stoimy przed problemem wyboru modelu który najlepiej pasuje do zaobserwowanego ciągu zdarzeń *O*.

Zdefiniujmy formalnie zadanie obliczania prawdopodobieństwa *P(O|λ):*

Dane są HMM $\lambda = (A, B)$ *i ciąg obserwacji* $O = (o_1, o_2, \dots, o_T)$ *oblicz prawdopodobieństwo P(O|λ).*

Pojedynczy stan ukryty HMM generuje tylko jeden symbol obserwacji, zatem liczba stanów ukrytych jest równa długości ciągu O. Toteż najprostsze rozwiązanie wyżej sformułowanego problemu to sporządzenie wszystkich *T*-elementowych wariacji z powtórzeniami ze zbioru wszystkich możliwych stanów ukrytych modelu HMM (moc tego zbioru to *N*). Każdy element wariacji posiada przypisany jeden symbol obserwacji (o tym samym indeksie). Następnie korzystając z macierzy *A* i *B* obliczamy prawdopodobieństwo wygenerowania przez tę wariację stanów zadanego ciągu *O*.

Sformułujmy zatem rozwiązanie bardziej formalnie. Każdą wariację ze zbioru stanów ukrytych HMM będziemy oznaczali:

$$Q_l^T = (q_1 q_2 \dots q_T) \tag{3.6}$$

Indeks *T* oznacza długość ciągu równą długości ciągu obserwacji, indeks *l* oznacza numer wariacji.

Prawdopodobieństwo emisji ciągu *O* przez ciąg stanów Q_l^T wynosi:

$$P(O|Q_l^T, \lambda) = \prod_{t=1}^{T} P(o_t|q_t, \lambda) \tag{3.7}$$

We wzorze tym zakładamy statystyczną niezależność kolejnych elementów ciągu obserwacji, dlatego wstawiając do równania rozkłady *B* otrzymujemy:

$$P(O|Q_l^T, \lambda) = \prod_{t=1}^{T} b_{q_t}(o_t) \tag{3.8}$$

Następnie obliczamy prawdopodobieństwo że sekwencja Q_l^T jest realizacją stanów ukrytych modelu HMM:

$$P(Q_l^T|\lambda) = a_{0q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T} \tag{3.9}$$

Prawdopodobieństwo zajścia zdarzenia *O* i Q_l^T jest iloczynem prawdopodobieństw opisanych wzorami (3.8) i (3.9):

$$P(O, Q_l^T|\lambda) = P(O|Q_l^T, \lambda) P(Q_l^T|\lambda) \tag{3.10}$$

Prawdopodobieństwo, że którakolwiek realizacja stanów ukrytych HMM wygeneruje ciąg obserwacji jest sumą prawdopodobieństw (3.10) dla wszystkich wariacji Q_l^T:

$$P(O|\lambda) = \sum_{\forall Q_l^T} P(O|Q_l^T, \lambda) P(Q_l^T|\lambda) \tag{3.11}$$

Zastanówmy się jaka jest złożoność takiego algorytmu. Aby wyliczyć równanie (3.8) musimy wykonać *T* mnożeń, tę samą złożoność otrzymujemy dla równania (3.9). Następnie obliczenia musimy wykonać dla wszystkich możliwych realizacji stanów ukrytych modelu HMM. Dla schematu *ergodic* liczba ta jest określone przez liczbę wszystkich *T*-wyrazowych wariacji z powtórzeniami zbioru *N*-elementowego, a więc wynosi N^T . Zatem złożoność algorytmu wynosi $O(TN^T)$. Nawet dla małych wartości *T* i *N* czas wykonania algorytmu jest nie do zaakceptowania.

Na szczęście w pracy [30] został przedstawiony wydajny algorytm obliczania prawdopodobieństwa *P(O|λ)* w czasie rzędu $O(TN^2)$. Algorytm ten nazywany jest w literaturze *forward algorithm*. Algorytm forward oblicza prawdopodobieństwo sekwencji *O* sumując prawdopodobieństwa wszystkich możliwych ścieżek dla stanów ukrytych, które mogą wygenerować ciąg O. Podczas wyznaczania prawdopodobieństwa *P(O|λ)* wyniki częściowe przechowujemy w tablicy dlatego algorytm jest zaliczany do programowania dynamicznego.

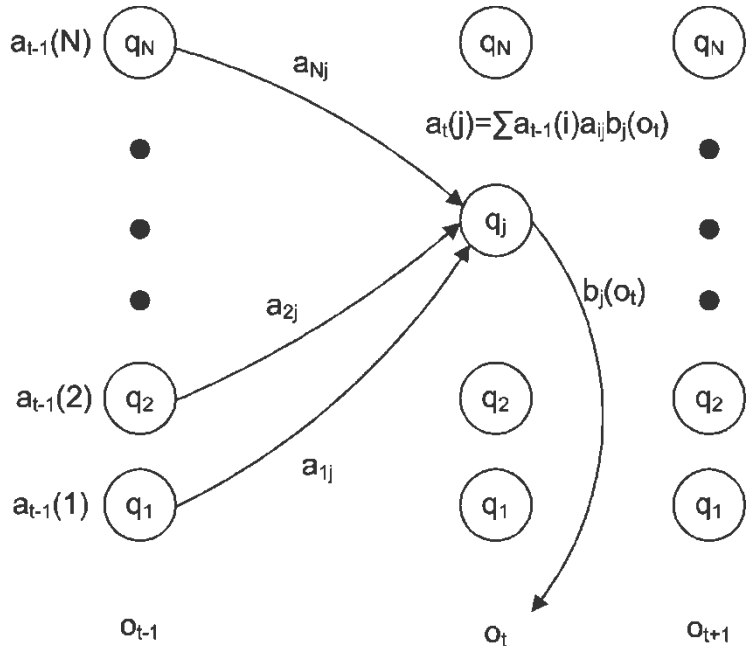
Algorytm *forward* tablicuje wyniki częściowe $\alpha_t(j)$ w tablicy dwuwymiarowej zwanej *forward trellis*. Elementy tej tablicy oznaczają prawdopodobieństwo, że model HMM jest w stanie *j* w chwili *t*. Wartość $\alpha_t(j)$ jest obliczana przez zsumowanie prawdopodobieństw wszystkich ścieżek prowadzących do tej komórki. A więc wartość komórki (t,j) oznacza prawdopodobieństwo:

$$\alpha_t(j) = P(o_1 o_2 \ldots o_t, q_t = j|\lambda) \tag{3.12}$$

Prawdopodobieństwo we wzorze (3.12) możemy obliczyć sumując wszystkie prawdopodobieństwa, że automat HMM znajduje się w chwili *t*-1 w stanie *i* oraz że w chwili *t* przechodzi do stanu *j* i stan ukryty *j* emituje symbol o_t. Formalnie taki ciąg zdarzeń możemy zapisać:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{3.13}$$

gdzie $\alpha_{t-1}(i)$ oznacza prawdopodobieństwo *forward* z poprzedniej kroku (przechowywane w tablicy), które mówi nam jakie jest prawdopodobieństwo, że w chwili *t*-1 automat jest w stanie *i*; natomiast a_{ij} jest prawdopodobieństwem przejścia ze stanu *i* do stanu *j* – element macierzy *A*; $b_j(o_t)$ to prawdopodobieństwo emisji symbolu o_t przez stan *j*. Schematycznie sytuację tę przedstawia rysunek 3.5.



Rys. 3.5 – Iteracja algorytmu forward

Aby policzyć prawdopodobieństwo *P(O|λ):*

1) Inicjalizujemy kolumnę dla chwili t=1

$a_1(j) = a_{0j} b_j(o_1), dla\ j = 1..N$

2) W kolejnych iteracjach wypełniamy kolejne komórki tablicy α

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t), dla\ j = 1..N, t = 1..T$$

3) Po wykonaniu wszystkich iteracji :

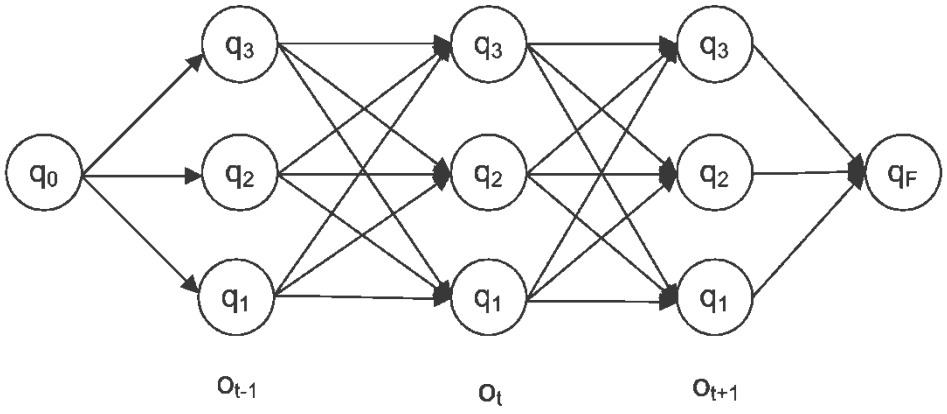
$P(O|\lambda) = \alpha_T(q_F)$

Poniżej umieszczam listing pseudokodu realizującego algorytm *forward*. Pseudokod został zaimplementowany w ramach niniejszej pracy dyplomowej.

```
Function double forwardPass(matrixA, alpha, observations,
noStates) {
  noObservations = observations.length;
  alpha = new double[noObservations][noStates + 2];
  //init
  for (int j = 1; j <= noStates; j++) {
    alpha[0][j] = matrixA[0][j]*pdfValue(j, observations[0]);
```

```
  }
  //recursion step
  for (int t = 1; t < noObservations; t++) {
    for (int j = 1; j <= noStates; j++) {
      double sum = 0;
      for (int i = 1; i <= noStates; i++) {
        double tempAlpha = alpha[t - 1][i];
        double tempA = matrixA[i][j];
        double tempB = pdfValue(j, observations[t]);
        double multi = tempAlpha * tempA * tempB;
        sum = sum + multi;
      }
      alpha[t][j] = sum;
    }
  }
  double prob = 0;
  for (int i = 1; i <= noStates; i++) {
    prob = prob + alpha[noObservations - 1][i]
                          *matrixA[i][getFinalState()];
  }
  alpha[noObservations - 1][getFinalState()] = prob;
  return prob;
}
```

Listing 3.1 – pseudokod algorytmu forward



Rys. 3.6 – wizualizacja ścieżek analizowanych podczas algorytmu Forward dla 3 stanów ukrytych i 3 symboli ciągu O

3.5 Algorytm Viterbi'ego

Dla każdego modelu zawierającego zmienne utajone zadanie polegające na wyznaczeniu najbardziej prawdopodobnej sekwencji wartości zmiennych ukrytych będących źródłem ciągu obserwacji nazywa się procesem dekodowania (ang. *decoding*). Zatem zadaniem dekodera HMM jest znalezienie takiej realizacji stanów ukrytych HMM, która z największym prawdopodobieństwem generuje zadany ciąg obserwacji *O*. Zdefiniujmy zatem zadanie dekodera bardziej formalnie:

> *Dane są HMM* $\lambda = (A, B)$ *i ciąg obserwacji* $O = (o_1, o_2, \dots, o_T)$ ***znajdź najbardziej prawdopodobny ciąg stanów*** $Q = (q_1 q_2 \dots q_T)$.

Z poprzedniego punktu wiemy, że nie możemy dla każdej wariacji stanów wyliczyć prawdopodobieństwo wygenerowania ciągu *O* przez tę wariację i wybrać ciąg stanów ukrytych z najwyższym prawdopodobieństwem. Zastanówmy się czy przez analogię do algorytmu *forward* nie możemy wyznaczyć najbardziej optymalnych ścieżek częściowych, wyniki stablicować i iteracyjnie wyznaczać kolejne ścieżki na podstawie wyników z poprzedniej iteracji.

Załóżmy, że $v_t(j)$ oznacza prawdopodobieństwo, że w chwili *t* automat jest w stanie *j* oraz ciąg stanów $(q_1 q_2 \dots q_{t-1})$ między czasem 1..t-1 jest najbardziej prawdopodobnym ciągiem stanów. Formalna definicja $v_t(j)$:

$$v_t(j) = max_{q_0, q_1, \dots, q_{t-1}} P(q_0, q_1, \dots, q_{t-1}, o_1 o_2 \dots o_{t-1}, q_t = j | \lambda) \tag{3.14}$$

Zastanówmy się w jaki sposób można wyznaczyć $v_t(j)$. Skoro $v_t(j)$ reprezentuje najbardziej prawdopodobną ścieżkę dla stanów $(q_1 q_2 \dots q_{t-1})$, a więc stanów poprzednich i prawdopodobieństwo, że w chwili obecnej automat jest w stanie *j*. Oznacza to, że ścieżka ta jest przedłużeniem najbardziej prawdopodobnej ścieżki w przedziale 1..t-1 o stan *j*. Zapiszmy tę myśl formalnie:

$$v_t(j) = max_{\forall i} v_{t-1}(i) a_{ij} b_j(o_t) \tag{3.15}$$

gdzie $v_{t-1}(i)$ to prawdopodobieństwo ścieżki z poprzedniej iteracji; a_{ij} to prawdopodobieństwo przejścia ze stanu *i* do stanu *j*; natomiast $b_j(o_t)$ to prawdopodobieństwo emisji symbolu o_t przez stan *j*. W równaniu (3.15) można zaobserwować rekurencję, gdyż iteracyjnie wyznaczamy nowe najbardziej prawdopodobne ścieżki korzystając z stablicowanych wyników iteracji poprzedniej. Algorytm zdefiniowany za pomocą rekurencji (3.15) jest zwany w literaturze algorytmem Viterbi'ego [21].

Ponieważ naszym zadaniem jest wyznaczenie najbardziej prawdopodobnej sekwencji stanów, a nie prawdopodobieństwa, podczas kolejnych iteracji algorytmu musimy zachowywać dodatkową informację o kolejnych stanach będących realizacją najbardziej prawdopodobnej ścieżki. Po zakończeniu algorytmu na podstawie tych wskaźników będziemy w stanie odtworzyć najbardziej prawdopodobną ścieżkę.

Warto tutaj również zwrócić uwagę na fakt, że algorytm Viterbi'ego posiada podobną realizację jak algorytm *forward*. Jedyna różnica to kryterium wyliczania kolejnych komórek tablicy. W algorytmie *forward* wyliczamy sumę wszystkich możliwych ścieżek z poprzedniej iteracji, natomiast w algorytmie Vitebi'ego bierzemy pod uwagę jedynie przejście najbardziej prawdopodobne.

Zatem aby wyznaczyć najbardziej prawdopodobną ścieżkę musimy wykonać następujące kroki:

1) Inicjalizujemy kolumnę tabeli v oraz tablicy wskaźników wstecz dla chwili t=1

$v_1(j) = a_{0j} b_j(o_1), dla\ j = 1..N$

$ptr_1(j) = 0, dla\ j = 1..N$

2) W kolejnych iteracjach wypełniamy kolejne komórki tablicy v

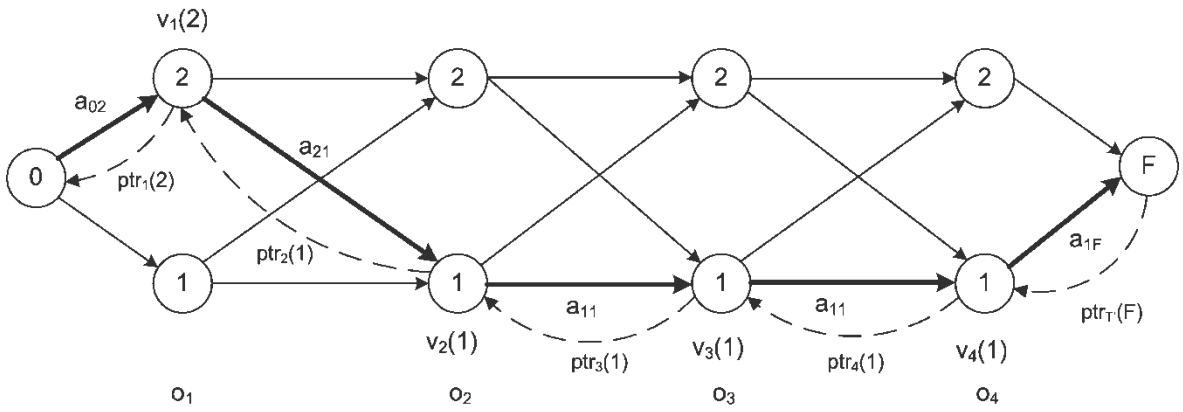
$v_t(j) = \max_{\forall i} v_{t-1}(i) a_{ij} b_j(o_t)\ , dla\ j = 1..N, t = 1..T$

$ptr_t(j) = \underset{\forall i}{argmax}\, v_{t-1}(i) a_{ij} b_j(o_t)\ , dla\ j = 1..N, t = 1..T$

3) Po wykonaniu wszystkich iteracji :

$P_{max} = v_T(q_F)$

W tabeli *ptr* znajdują się indeksy kolejnych stanów tworzących najbardziej prawdopodobną ścieżkę.



Rys. 4.7 – Na schemacie zilustrowano przebieg algorytmu Viterbi'ego dla modelu składającego się z 2 stanów ukrytych i ciągu obserwacji o długości 4. Strzałki oznaczają wszystkie możliwe ścieżki analizowane przez algorytm, pogrubione strzałki oznaczają ścieżkę Viterbi'ego. Dodatkowo liniami przerywanymi zaznaczono elementy tablicy wskaźników.

Poniżej znajduje się pseudokod algorytmu Viterbi'ego:

```
procedure viterbiSearch(matrixA, alpha, observations, noStates) {
  noObservations = observations.length;
  viterbi = new double[noObservations][noStates + 2];
  ptr = new int[noObservations][noStates + 2];
  //init
  for (int j = 1; j <= noStates; j++) {
    viterbi [0][j] = matrixA[0][j]*pdfValue(j, observations[0]);
    ptr[0][j] = 0;
  }
  //recursion step
  for (int t = 1; t < noObservations; t++) {
    for (int j = 1; j <= noStates; j++) {
      double max = 0;
      int maxIndex = 0;
      for (int i = 1; i <= noStates; i++) {
        double tempAlpha = viterbi[t - 1][i];
        double tempA = matrixA[i][j];
        double tempB = pdfValue(j, observations[t]);
        double multi = tempAlpha * tempA * tempB;
        if (multi > max) {
          max = multi;
          maxIndex = i;
        }
      }
      viterbi[t][j] = max;
      ptr[t][j] = maxIndex;
    }
  }

  double prob = 0;
  prob = max(viterbi[noObservations - 1][i]
                          * matrixA[i][getFinalState()]);
  Path path = argmax(viterbi);
}
```

Listing 3.2 – pseudokod algorytmu Viterbi'ego

3.6 Algorytm Baum'a-Welch'a

Problem nauki, czyli optymalnego doboru parametrów $\lambda = (A, B)$ modelu HMM, nie jest już tak banalny jak dwa poprzednie zadania. Ze względu na fakt, że większość zmiennych losowych występujących w HMM nie jest obserwowalna, algorytm nauki powinien estymować parametry rozkładów zmiennych ukrytych na podstawie wartości zmiennych losowych obserwowalnych zależnych od zmiennych ukrytych. Do dnia dzisiejszego nie opracowano analitycznej metody pozwalającej na dokładne wyliczenie parametrów HMM, takich które maksymalizują prawdopodobieństwo *P(O|λ)*. Jednakże istnieje szereg algorytmów iteracyjnych znajdujących maksima lokalne. Najszersze zastosowanie w systemach rozpoznawania mowy znalazł algorytm Baum'a-Welch'a [3] (często można znaleźć w literaturze alternatywną nazwę ang. *forward-backward algorithm*). Algorytm ten jest przykładem algorytmu EM [31] (ang. *expectation maximization*), którego podstawy teoretyczne przedstawiam w punkcie następnym.

Niżej przedstawiony algorytm w dużej mierze opiera się o pionierską pracę Rabiner'a [27].

Zdefiniujmy w pierwszej kolejności w sposób formalny problem nauki parametrów *A, B* :

> *Dane są ciąg obserwacji* $O = (o_1, o_2, \dots, o_T)$ *oraz zbiór możliwych stanów ukrytych modelu* $S = \{1..N\}$, *znajdź parametry A i B maksymalizujące P(O|λ).*

W dalszej części zakładam, że wartości ciągu *O* są wartościami dyskretnymi. Estymujemy zatem rozkłady prawdopodobieństwa *O*. W rozdziale czwartym omawiam wersję algorytmu Baum'a-Welcha estymującą parametry funkcji gęstości *B* – jest to przypadek, gdy wartości ciągu *O* są wartościami ciągłymi.

Zatem O_t przyjmuje wartości ze zbioru $V = \{v_1, v_2, \dots, v_m\}$.

Wprowadźmy definicję tzw. prawdopodobieństwa wstecznego (ang. *backward probability*). Termin ten formułujemy na podobnej zasadzie jak prawdopodobieństwo *α* (ang. *forward probability*) w algorytmie *forward*.

$$\beta_t(i) = P(o_{t+1}o_{t+2} \dots o_T | q_t = i, \lambda) \tag{3.16}$$

Prawdopodobieństwo wsteczne to prawdopodobieństwo warunkowe zaobserwowania sekwencji $o_{t+1}o_{t+2} \dots o_T$ (od czasu t+1 do końca) wiedząc, że automat λ jest w stanie *i* w chwili *t*. Zastanówmy się w jaki sposób możemy obliczyć prawdopodobieństwo wsteczne. Odrywamy z ciągu $o_{t+1}o_{t+2} \dots o_T$ symbol o_{t+1} i zakładamy, że został on wygenerowany przez jakikolwiek stan w chwili t+1. Skoro wiemy, że w chwili *t* automat jest w stanie *i* to prawdopodobieństwo

wygenerowania w chwili t+1 symbolu o_{t+1} wynosi $\sum_{j=1}^{N} a_{ij} b_j(o_{t+1})$. Zatem algorytm wyznaczania wartości $\beta_t(i)$ (zwany algorytmem *backward*) przebiega następująco:

1) Inicjalizujemy kolumnę dla chwili t=T

$\beta_T(i) = a_{iF}, dla\ i = 1..N$

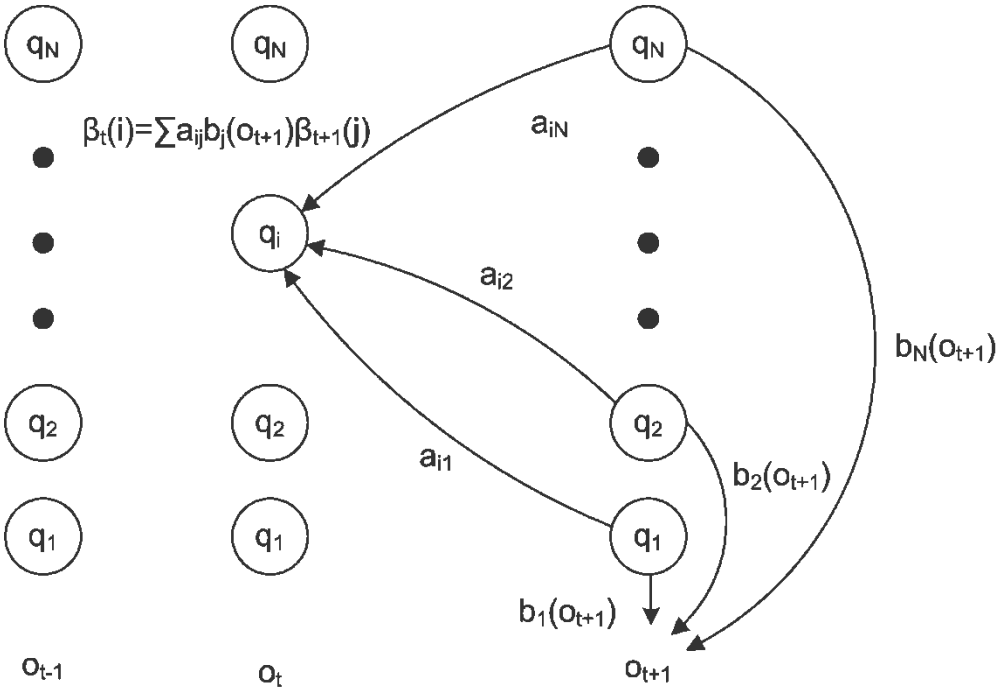
2) W kolejnych iteracjach wypełniamy kolejne komórki tablicy β

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1})\, \beta_{t+1}(j), dla\ i = 1..N, t = 1..T$$

3) Po wykonaniu wszystkich iteracji :

$P(O|\lambda) = \beta_1(0)$

Rezultatem algorytmu jest prawdopodobieństwo wygenerowania ciągu obserwacji *O* przez model HMM. Tyle że policzyliśmy to prawdopodobieństwo przemierzając kratownicę wszystkich ścieżek od końca. Schematycznie krok 2) został zilustrowany na rysunku 3.8.



Rys. 3.8 – Iteracja algorytmu backward

Poniżej znajduje się pseudokod algorytmu backward zaimplementowany w ramach pracy magisterskiej.

```
function double backwardPass(matrixA, alpha, observations,
noStates) {
  int noObservations = observations.length;
  beta = new double[noObservations][noStates + 2];
  //initialization
  for (int i = 1; i <= noStates; i++) {
    beta[noObservations - 1][i] = matrixA[i][getFinalState()];
  }

  //recursion
  for (int t = noObservations - 2; t >= 0; t--) {
    for (int i = 1; i <= noStates; i++) {
      double sum = 0;
      for (int j = 1; j <= noStates; j++) {
        double tempBeta = beta[t + 1][j];
        double tempA = matrixA[i][j];
        double tempB = pdfValue(j, observations[t + 1]);
        double multi = tempBeta*tempA*tempB;
        sum = sum+multi;
      }
      beta[t][i] = sum;
    }
  }

  //termination
  double prob = 0;

  for (int j = 1; j <= noStates; j++) {
    prob += matrixA[0][j]*pdfValue(j, observations[0])
              *beta[0][j];
  }
  beta[0][0] = prob;
  return prob;
}
```

Listing 3.3 – pseudokod algorytmu backward

Zdefiniujmy zatem estymator prawdopodobieństwa przejść pomiędzy stanami jako:

$$\bar{a}_{ij} = \frac{oczekiwana\ liczba\ przejsc\ ze\ stanu\ i\ do\ stanu\ j}{oczekiwana\ liczba\ przejsc\ ze\ stanu\ i} \tag{3.17}$$

Załóżmy, że posiadamy estymator prawdopodobieństwa $\xi_t(i,j)$ przejścia ze stanu i do stanu j w określonym momencie czasu. Wtedy wartość $\bar{a}_{ij}$ możemy obliczyć wg następującego wzoru:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)} \tag{3.18}$$

Zdefiniujmy formalnie prawdopodobieństwo $\xi_t(i,j)$:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \tag{3.19}$$

Przypomnijmy sobie definicję prawdopodobieństwa *forward* i *backward* z poprzednich podrozdziałów:

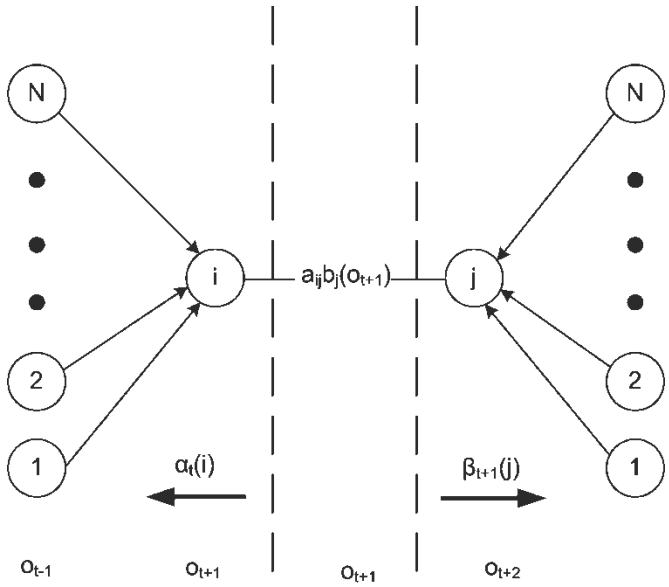
$$\alpha_t(i) = P(o_1 o_2 \dots o_t, q_t = i | \lambda) \tag{3.12}$$

$$\beta_t(j) = P(o_{t+1} o_{t+2} \dots o_T | q_t = j, \lambda) \tag{3.16}$$

Zdefiniujmy prawdopodobieństwo zajścia iloczynu zdarzeń $q_t = i, q_{t+1} = j \cap O$:

$$P(q_t = i, q_{t+1} = j, O|\lambda) = \underbrace{P(q_t = i, o_1 o_2 \dots o_t|\lambda)}_{\alpha_t(i)} \cdot \underbrace{P(q_{t+1} = j, o_{t+1} o_{t+2} \dots o_T | q_t = i, \lambda)}_{a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}, \tag{3.20}$$

Interpretację powyższego wzoru przedstawiono na rysunku 3.9.



Rys. 3.9 – Ilustracja prawdopodobieństwa łącznego

Stosując wzór na prawdopodobieństwo łączne otrzymujemy:

$$P(q_t = i, q_{t+1} = j, O|\lambda) = P(q_t = i, q_{t+1} = j|O, \lambda)P(O|\lambda) \tag{3.21}$$

Zatem wstawiając (3.12) (3.16) (3.19) (3.20) do (3.21) otrzymujemy:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{a_T(N)} \tag{3.22}$$

Wstawiając (3.22) do (3.18) otrzymujemy wzór na wartość oczekiwaną prawdopodobieństwa przejścia automatu ze stanu *i* do stanu *j*.

Zdefiniujmy estymator prawdopodobieństwa emisji symbolu v_k przez stan *j*:

$$\overline{b_j}(v_k) = \frac{oczekiwana\ liczba\ razy\ zaobserwoania\ symbolu\ v_k\ wiedząc\ ze\ stan\ ukryty\ to\ j}{oczekiwana\ liczba\ razy\ wystapienia\ stanu\ ukrytego\ j} \tag{3.23}$$

Aby obliczyć wartość $\overline{b_j}(v_k)$ wg wzoru (3.23) musimy wprowadzić prawdopodobieństwo, że automat jest w stanie *j* w chwili t wiedząc, że pojawił się ciąg obserwacji O wygenerowany przez automat λ. Formalną definicję tego prawdopodobieństwa opisuje wzór (3.24):

$$\gamma_t(j) = P(q_t = j|O, \lambda) \tag{3.24}$$

Stosując wzór na prawdopodobieństwo iloczynu dwóch zdarzeń zależnych:

$$P(A \cap B) = P(A)P(B|A) = P(B)P(A|B) \tag{3.25}$$

możemy prawdopodobieństwo $\gamma_t(j)$ określić następująco:

$$\gamma_t(j) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \tag{3.26}$$

Licznik równania (3.26) jest iloczynem $\alpha_t(j)\beta_t(j)$ natomiast mianownik jest rezultatem algorytmu *forward*. Ostateczny wzór na prawdopodobieństwo $\gamma_t(j)$ wygląda następująco:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(N)} \tag{3.27}$$

Po zdefiniowaniu prawdopodobieństwa $\gamma_t(j)$ możemy określić wartość oczekiwaną $\overline{b_j}(v_k)$ następująco: jest to stosunek prawdopodobieństwa wystąpienia stanu *j* w dowolnej chwili i zaobserwowania symbolu v_k do prawdopodobieństwa wystąpienia stanu j. Tym samym wzór estymatora prawdopodobieństwa emisji symbolu v_k przez stan *j* wygląda następująco:

$$\overline{b_j}(v_k) = \frac{\sum_{t=1, o_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{3.28}$$

Wzór (3.28) obowiązuje dla dyskretnych wartości emitowanych przez stany ukryte. W systemach rozpoznawania mowy składowe wektora cech są wartościami ciągłymi. W rozdziale czwartym przedstawiam rozszerzenie algorytmu Baum'a-Welch'a estymujące parametry funkcji gęstości prawdopodobieństwa $\overline{b_j}(v_k)$.

Ostatecznie algorytm Baum'a-Welch'a przebiega następująco:

1) Inicjalizujemy macierze *A* i *B*. Początkowe wartości macierzy *A* i *B* są zazwyczaj średnimi policzonymi ze zbioru uczącego.

2) Wykonujemy algorytm *forward* i *backward* aby oszacować wartości $\alpha_t(i)$ $\beta_t(j)$

3) Wykonujemy krok E (ang. *expectation*) polegający na oszacowaniu wartości oczekiwanych:

$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(N)}$, dla wszystkich t, j

$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)}$, dla wszystkich t, i, j

4) Wykonujemy krok M (ang. maximization) polegający na maksymalizacji kryterium MLE podczas estymowania parametrów:

$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{j=1}^{N}\xi_t(i,j)}$, dla wszystkich i, j

$\overline{b_j}(v_k) = \frac{\sum_{t=1,o_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$, dla wszystkich j, v_k

5) Jeśli kolejne wartości prawdopodobieństwa $P(O|\lambda)$ są zbieżne lub liczba iteracji przekroczyła maksimum zakończ algorytm; w przeciwnym wypadku wróć do punktu 2)

Algorytm Baum'a-Welch'a iteracyjnie estymuje parametry modelu HMM. Algorytm rozpoczynamy z ręcznie zainicjalizowanymi parametrami automatu i następnie wykorzystujemy te wartości do szacowania kolejnych co raz dokładniejszych estymatorów parametrów ukrytego modelu Markowa.

Podczas fazy E obliczamy wartości oczekiwane wystąpienia stanu ukrytego automatu $\gamma_t(j)$ oraz wartości oczekiwane liczby przejść pomiędzy stanami $\xi_t(i,j)$ na podstawie wartości *A* i *B* z iteracji poprzedniej. Natomiast podczas fazy M korzystamy z wartości oszacowanych podczas fazy E, aby wyznaczyć nowe dokładniejsze estymatory wartości *A* i *B*. Tym samym z każdą iteracją wartości parametrów *A* i *B* są co raz lepiej dopasowane do zbioru uczącego.

W punkcie następnym przedstawiam teoretyczne aspekty metody EM.

3.7 Algorytm EM

Algorytm Expectation-Maximization [31] jest metodą estymacji parametrów rozkładu populacji na podstawie kryterium największej wiarygodności próby losowej. Algorytm ten stosuje się w przypadku gdy dane są niekompletne lub gdy w modelu występują zmienne ukryte.

W rozdziale 3.6.1 opisałem szczegółowo metodę największej wiarygodności (MLE). Ponieważ algorytm EM korzysta z kryterium MLE, w celu przejrzystości opisu przypominam formalną definicję funkcji wiarygodności.

$$L(X,\theta) = \prod_{i=1}^{n} p(x_i,\theta) \tag{3.21}$$

$$\theta^E = argmax_{\theta\in\Theta}\, L(X,\theta) \tag{3.22}$$

Bardzo często analityczna optymalizacja funkcji *L* jest niemożliwa ze względu na obecność zmiennych ukrytych. Jednakże gdybyśmy założyli istnienie wartości parametrów tych zmiennych moglibyśmy iteracyjnie szacować coraz dokładniejsze estymatory tych parametrów. Problem ten często spotykamy w systemach rozpoznawania obiektów.

W tym celu zakładamy, że θ jest wektorem parametrów, *X* zmienną losową podlegającą obserwacji. Wiemy również, że na rozkład zmiennej losowej *X* ma wpływ zmienna losowa *Y* nieobserwowalna w doświadczeniu. Zmienna losowa *Y* jest zmienną ukrytą (ang. *hidden/latent variable*). Realizację próby losowej zmiennej *X* oznaczać będziemy przez $\mathcal{X}$={$x_1,\ldots,x_n$}. Przez *Z* rozumiemy parę zmiennych losowych *Z=(X,Y)*. Korzystając ze wzoru (3.25) funkcję gęstości prawdopodobieństwa zmiennej *Z* możemy zapisać w następujący sposób:

$$p(Z|\theta) = p(X,Y|\theta) = p(Y|X,\theta)p(X|\theta) \tag{3.29}$$

Ponieważ w kroku E algorytmu korzystamy z oszacowań parametru θ z poprzedniej iteracji, dlatego zakładamy, że θ jest stałe.

Zatem możemy zdefiniować funkcję wiarygodności dla próby losowej $\mathcal{Z}$ ={$\mathcal{X},\mathcal{Y}$}

$$L(\theta|\mathcal{Z}) = L(\theta|\mathcal{X},\mathcal{Y}) = p(\mathcal{X},\mathcal{Y}|\theta) \tag{3.30}$$

Funkcja ta jest zmienną losową ponieważ realizacja próbki $\mathcal{Y}$ jest nieznana, czyli funkcja wiarygodności jest funkcją zmiennej $\mathcal{Y}$, gdyż wartości $\mathcal{X}$ i θ są stałe.

W fazie E algorytm EM oblicza wartość oczekiwaną funkcji $L(\theta|\mathcal{Z})$ względem zmiennej $\mathcal{Y}$ posiadając wartości próbki losowej $\mathcal{X}$ oraz estymatory parametrów:

$$Q\left(\theta,\theta^{i-1}\right) = E\left[\log p(\mathcal{X},\mathcal{Y}|\theta)|\mathcal{X},\theta^{i-1}\right] \tag{3.31}$$

gdzie θ^{i-1} to obecny estymator parametrów, który wykorzystujemy do obliczenia wartości oczekiwanej funkcji *L*, natomiast θ jest parametrem, który optymalizujemy, aby znaleźć maksimum Q.

Pamiętając, że :

$$E[h(Y)|X = x] = \int h(y)f(y|x)dy \tag{3.32}$$

Możemy równanie wartości oczekiwanej *L* zapisać następująco:

$$E\left[logp(\mathcal{X},\mathcal{Y}|\theta)|\mathcal{X},\theta^{i-1}\right] = \int logp(\mathcal{X},y|\theta)f(y|\mathcal{X},\theta^{i-1})dy \tag{3.33}$$

Funkcja $f(y|\mathcal{X},\theta^{i-1})$ jest funkcję gęstości prawdopodobieństwa rozkładu brzegowego zmiennej ukrytej *Y*. Całkę w równaniu (3.33) liczymy po całej przestrzeni wartości zmiennej *Y*.

W fazie E algorytmu obliczamy wartość oczekiwaną funkcji *L* na podstawie próby losowej $\mathcal{X}$ oraz estymatora parametrów θ^{i-1} z poprzedniej iteracji. Otrzymana wartość jest funkcją zmiennej θ ($Q(\theta, \theta^{i-1})$).

W fazie M algorytmu EM maksymalizujemy wartość oczekiwaną $E[L(\theta|\mathcal{Z})]$ względem zmiennej parametrów θ:

$$\theta^i = argmax_\theta\, Q(\theta, \theta^{i-1}) \tag{3.34}$$

Obydwa kroki są powtarzane. W każdej iteracji otrzymujemy co raz dokładniejsze estymatory parametrów.

W pracy [31] można znaleźć dowód zbieżności algorytmu EM. Ponieważ optymalizacja funkcji *Q* często jest analitycznie niemożliwa w implementacjach praktycznych wyznacza się często jedynie taką wartość θ^i dla której wartość funkcji *Q* jest większa niż wartość z poprzedniej iteracji. Taki algorytm jest również zbieżny.

3.7.1 Wariant algorytmu uczenia parametrów modelu HMM

W punkcie 3.6 przedstawiono algorytm wyznaczania parametrów HMM w oparciu o wyrafinowane metody analityczne. W punkcie tym przedstawię wariant algorytmu Forward-Backward opracowany w oparciu o schemat z punktu 3.7 wzory podaję w oparciu o pracę [35].

Przypomnijmy formalną definicję optymalnego dobory parametrów modelu HMM:

> *Dane są ciąg obserwacji* $O = (o_1, o_2, \dots, o_T)$ *oraz zbiór możliwych stanów ukrytych modelu* $S = \{1..N\}$*, znajdź parametry A i B maksymalizujące P(O|λ).*

W zadaniu tym ciąg obserwacji O jest realizacją zmiennej losowej obserwowalnej, natomiast ciąg stanów $Q = (q_1 q_2 \dots q_T)$ jest realizacją zmiennej ukrytej nie obserwowalnej w doświadczeniu. Zatem funkcja wiarygodność dla kompletnych danych wygląda następująco:

$$L(O, Q|\lambda) = P(O, Q|\lambda) \tag{3.35}$$

Toteż funkcja Q przyjmuje postać:

$$Q(\lambda, \lambda^{i-1}) = \sum_{Q \in V} log P(O, Q|\lambda) P(O, Q|\lambda^{i-1}) \quad (3.36)$$

gdzie V jest zbiorem wszystkich możliwych ciągów o długości T, których elementy składają się z numerów stanów; λ^{i-1} - jest estymatorem parametrów z poprzedniej iteracji.

Wyrażenie $P(O, Q|\lambda)$ oznacza prawdopodobieństwo wygenerowania ciągu obserwacji O przez sekwencję stanów ukrytych $Q = (q_1 q_2 \dots q_T)$ obliczenie tego prawdopodobieństwa nie sprawia większych problemów:

$$P(O, Q|\lambda) = \prod_{t=1}^{T} a_{q_{t-1} q_t} b_{q_t}(o_t) \quad (3.37)$$

Następnie wstawiamy (3.37) do (3.36):

$$Q(\lambda, \lambda^{i-1}) = \sum_{Q \in V} \left(\sum_{t=1}^{T} log a_{q_{t-1} q_t} \right) P(O, Q|\lambda^{i-1}) + \sum_{Q \in V} \left(\sum_{t=1}^{T} log b_{q_t}(o_t) \right) P(O, Q|\lambda^{i-1}) \quad (3.38)$$

Obydwa wyrażenia możemy zoptymalizować oddzielnie.

Pierwsze wyrażenie przyjmuje postać:

$$\sum_{Q \in V} \left(\sum_{t=1}^{T} log a_{q_{t-1} q_t} \right) P(O, Q|\lambda^{i-1}) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=1}^{T} log a_{ij} P(O, q_{t-1} = i, q_t = j|\lambda^{i-1})$$

Aby znaleźć ekstremum tego wyrażenia należy zastosować mnożnik Lagrange'a z warunkiem $\sum_{j=1}^{N} a_{ij} = 1$, maksimum otrzymujemy dla wartości parametrów:

$$a_{ij} = \frac{\sum_{t=1}^{T} P(O, q_{t-1}=i, q_t=j|\lambda^{i-1})}{\sum_{t=1}^{T} P(O, q_{t-1}=i|\lambda^{i-1})} \quad (3.29)$$

Drugie wyrażenie równania (3.38) możemy rozpisać następująco:

$$\sum_{Q \in V} \left(\sum_{t=1}^{T} \log b_{q_t}(o_t) \right) = \sum_{i=1}^{N} \sum_{t=1}^{T} \log b_i(o_t) P(O, q_t = i|\lambda^{i-1})$$

Ponownie stosując metodę mnożników Lagrange'a tym razem z warunkiem $\sum_{k=1}^{m} b_i(v_k) = 1$, maksimum otrzymujemy dla rozkładu:

$$b_i(v_k) = \frac{\sum_{t=1, o_t=v_k}^{T} P(O, q_t=i|\lambda^{i-1})}{\sum_{t=1}^{T} P(O, q_t=i|\lambda^{i-1})} \quad (3.30)$$

Metoda wyznaczania parametrów modelu HMM w punkcie 3.6 jest o wiele łatwiejsza analitycznie oraz posiada intuicyjną interpretację, dlatego jest znacznie częściej spotykana w literaturze fachowej.

4 Ukryte modele Markowa w systemach ASR

4.1 Wstęp

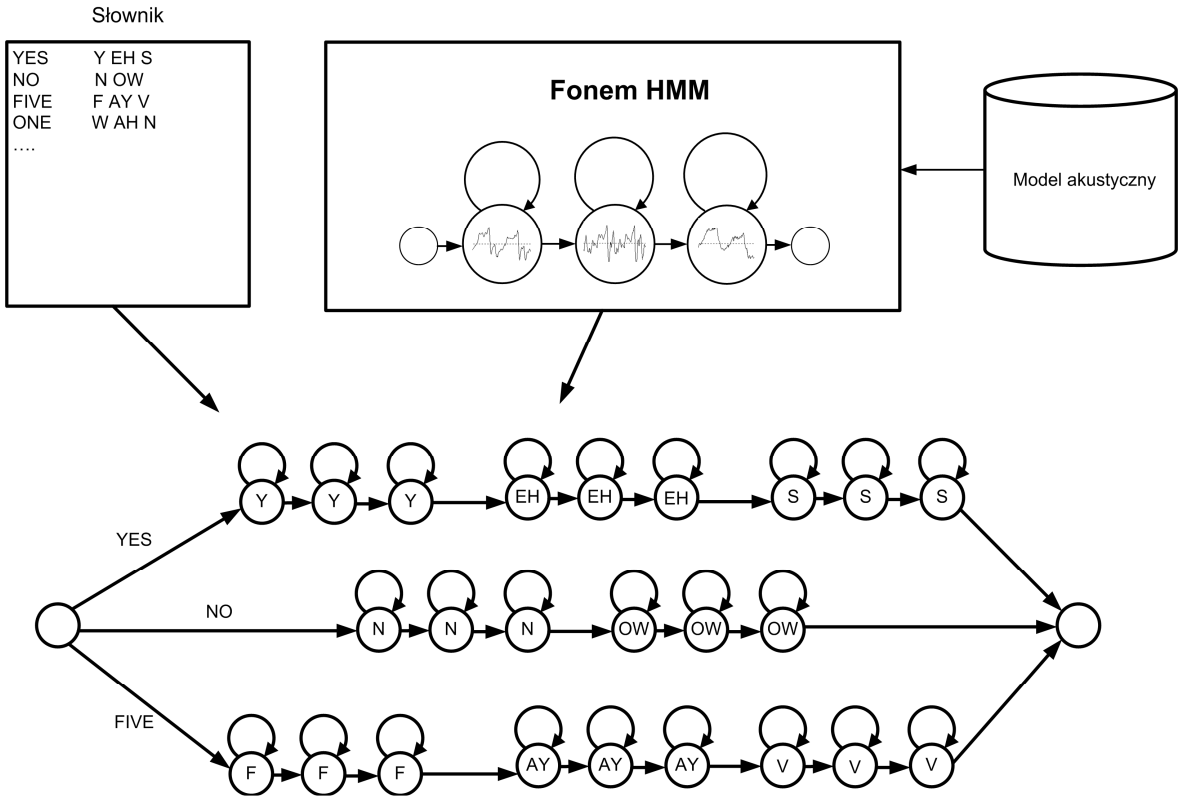
Ukryte modele Markowa znalazły szerokie zastosowanie w systemach rozpoznawania mowy. Przyczyny tego faktu są dwie. Po pierwsze ukryte modele Markowa dostarczają aparat matematyczny pozwalający swobodnie modelować proces wytwarzania dźwięków mowy przez człowieka. Automat HMM charakteryzuje statystyczne właściwości źródła generującego falę akustyczną sygnału mowy. Dodatkowo model HMM nie narzuca projektantowi systemu ASR żadnego rozwiązania modelującego deterministyczne atrybuty sygnału akustycznego. Modele HMM można zatem bez większych przeróbek łączyć z różnorodnymi algorytmami ekstrakcji cech akustycznych (np. MFCC, PLP, HFCC). Po drugie modele HMM są proste w realizacji inżynierskiej. Algorytmy dekodowania i nauki są proste w implementacji oraz posiadają akceptowalną złożoność czasową. Dodatkowo algorytm nauki w praktyce działa w trybie nienadzorowanym. Systemy ASR korzystające z ukrytych modeli Markowa posiadają również sporo większą skuteczność rozpoznawania niż implementacje korzystające z odmiennych mechanizmów.

W systemach ASR przeznaczonych do rozpoznawania mowy ciągłej słowa reprezentowane są za pomocą ciągu fonemów. Fonem jest z kolei najmniejszą jednostką mowy rozróżnialną przez użytkownika danego języka. Następnie fonem dekomponowany jest na mniejsze jednostki fonetyczne reprezentowane przez stany ukryte modelu HMM. Technicznie sygnał mowy jest niestacjonarny, a więc jego właściwości statystyczne w czasie nie są stałe. Natomiast dla krótkich przedziałów czasu (ok. 10ms) sygnał jest quasi-stacjonarny. Te quasi-stacjonarne cząstki reprezentowane są przez stany HMM.

Proces generowania mowy przebiega liniowo. Fonemy realizujące słowo są wypowiadane od pierwszego do ostatniego. Zatem ukryte modele Markowa stosowane w systemach ASR nie reprezentują pełnego schematu połączeń pomiędzy stanami ukrytymi. Najczęściej stosowana topologia w systemach ASR to schemat liniowy przedstawiony na rysunku 3.4.

Słowa tworzące gramatykę zamieniane są za pomocą słownika transkrypcji fonetycznej na ciąg fonemów realizujących słowo. Następnie z modelu akustycznego pobierany jest automat HMM realizujący fonem. Modele HMM łączone są w słowa. Natomiast słowa łączone są w jeden graf tworzący przestrzeń

poszukiwań wykorzystywaną przez dekoder. Schematycznie architektura systemu ASR w oparciu o modele HMM została przedstawiona na rysunku 4.1.



Rys. 4.1 – Topologia grafu przeszukiwań

4.2 GMM

W poprzednim rozdziale podczas omawiania algorytmu Baum'a-Welch'a zakładaliśmy, że obserwacje O przyjmują wartości z pewnego skończonego zbioru. Każdy element tego zbioru reprezentował pewien prototypowy element. Następnie podczas nauki modelu HMM estymowaliśmy rozkład prawdopodobieństwa emisji obserwacji dla każdego stanu ukrytego. W podejściu tym pojawia się jednak kilka problemów. Pierwszy z nich to liczba elementów prototypowych, która musi zostać dobrana *a priori*. Drugi problem to odwzorowanie przestrzeni d-wymiarowej wektora cech na zbiór elementów prototypowych. Poza tym produkcja mowy nie jest procesem opisanym wartościami symbolicznymi.

Współczesne systemy rozpoznawania mowy parametry akustyczne modelują za pomocą wartości ciągłych. Zamiast estymować rozkład prawdopodobieństwa emisji obserwacji estymujemy parametry funkcji gęstości prawdopodobieństwa. Najszerzej stosowanym kształtem rozkładu jest rozkład

będący superpozycją wielu wielowymiarowych rozkładów Gaussa (ang. Gaussian mixture model GMM). Warto tutaj zwrócić uwagę na fakt, że kształt rozkładu jest pewną wiedzą *a priori*. Założenie z góry kształtu (niepopartego wcześniejszymi badaniami czy testami statystycznymi) rozkładu prawdopodobieństwa w wielu przypadkach jest założeniem z byt mocnym. Badania przeprowadzone przez Rabiner'a wykazały, że cechy akustyczne sygnału mowy dla różnych mówców posiadają rozkład gamma. Jednak pamiętajmy, że za pomocą dostatecznej liczby komponentów Gaussa możemy z wystarczającą dokładnością aproksymować dowolny rozkład. Dodatkowo rozkład GMM posiada wygodny i łatwy w implementacji aparat matematyczny.

Jednowymiarowy rozkład Gaussa opisany jest następującym wzorem:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\ (-\frac{(x-\mu)^2}{2\sigma^2}) \tag{4.1}$$

gdzie μ oznacza wartość średnią, a σ jest odchyleniem standardowym.

W zadaniu rozpoznawania mowy wektor cech jest wielowymiarowy np. wektor MFCC posiada 39 składowych. Aby przypisać prawdopodobieństwo emisji dla d-wymiarowego punktu obliczamy wielowymiarową funkcję Gaussa:

$$f(\bar{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} exp\ (-\frac{1}{2}(\bar{x}-\bar{\mu})^T\Sigma^{-1}(\bar{x}-\bar{\mu})) \tag{4.2}$$

gdzie $\bar{x}$ jest *d*-wymiarowym wektorem cech, $\bar{\mu}$ - to *d*-wymiarowy wektor wartości średnich, natomiast Σ jest macierzą kowariancji składającą się z *d* kolumn i *d* wierszy. Każde element macierzy kowariancji określa zależność liniową pomiędzy dwoma wymiarami wektora cech. W przypadku gdy pojedyncze składowe wektora cech są nieskorelowane między sobą, wszystkie elementy macierzy kowariancji poza główną przekątną przyjmują wartość zero. Macierz wyłącznie z niezerowymi elementami na głównej przekątnej nazywana jest macierzą diagonalną. Składowe wektora cech MFCC są niezależne, dlatego budując model akustyczny dla wektora cech MFCC estymujemy jedynie elementy znajdujące się na głównej przekątnej macierzy kowariancji.

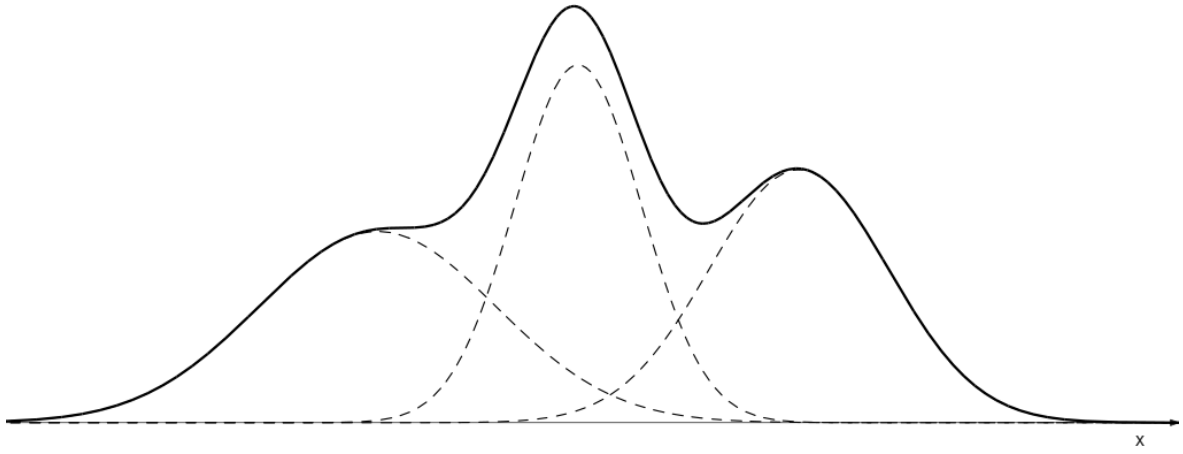
Niestety składowe wektora cech MFCC nie posiadają rozkładów normalnych. Aby zamodelować dowolny rozkład za pomocą skończonej liczby komponentów Gaussa stosujemy tzw. technikę kompozycji funkcji gęstości (ang. *mixture density*). Dowolny rozkład modelujemy za pomocą sumy komponentów Gaussa. Każdy komponent posiada wagę c_k. Wypadkowa funkcja gęstości jest zatem kombinacją wypukłą funkcji Gaussa:

$$f(x) = \sum_{k=1}^{M} c_k p_k(x), gdzie\ c_1, \dots, c_M \geq 0, \sum_{k=1}^{M} c_k = 1 \tag{4.3}$$

Ostateczną postać funkcji gęstości prawdopodobieństwa GMM możemy zapisać za pomocą następującego wzoru:

$$f(\bar{x}) = \sum_{k=1}^{M} c_k \frac{1}{\sqrt{2\pi|\Sigma_k|}} \exp\ (-\frac{1}{2}(\bar{x}-\bar{\mu}_k)^T\Sigma_k^{-1}(\bar{x}-\bar{\mu}_k)) \tag{4.4}$$

Na rysunku 4.2 przedstawiono kształt funkcji gęstości złożonej z trzech komponentów Gaussa.



Rys. 4.2 – Komponenty Gaussa oznaczono linią przerywaną, natomiast linią pogrubioną oznaczono funkcję GMM

4.2.1 Rozszerzenie algorytmu Baum'a-Welch'a dla GMM

Aby estymować wartości średnie, wariancji i wag dla poszczególnych komponentów GMM musimy zmodyfikować definicję prawdopodobieństwa gamma z poprzedniego rozdziału. Niech $\gamma_t(j,k)$ oznacza prawdopodobieństwo, że w chwili *t* automat znajduje się w stanie *j* i emituje symbol o_t za pomocą *k*-tego komponentu. Wtedy re-estymujemy prawdopodobieństwo gamma następująco:

$$\gamma_t(j,k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(j)}\right]\left[\frac{c_{jk}G(o_t,\mu_{jk},\Sigma_{jk})}{\sum_{m=1}^{M}c_{jm}G(o_t,\mu_{jm},\Sigma_{jm})}\right] \tag{4.5}$$

Aby obliczyć wartość c_{jk} musimy obliczyć stosunek oczekiwanej liczby razy, że system jest w stanie *j* i korzysta z komponentu *k* do oczekiwanej liczby razy, że automat jest w stanie *j*:

$$c_{jk} = \frac{\sum_{t=1}^{T}\gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{m=1}^{M}\gamma_t(j,m)} \tag{4.6}$$

Stosując podobny tok myślenia obliczamy wartości wektora średnich i macierzy kowariancji:

$$\bar{\mu}_{jk} = \frac{\sum_{t=1}^{T}\gamma_t(j,k)\bar{o}_t}{\sum_{t=1}^{T}\gamma_t(j,k)} \tag{4.7}$$

$$\bar{\Sigma}_{jk} = \frac{\sum_{t=1}^{T}\gamma_t(j,k)(\bar{o}_t-\bar{\mu}_{jk})(\bar{o}_t-\bar{\mu}_{jk})^T}{\sum_{t=1}^{T}\gamma_t(j,k)} \tag{4.8}$$

Należy w tym miejscu podkreślić, że wartości funkcji gęstości prawdopodobieństwa GMM nie są tak naprawdę prawdopodobieństwami.

Ponieważ obliczamy wartość funkcji GMM w pojedynczym punkcie, toteż wartość ta może być sporo większa od 1 (dla malej wariancji). Aby otrzymać prawidłową wartość prawdopodobieństwa należy obliczyć całkę z funkcji gęstości prawdopodobieństwa w określonym przedziale. Całka dla pojedynczego punktu wynosi zero. Prawdopodobieństwo obserwacji pojedynczego punktu z nieskończonego przedziału dąży zatem do zera. W trakcie rozpoznawania mowy dla każdej klasyfikowanej ramki posiadamy wektor cech reprezentujący punkt w przestrzeni *d*-wymiarowej. Prawdopodobieństwo obserwacji tego punktu wynosi zatem zero. Jeżeli prawdopodobieństwo obserwacji dowolnego wektora cech jest równe zero to problem rozpoznawania mowy jest wtedy nierozwiązywalny. Aby otrzymać prawidłową wartość prawdopodobieństwa należy policzyć całkę dla niewielkiego przedziału dx wokół punktu wektora cech. Jednak zamiast obliczać całkę możemy używać wartości funkcji gęstości tak długo, gdy wartości te dla różnych funkcji i obserwacji są porównywalne. Ponieważ w zadaniu rozpoznawania mowy za każdym razem posiadamy tę samą skalę dla osi x i y, toteż w systemach ASR korzysta się z wartości funkcji gęstości.

4.3 Skala logarytmiczna

Ze względu na prostotę wszystkie wzory przedstawione do tej pory przedstawione były w skali liniowej. Jednak w systemach ASR wszystkie obliczenia przeprowadzane są w skali logarytmicznej. Główną przyczyną stosowania skali logarytmicznej jest zjawisko niedomiaru arytmetycznego (ang. *arithmetic underflow*) w liczbach zmiennoprzecinkowych. Niedomiar arytmetyczny pojawia się, gdy w wynik operacji na liczbach zmiennoprzecinkowych jest pomiędzy wartością zero i najmniejszą możliwą wartością w standardzie liczb zmiennoprzecinkowych. Wartość takiej operacji jest z reguły zaokrąglana do zera. Obliczając prawdopodobieństwo dla całych zdań podczas treningu i dekodowania mnożymy bardzo dużo (często kilka tysięcy) małych wartości prawdopodobieństwa. Tym samym zjawisko to pojawia się już w pierwszej fazie implementacji systemu ASR. Natomiast stosując skalę logarytmiczną podczas obliczania prawdopodobieństwa dla zdań dodajemy wiele ujemnych wartości, których rezultaty mieszczą się w zakresie liczb zmiennoprzecinkowych.

4.4 Trening

W poprzednich punktach omówiono podstawowe elementy treningu HMM. W punkcie tym zostaną omówione rozszerzenie algorytmu Baum'a-Welch'a stosowane podczas budowy modelu akustycznego. W punkcie 4.2 przedstawiono metodę estymacji parametrów funkcji gęstości GMM.

Zadanie treningu HMM podczas budowy modelu akustycznego nie polega na nauce struktury wewnętrznych połączeń pomiędzy stanami ukrytymi. Modelujemy każde słowo za pomocą struktury liniowej HMM (rys. 3.4). Kolejne

stany automatu HMM określamy na postawie słownika z transkrypcją fonetyczną. Zazwyczaj modelujemy pojedyncze jednostki fonetyczne, a nie całe słowa. Zatem pojedynczy automat HMM reprezentujący fonem występuje podczas treningu w wielu słowach.

4.4.1 Inicjalizacja

Algorytm Baum'a-Welch'a działa w trybie bez nadzoru. Dane wejściowe algorytmu to zbiór stanów ukrytych oraz ciąg wektorów cech wypowiedzi. Podczas każdej iteracji na podstawie estymatorów parametrów HMM z poprzedniej iteracji dobieramy coraz dokładniejsze wartości tych parametrów. Dodatkowo algorytm podczas nauki uczy się również segmentacji wypowiedzi na elementarne jednostki fonetyczne. Zatem nie musimy ręcznie określać w którym miejscu rozpoczynają się i kończą poszczególne jednostki fonetyczne.

Jednakże skuteczność algorytmu *forward-backward* zależy w dużym stopniu od wartości początkowych wyliczanych estymatorów parametrów. Najpowszechniej stosowaną metodą inicjalizacji parametrów, w zadaniu rozpoznawania mowy, jest tzw. technika *flat start*. W pierwszej kolejności ustawiamy na zero wszystkie przejścia pomiędzy stanami ukrytymi automatu HMM, które uznajemy za niedozwolone. W systemach ASR dozwolone jest tylko przejście do następnego stanu i pozostanie w stanie obecnym. Toteż każdemu przejściu przypisujemy równe prawdopodobieństwo 0.5. Podczas wyliczania kolejnych wartości prawdopodobieństwa przejść a_{ij} korzystamy z poprzednich wartości, zatem prawdopodobieństwa przejść zainicjalizowane zerami nie zmienią się podczas procesu nauki. Następnie inicjalizujemy wszystkie wartości wektorów μ_{jk} i Σ_{jk} wartościami średniej i wariancji obliczonymi dla całego zbioru treningowego.

4.4.2 Embedded training

Najlepsze rezultaty rozpoznawania mowy uzyskujemy, gdy każdy automat HHM reprezentuje pojedyncze słowo. Jednak ze względu na ograniczoną liczbę próbek uczących stosujemy tzw. *embedded training*. Technika ta polega na modelowaniu za pomocą automatu HMM pojedynczych fonemów. Automaty HMM są następnie składane w jeden większy model HMM reprezentujący zdanie próbki uczącej. Oznacza to, żę każdy fonem HMM jest umieszczany w różnych słowach elementów treningowych. Dla każdego tak zbudowanego modelu zdania parametry oszacowane podczas treningu HMM trafiają do akumulatora. Po przetworzeniu wszystkich elementów treningowych z zakumulowanych estymatorów cząstkowych parametrów liczona jest wartość oczekiwana. Wartość ta jest następnie wykorzystywana podczas następnych iteracji algorytmu nauki.

Procedurę nauki kończymy zazwyczaj po 2-5 iteracjach. Większa liczba iteracji powoduje przeuczenie modelu.

4.4.3 Mixture splitting

Liczba komponentów Gaussa tworzących rozkład GMM musi być zanana z góry. Zbyt mała liczba komponentów niedokładnie modeluje proces generowania sygnałów mowy i w efekcie uzyskujemy słabe wyniki klasyfikacji. Natomiast zbyt duża liczba komponentów Gaussa powoduje, że podczas nauki parametry zapamiętują szumy towarzyszące danym treningowym oraz dokładnie dopasowują się do zbioru uczącego. Ponieważ pełna procedura *embedded training*'u jest procesem długotrwałym zazwyczaj iteracyjnie zwiększamy liczbę komponentów Gaussa poprzez 'rozdwojenie' komponentów z poprzedniej iteracji. Model akustyczny dla każdej liczby komponentów GMM zachowujemy i sprawdzamy stopień generalizacji takiego modelu na danych testowych. Proces rozdzielania (ang. *mixture splitting*) kończymy, gdy uzyskamy maksymalna liczbę komponentów lub gdy stopień generalizacji zaczyna maleć.

Procedura rozdzielania komponentów przebiega następująco:

1) Rozpoczynamy embedded training z jednym komponentem Gaussa. Wykonujemy 2-5 iteracji. Pod koniec fazy obliczamy stopień generalizacji modelu akustycznego dla danych testowych.

2) Następnie wybieramy *n* komponentów dla każdego stanu z największymi wagami i wykonujemy operację split wg. następującego schematu:

$$c_{jk}G(\mu_{jk},\Sigma_{jk}) \rightarrow \begin{cases} \frac{c_{jk}}{2}G(\mu_{jk}-0.2\sigma_{jk},\Sigma_{jk}) \\ \frac{c_{jk}}{2}G(\mu_{jk}+0.2\sigma_{jk},\Sigma_{jk}) \end{cases}$$

3) Wykonujemy 2-5 iteracji i ponownie sprawdzamy stopień generalizacji modelu. Jeśli stopień generalizacji zmniejszył się kończymy procedurę. W przeciwnym wypadku przechodzimy ponownie do kroku 2.

4.4.4 Viterbi training

Alternatywą dla treningu Baum'a-Welch'a jest tak zwany trening Viterbi'ego. Podczas algorytmu *forward-backward* analizujemy wszystkie możliwe ścieżki automatu HMM oraz podejmujemy tzw. miękkie decyzje (ang. *soft decisions*). Termin *soft decisions* oznacza, że algorytm Baum'a-Welch'a nie wyznacza jednoznacznie granicy pomiędzy poszczególnymi stanami w sygnale mowy, a jedynie z pewnym prawdopodobieństwem określa te granice. Natomiast w trakcie treningu Viterbi'ego analizujemy jedynie ścieżkę wyznaczoną za pomocą algorytmu Viterbi'ego oraz korzystamy z segmentacji dokonanej poprzez tę ścieżkę. Dla segmentacji wyznaczonej poprzez algorytm Viterbi'ego szacujemy wartości parametrów modelu HMM. Ponieważ algorytm ten dokonuje tzw. twardych decyzji (ang. *hard decisions*) estymowanie wartości parametrów polega na zliczaniu i obliczaniu wartości oczekiwanych. Parametry modelu w tym trybie nauki inicjalizujemy również za pomocą techniki *flat start*. Zatem podczas

pierwszej iteracji algorytm przeprowadza równomierną segmentację – dzieląc ciąg obserwacji na równe segmenty.

Trening Viterbi'ego jest znacznie szybszy niż algorytm Baum'a-Wech'a. Jednak estymatory parametrów nie są tak dokładne jak w przypadku algorytmu EM. Najszersze zastosowanie trening Viterbi'ego znalazł w systemach hybrydowych HMM/MLP lub HMM/SVM.

4.4.5 Listing algorytmu embedded training

Poniżej znajduje się pseudokod algorytmu *embedded training* zrealizowanego w ramach pracy dyplomowej:

```
prepareTrainingSet()
performFlatInitialization()
do begin
  for each trainigItem do
    performEStep
      forwardPass()
      backwardPass()
    endEStep
    newEstimates = performMStep()
    accumulator.collectEstimates(newEstimates)
    acousticModel.update(accumulator.getEstimates())
  endFor
  accumulator.normalizeEstimates()
  if noMixtures < desiredNoMixture then
    mixtureSplitter.split(acousticModel)
  endIf
while not convergence or noMixture < desiredNoMixtures
```

Listing 4.1 – Embedded training

4.5 Decoding

Algorytm Viterbi'ego podczas wyznaczania najbardziej prawdopodobnej ścieżki dla ciągu obserwacji analizuje wszystkie możliwe przejścia w kratownicy przeszukiwań. Dla każdego węzła dodatkowo obliczana jest wartość funkcji gęstości prawdopodobieństwa GMM. W wyniku tego dla dużych gramatyk czas dekodowania jest sporo dłuższy od czasu trwania dekodowanego słowa. Dlatego

aby proces rozpoznawania mowy przebiegał w czasie rzeczywistym, pomijamy ścieżki posiadające małe prawdopodobieństwa. Optymalizacja ta nosi nazwę *beam search*. Algorytm ten sparametryzowany jest za pomocą promienia poszukiwań *r* (ang. *beam*), który ogranicza liczbę analizowanych ścieżek.

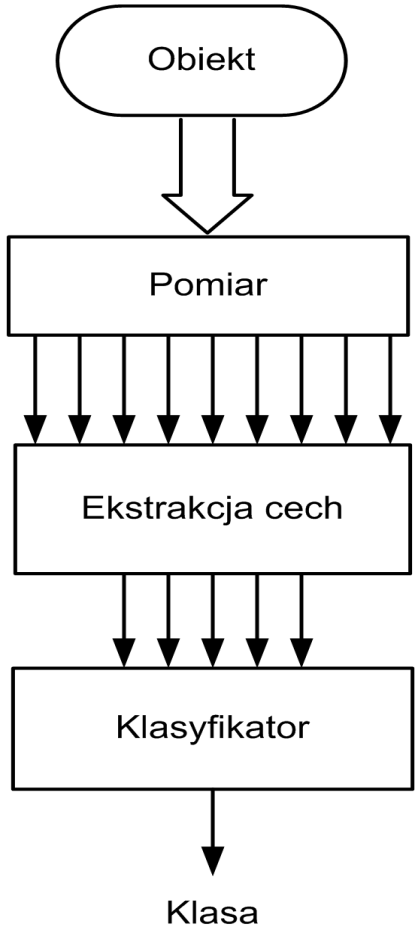
W trakcie dekodowania dla każdej chwili t obliczamy prawdopodobieństwo ścieżki łączącej węzeł startowy z aktualnie analizowanym węzłem. Następnie usuwamy wszystkie ścieżki posiadające najmniejsze prawdopodobieństwo. Zatem pozostawiamy jedynie *r* ścieżek częściowych posiadających największe prawdopodobieństwo. W chwili t+1 analizujemy wszystkie przedłużenie o jeden węzeł ścieżek z chwili t. Następnie ponownie usuwamy ścieżki z najmniejszym prawdopodobieństwem pozostawiając *r* ścieżek najbardziej prawdopodobnych.

Wszystkie ścieżki częściowe reprezentowane są za pomocą token'ów. Token taki zawiera referencję na aktualny wierzchołek, prawdopodobieństwo dla ścieżki oraz wskaźniki wstecz pozwalające odtworzyć ścieżkę Viterbi'ego. Wszystkie ścieżki przechowywane są w tzw. *active list*. Natomiast procedura usuwania najmniej prawdopodobnych ścieżek nazywana jest ang. *pruning*.

5 Ekstrakcja cech akustycznych

5.1 Wstęp

Pierwszym etapem przepływu danych w systemie rozpoznawania obiektów jest pozyskanie informacji opisującej obiekt. Zakładamy, że każdy rozpoznawany obiekt jest opisany za pomocą zbioru atrybutów podlegających pomiarowi. Niestety atrybuty bezpośrednio podlegające obserwacji składają się zazwyczaj z dużej liczby wymiarów oraz poszczególne składowe posiadają wiele informacji wzajemnej (ang. *mutual information*) o pozostałych składowych. Ponadto atrybuty opisujące obiekt posiadają zazwyczaj sporo informacji nieistotnej podczas klasyfikacji. Toteż proces klasyfikacji zazwyczaj poprzedzony jest etap ekstrakcji cech (ang. *feature extraction*).



Rys. 5.1 – Przepływ danych podczas procesu rozpoznawania

Formalnie ekstrakcję cech możemy zdefiniować jako transformację przestrzeni *p*-wymiarowej wektora atrybutów opisujących obiekt na przestrzeń *d*-wymiarową wektora cech:

$$f: R^p \rightarrow R^d, d < p \qquad (5.1)$$

Transformacja ta może być liniowa, nieliniowa lub superpozycją dowolnych odwzorowań.

Podstawowym celem ekstrakcji cech jest redukcja liczby wymiarów wektora opisującego rozpoznawany obiekt. Zdecydowana większość algorytmów klasyfikacji posiada złożoność czasową w znacznym stopniu zależną od liczby wymiarów. Redukując liczbę wymiarów wektora cech redukujemy czas nauki oraz czas klasyfikacji, ale redukujemy również ilość informacji opisującej obiekt.

Drugą ważną cechą algorytmu ekstrakcji cech jest wydobycie informacji istotnej z punktu widzenia konkretnego problemu rozpoznawania obiektów. Dość

często zdarza się, że składowe wektora cech są między sobą skorelowane. Przykładem może być estymacja parametryczna rozkładu opisanego wielowymiarową funkcją Gaussa. Jeżeli założymy niezależność wymiarów rozkładu estymujemy wtedy elementy diagonalnej macierzy kowariancji. Natomiast jeśli składowe wektora są skorelowane wtedy musimy estymować wszystkie elementy macierzy kowariancji, tym samym zwiększając złożoność modelu. Toteż zadaniem ekstrakcji cech jest pozbycie się informacji nadmiarowej, nieistotnej, a zachowanie informacji kluczowej do rozróżniania obiektów z odmiennych klas.

Z ostatniej myśli wyłania się kolejny ważny aspekt ekstrakcji cech. Konstruując reguły decyzyjne klasyfikatora zakładamy, że obiekty należące do tej samej klasy posiadają podobne właściwości. Rzadko zdarza się, aby właściwości te były identyczne, zazwyczaj mamy do czynienia z pewnym statystycznym rozrzutem koncentrującym się wokół wartości średniej. Natomiast w przypadku obiektów należących do różnych klas zakładamy silną dyspersję. Mówiąc obrazowo większość klasyfikatorów zakłada, że obiekty z tej samej klasy w przestrzeni R^d znajdują się blisko siebie, natomiast obiekty z różnych klas są znacznie oddalone od siebie (ang. *locality property*). Wyekstrahowane cechy powinny separować klasy między sobą. Pamiętajmy tutaj, że w zależności od typu funkcji dyskryminujących otrzymujemy odmienne kształty regionów decyzyjnych, a więc dla różnych klasyfikatorów musimy spełnić odmienne warunki koncentracji obiektów należących do tej samej klasy.

Zadanie klasyfikatora polega na nauce w jaki sposób przestrzeń obiektów podzielona jest między klasami. Znając kształt powierzchni rozdzielających regiony decyzyjne przez klasyfikator, zadaniem ekstrakcji cech jest taka transformacja przestrzeni wejściowej na przestrzeń cech, aby zminimalizować błąd klasyfikacji.

5.2 Reprezentacja sygnału mowy w systemach ASR

Sygnał akustyczny mowy poza informacją fonetyczną przenosi wiele informacji zbędnej z punktu widzenia systemu rozpoznawania mowy. Ta nadmiarowość informacji jest główną przyczyną dużej zmienności parametrów opisujących sygnał. Toteż zadaniem ekstrakcji cech w zadaniu rozpoznawania mowy jest wydobycie jak największej ilości informacji pozwalającej rozróżniać fonemy oraz usunięcie informacji charakterystycznej dla mówcy, kanału transmisyjnego, czy szumów. Tym samym wybór odpowiedniej reprezentacji parametrycznej sygnału akustycznego jest niezmiernie ważnym zadaniem, wpływającym w dużym stopniu na skuteczność systemu rozpoznawania mowy.

Fala akustyczna reprezentująca wypowiadaną głoskę rzadko jest identyczna dla dwóch różnych wypowiedzi. Bardzo często sygnał ten różni się również dla tego samego mówcy. Przyczyny tak dużego rozrzutu składowych sygnału mowy są następujące:

- Sygnał mowy zawiera wiele informacji typowej dla mówcy, takiej jak barwa głosu, ton krtaniowy, emocje. Zatem jedną z przyczyn jest odmienna budowa aparatu mowy dla różnych ludzi.
- Głoski w słowie wymawiane są kolejno w jednym ciągu. Przejścia między sąsiednimi głoskami są płynne. Proces ten zwany fachowo koartykulacją jest również źródłem sporej dyspersji sygnału mowy dla tego samego fonemu.
- Charakterystyka kanału transmisyjnego nie jest idealna, wprowadza on wiele zniekształceń (np. splot sygnału, składowa stała). Zniekształcenia te są w wielu przypadkach trudne do usunięcia.
- Szumy z zewnątrz.

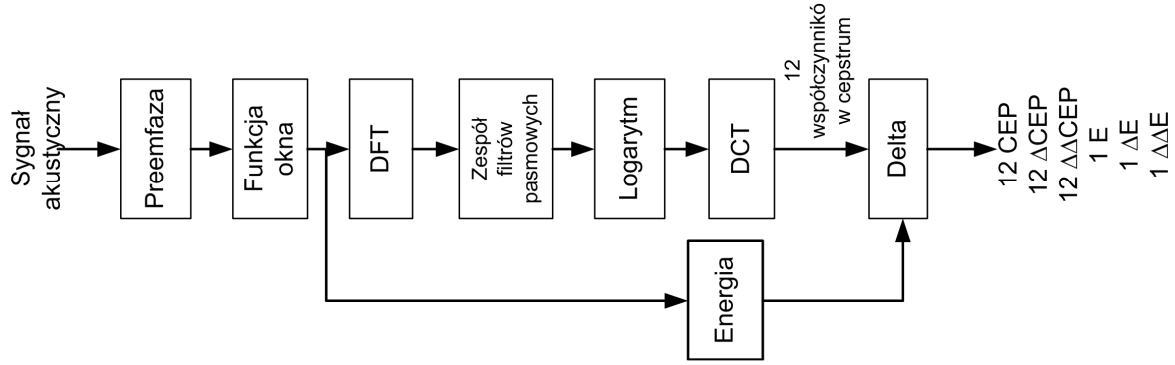
Tym samym parametry reprezentujące sygnał mowy powinny modelować percepcyjne właściwości ludzkiego systemu słuchowego [37]. Dodatkowo parametry te powinny tworzyć taki wektor cech, aby wartości reprezentujące ten sam fonem znajdowały się blisko siebie w przestrzeni cech, natomiast wartości reprezentujące odmienne fonemy umieszczone były jak najdalej od siebie. Ujmując to bardziej formalnie wariancja wewnątrz klasy powinna być mała, wariancja między klasami jak największa. Należy pamiętać, że dobry algorytm ekstrakcji powinien usunąć wariancję parametrów, której źródłem są zjawiska niezwiązane z mową.

5.3 Algorytm MFCC

Algorytm MFCC [32] jest metodą konwertująca próbki sygnału cyfrowego mowy na wektory obserwacji reprezentujące zdarzenia elementarne w przestrzeni probabilistycznej modelu HMM. Reprezentacja sygnału mowy pojawiająca się na wyjściu algorytmu MFCC jest w dużym stopniu niezmienna dla rozmaitych mówców, a jednocześnie parametry MFCC są percepcyjnie istotnym opisem dźwięków. Dodatkowo wektor MFCC posiada jedną bardzo ważną w praktycznych zastosowaniach własność. Składowe wektora cech nie są skorelowane. Niezależność elementów wektora obserwacji jest podstawowym założeniem modeli HMM opisanych w rozdziale 4. Odejmując od parametrów MFCC wartości średnie cepstrum (ang. *cepstral mean substraction*) możemy w łatwy sposób wyeliminować statyczne szumy wprowadzone przez kanał transmisyjny.

Należy jednak podkreślić, że MFCC, jak również alternatywne algorytmy ekstrakcji cech akustycznych mowy, nie są modelem percepcyjnym systemu słuchowego człowieka. Algorytmy te są jedynie inspirowane biologicznymi aspektami ludzkiego aparatu mowy. Algorytm MFCC posiada solidne fundamenty psychoakustyczne opracowane przez neurobiologów. MFCC został jednak skonstruowany empirycznie po to, aby wiernie reprezentować informację fonetyczną w klasyfikatorach komputerowych. Przykładem może być dyskretna

transformata cosinusowa, która nie posiada biologicznej realizacji w aparacie mowy człowieka.

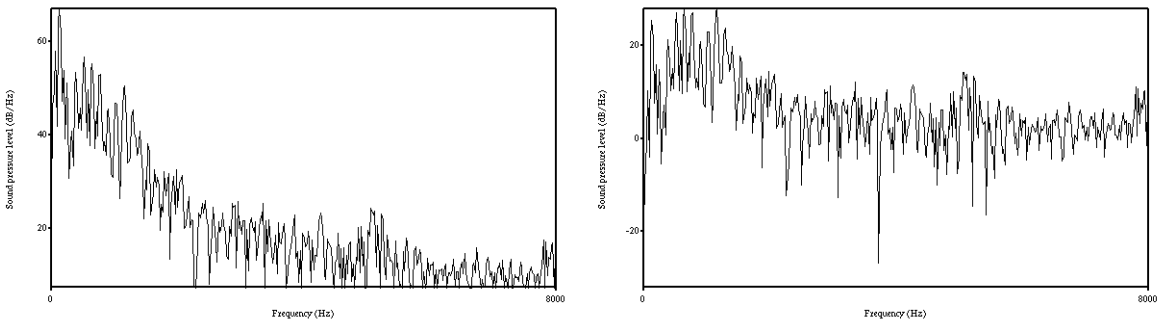


Rys. 5.2 – Przepływ danych w algorytmie MFCC

Rysunek 5.2 ilustruje czynności wykonywane podczas ekstrakcji cech MFCC. W następnych podpunktach omówię szczegółowo kolejne etapy algorytmu MFCC.

5.3.1 Preemfaza

W pracy [38] można przeczytać, że struktura widmowa dźwięków mowy nie jest jednorodna. Najwięcej energii koncentruje się w paśmie do ok. 1000Hz. Powyżej tej częstotliwości energia maleje średnio 20dB na oktawę (patrz rysunek 5.3). Dlatego pierwszym etapem algorytmu MFCC jest wzmocnienie wysokich częstotliwości, aby informacja zawarta w wyższych formantach miała większe znaczenie podczas budowy modelu akustycznego. Empirycznie udowodniono, że przekształcenie to poprawia skuteczność rozpoznania.

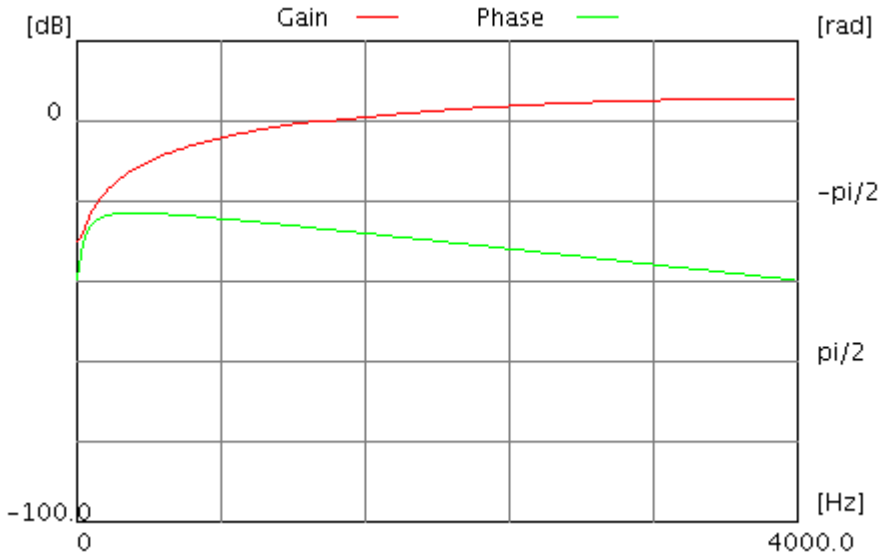


Rys. 5.3 – Widmo fonemu ‘a’ przed preemfazą po lewej i po preemfazie po prawej

Preemfazę pasma wyższych częstotliwości realizuje się za pomocą filtra pierwszego rzędu o skończonej odpowiedzi impulsowej (ang. FIR). Wartość wyjściowa filtru opisana jest wzorem:

$$y[n] = x[n] - 0.97x[n-1] \tag{5.2}$$

Charakterystyka amplitudowa i fazowa filtra opisanego równaniem (5.2) przedstawiona jest na rysunku 5.4.



Rys. 5.4 – Charakterystyka filtra

5.3.2 Funkcja okna

W rozdziale 4 wspomnieliśmy, że sygnał mowy jest niestacjonarny, oznacza to że widmo sygnału mowy zmienia się wraz z upływającym czasem. Jednakże dla bardzo małych przedziałów czasu (ok. 10 ms) możemy założyć, że sygnał jest quasi-stacjonarny. W przedziale tym zmiany spektralne sygnału są na tyle małe, że możemy je pominąć. Ponieważ w systemach ASR elementarną jednostką fonetyczną są sub-fonemy, toteż nie możemy wyekstrahować cech dla całej wypowiedzi. Tym samym ekstrakcja cech akustycznych przeprowadzana jest dla niewielkich przedziałów, w których spektrum sygnału mowy jest prawie stacjonarne.

Operacja ekstrakcji stacjonarnego fragmentu sygnału polega na wymnożeniu funkcji okna przez sygnał wejściowy. Okno czasowe posiada niezerowe wartości dla interesującego nas regionu i zerowe poza nim. Funkcję okna czasowego opisują trzy parametry: szerokość wyrażona w milisekundach, przesunięcie pomiędzy kolejnymi fragmentami, kształt okna. W zadaniu rozpoznawania mowy wyekstrahowane próbki za pomocą funkcji okna nazywamy ramkami (ang. *frame*).

Formalnie proces pobierania skończonej liczby próbek z nieskończonego sygnału możemy opisać następującym wzorem:

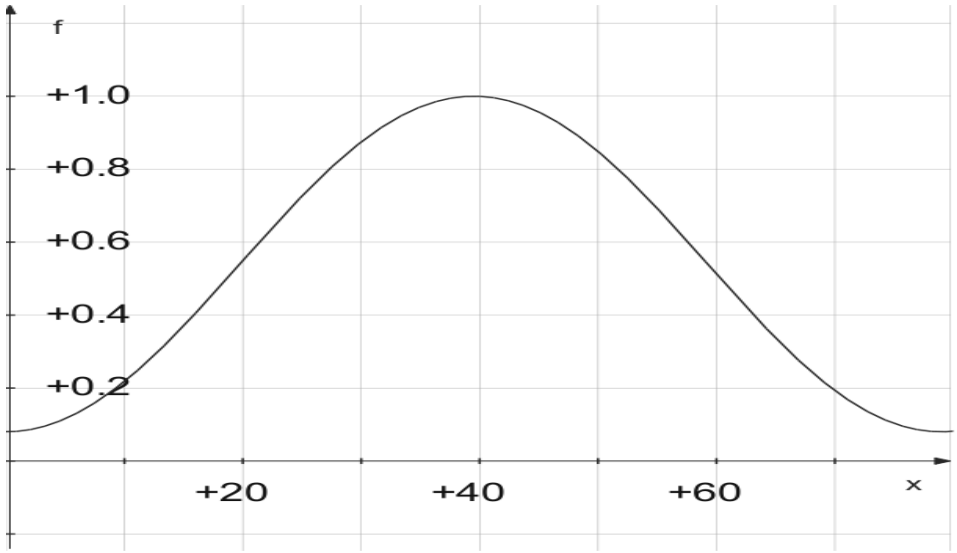
$$y[n] = u[n]w[n], -\infty < n < \infty \tag{5.3}$$

Najprostszą funkcją okna jest okno prostokątne $w[n] = 1$. Funkcja ta ‘ucina’ sygnał na granicy okna wprowadzając tym samym nieciągłość. W pracy [20]

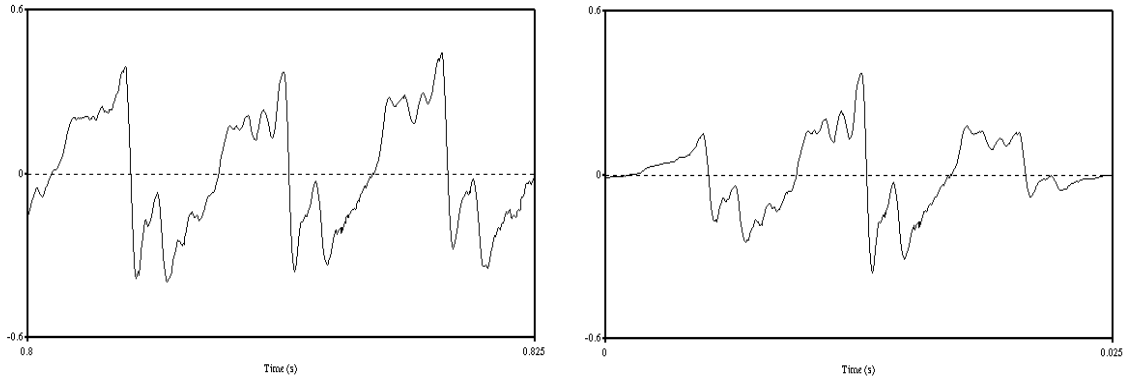
pokazano, że funkcja taka charakteryzuje się największym przeciekiem DFT. Toteż w algorytmie MFCC stosuje się zazwyczaj funkcję Hamminga, która minimalizuje nieciągłości w punktach końcowych przedziału próbkowania. Okno Hamminga opisane jest wzorem:

$$w[n] = \begin{cases} 0.538 - 0.462\cos\left(\frac{2\pi n}{N-1}\right), dla\ 0 \le n < N \\ 0,\ w\ przeciwnym\ razie \end{cases} \tag{5.4}$$

Na rysunku 5.5 przedstawiono kształt okna Hamminga, natomiast rysunek 5.6 ilustruje rezultat funkcji okna Hamminga dla fonemu ‘a’.



Rys. 5.5 – Okno Hamminga dla 80 próbek



Rys. 5.6 – Po lewej okno prostokątne z sygnału, po prawej okno Hamminga z sygnału

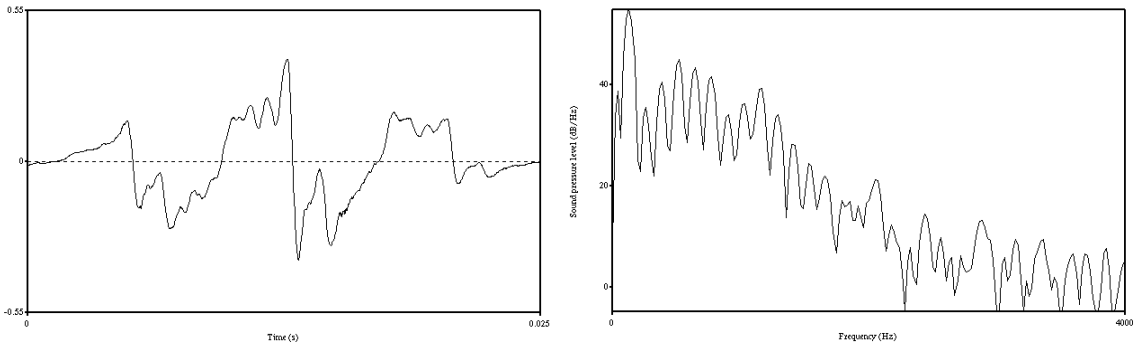
5.3.3 Dyskretna transformacja Fouriera

Kolejnym etapem algorytmu MFCC jest dyskretna transformacja Fouriera (DFT). Transformacja DFT rozkłada sygnał dyskretny na zbiór funkcji okresowych, które realizują sygnał wejściowy. Tym samym z sygnału akustycznego wydobywamy tzw. informację spektralną na podstawie, której możemy określić jak wiele energii zawierają poszczególne podpasma częstotliwości.

Wejściem transformacji DFT jest zbiór wartości otrzymany w wyniku równomiernego próbkowania w dziedzinie czasy ciągłego sygnału mowy, wymnożonego następnie przez funkcję okna, oznaczanego x[n]. Natomiast wyjściem DFT jest transformata X(m) będąca dyskretnym ciągiem w dziedzinie częstotliwości [20] opisanym wzorem:

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N} \tag{5.5}$$

Rysunek 5.7 przedstawia transformatę DFT z okna sygnału reprezentującego fonem ‘a’.



Rys. 5.7 – Dyskretna transformacja Fouriera

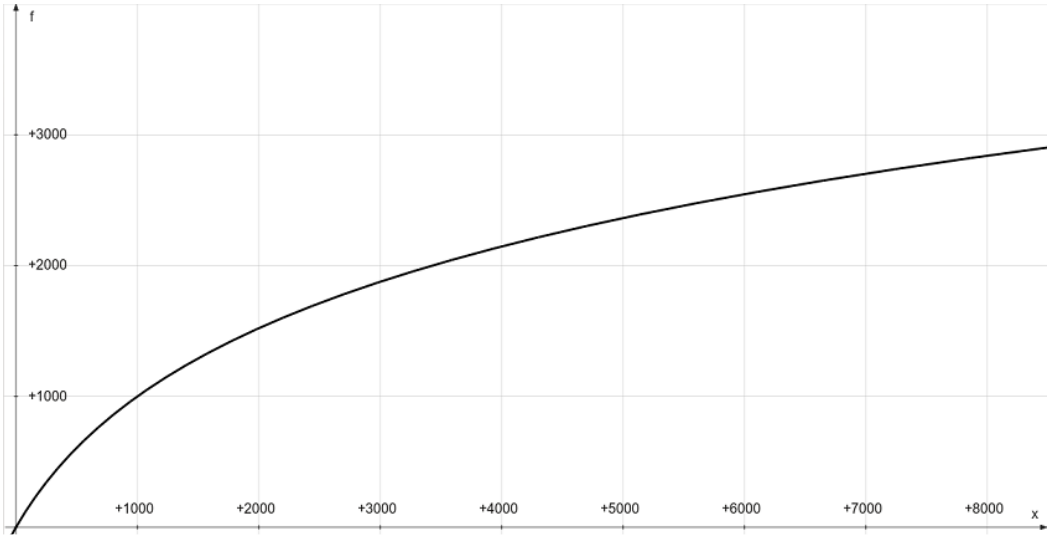
5.3.4 Zespół filtrów pasmowych

Jednym z ważniejszych atrybutów wrażenia słuchowego jest wysokość dźwięku. W dużej mierze wysokość dźwięku determinowana jest przez jego częstotliwość. Najczęściej stosowaną metodyką wyznaczania wysokości tonu jest metoda podwajania wrażenia wysokości dźwięku. W metodzie tej słuchacze dobierają tak częstotliwość tonu testowego, aby wrażenie wysokości było o połowę wyższe od wrażenia wysokości tonu odniesienia [38]. Badania przeprowadzone przez Stevens’a, Volkman’a i Newman’a w 1937 doprowadziły do utworzenia psychofizycznej skali wysokości wrażeń słuchowych wyrażonej w melach. Skala ta charakteryzuje się tym, że dwukrotna liczba meli odpowiada podwojeniu wysokości tonu. Do częstotliwości 1000Hz wysokość tonu wyrażona w melach jest w przybliżeniu równa częstotliwości tonu wyrażonego w Hertzach. Natomiast dla częstotliwości powyżej 1000Hz zaobserwowano, że zależność

wysokości tonu w skali melowej od częstotliwości w Hz jest wyraźnie nieliniowa. W przedziale tym zaobserwowane wyraźny spadek percepcji wysokości dźwięku. Toteż w zastosowaniach inżynierskich wysokość tonu w melach przelicza się za pomocą wzoru:

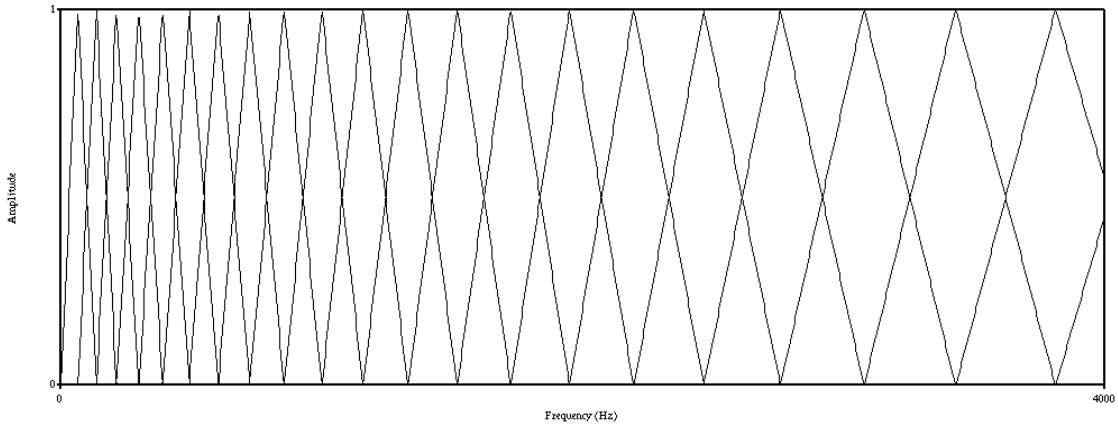
$$m = 1127\ln\left(1 + \frac{f}{700}\right) \tag{5.6}$$

gdzie *f* jest częstotliwością wyrażoną w Hertzach. Na rysunku 5.8 przedstawiono zależność skali melowej od częstotliwości wyrażonej w Hertzach.



Rys. 5.8 – Zależność skali melowej od częstotliwości w Hz

Następnie podczas wyznaczania składowych MFCC stosujemy zespół filtrów pasmowych, którego zadaniem jest dekompozycja sygnału na składowe należące do różnych przedziałów częstotliwości. Mając na uwadze fakt, że w systemie słuchowym szerokość pasm filtrów zmienia się wraz ze zmianą ich częstotliwości środkowej [38]. Dlatego też, w algorytmie MFCC szerokość pasma filtru pasmowego zmienia się wraz z jego częstotliwością środkową. Do częstotliwości 1000Hz stosujemy 10 filtrów rozmieszczonych liniowo. Natomiast pasma filtrów powyżej 1000Hz są rozstawione logarytmicznie. Empirycznie udowodniono również, że najlepsze rezultaty klasyfikacji otrzymuje się dla filtrów realizujących funkcję trójkątną. Na rysunku 5.9 zilustrowano rozlokowanie pasm zespołu filtrów MFCC.



Rys. 5.9 – Zespół filtrów pasmowych MFCC. Każdy filtr trójkątny agreguje energię swojego podpasma.

5.3.5 Logarytm mocy

Badania nad oceną głośności dźwięków dowiodły, że głośność dźwięku jest liniowo zależna od logarytmu amplitudy ciśnienia akustycznego. Ucho ludzkie zatem jest bardziej czułe na niewielkie zmiany amplitudy dla małych poziomów ciśnienia akustycznego niż na zmiany amplitudy dla dużych poziomów ciśnienia. W tym celu wyznaczamy logarytm dla każdej wartości wyjściowej zespołu filtrów pasmowych opisanych w poprzednim punkcie.

5.3.6 Cepstrum

Obwiednia widma sygnału akustycznego uzyskana w poprzednim podpunkcie zawiera charakterystyczną dla mówcy informację akustyczną. Tym samym nie może posłużyć nam jako wektor cech akustycznych w zadaniu rozpoznawania mowy. Dlatego też, kolejnym etapem algorytmu MFCC jest ekstrakcja informacji fonetycznej z obwiedni widma. W tym celu wyznaczamy cepstrum z widma sygnału. Aby wyjaśnić istotę transformacji wyznaczającej cepstrum musimy w pierwszej kolejności omówić model wytwarzania dźwięków mowy.

Model traktu głosowego i źródła drgań (ang. *source-filter model*) jest sposobem odwzorowania przepływu powietrza w torze głosowym i wzbudzenia drgań o strukturze widmowo-czasowej typowej dla jednostek fonetycznych mowy. Mechanizm produkcji głosek dźwięcznych wygląda następująco:

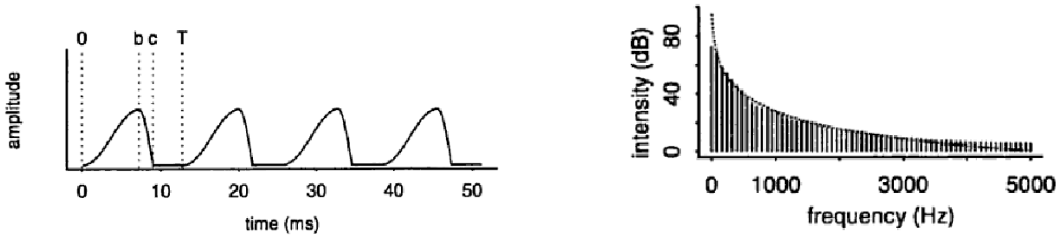
- Powietrze jest wypierane z płuc do tchawicy
- Napięte fałdy głosowe w krtani pod wpływem wypieranego powietrza zaczynają drgać. Fałdy głosowe nie są ze sobą połączone, lecz tworzą tzw. szparę głośni. Gdy szpara jest rozwarta powietrze

przepływa swobodnie, natomiast gdy szpara jest zwężona powietrze opływa fałdy głosowe powodując ich drganie.

- Strumień powietrza jest następnie formowany z quasi-okresowe impulsy poprzez zwieranie i rozwieranie szpary głosowej.
- Następnie tak uformowany sygnał ciśnienia akustycznego jest modulowany podczas wędrówki strumienia powietrza przez komorę krtaniową, gardłową, ustną i nosową.
- Podczas artykulacji objętość i kształt tych komór rezonansowych ulega ciągłym zmianom.

W przypadku głosek bezdźwięcznych szpara głośni jest rozwarta i strumień powietrza wypierany z płuc ma przebieg nieperiodyczny - szumowy.

Tym samym model procesu generowania dźwięków mowy składa się ze źródła reprezentującego drgania głośni oraz filtrów modulujących ton krtaniowy i reprezentujących kształt traktu głosowego. Ton krtaniowy to sygnał o częstotliwości podstawowej charakterystycznej dla mówcy i o widmie składającym się z wszystkich kolejnych harmonicznych. Częstotliwość podstawowa odpowiada częstotliwości drgań szpary głosowej. Natomiast obwiednia tonu krtaniowego posiada nachylenie od –6 do –12 dB/oktawę. Na rysunku 5.10 po lewej przedstawiono zmiany powierzchni szpary głośni w zależności od czasu.



Rys. 5.10 – Drgania głośni po lewej, Widmo drgań głośni po prawej. Rysunki skopiowane z Harrington i Cassidy

Artykulacja określa zmiany traktu głosowego w wyniku, których następuje filtracja tonu krtaniowego. Filtracja sygnału odbywa się poprzez zmiany kształtu komór rezonansowych. Zmiany te następują poprzez odpowiedni układ języka, podniebienia, warg, szczęk itp. Te właśnie parametry kanału głosowego identyfikują wymawiane fonemy. Ponieważ ton krtaniowy zawiera szereg harmonicznych, toteż poprzez zmiany parametrów komór rezonansowych pewne harmoniczne wzmacniamy, a inne tłumimy. Harmoniczne wzmacniane nazywamy formantami i oznaczamy F1,F2 id. Natomiast częstotliwości tłumione nazywamy antyformantami.

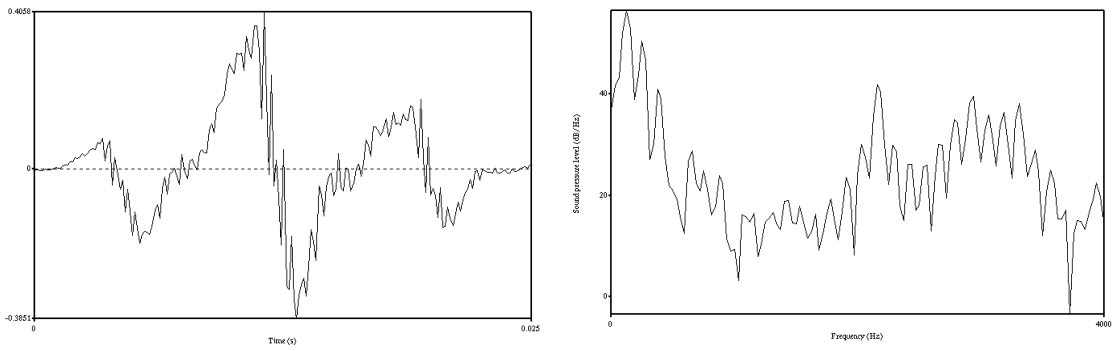
Aby rozdzielić źródło sygnału i zespół filtrów realizowany za pomocą traktu głosowego wyznaczamy cepstrum z sygnału mowy. Parametry filtrów determinują fonem, który jest produkowany przez taki układ aparatu mowy. Proces odseparowania źródła sygnału od filtrów często nazywany jest odwrotną filtracją sygnału (ang. *inverse filtering*). W zadaniu rozpoznawania mowy rolę filtracji odwrotnej spełnia transformacja cepstrum.

Cepstrum jest rezultatem obliczania widma z logarytmu widma sygnału. Zatem widmo sygnału traktujemy jako sygnał okresowy i przechodzimy podczas obliczania cepstrum z dziedziny częstotliwości ponownie do dziedziny czasu. Tym samym cepstrum informuje nas o amplitudzie zmian harmonicznych widma sygnału. Termin cepstrum powstał poprzez odwrócenie kolejności pierwszych liter słowa spectrum. Dlatego też, dziedzina transformaty cepstrum nazywana jest quefrency (poprzez odwrócenie szyku liter słowa frequency). Quefrency jest miarą czasu, jednak nie jest to czas próbkowania sygnału.

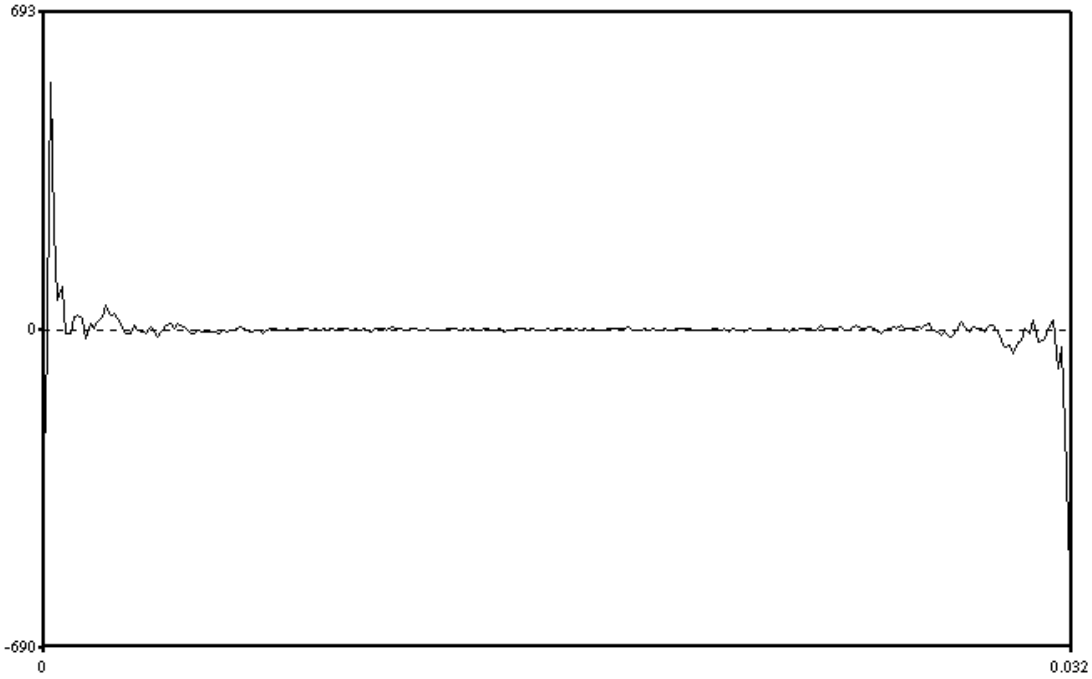
Formalnie transformacja cepstrum dokonuje dekonwolucji sygnału. Dekonwolucja lub inaczej odplatanie jest procesem odwrotnym do splotu sygnału. Podczas procesu dekonwolucji zakładamy, że system akustyczny traktu głosowego jest systemem liniowym oraz źródło jest niezależne od filtra. Termin system liniowy oznacza, że sygnał wyjściowy takiego systemu jest superpozycją pojedynczych składowych podawanych na jego wejście. Obydwa założenia są spełnione dla *source-filter model*. Zatem podczas wytwarzania mowy ton krtaniowy jest spleciony w dziedzinie czasu z filtrem formantów realizowanym przez trakt głosowy. Formalnie proces wytwarzania mowy (wg. modelu source-filter) możemy zapisać następująco:

$$s * f = x \tag{5.7}$$

gdzie *s* jest tonem krtaniowym, *f* funkcją filtra formantów, a *x* sygnałem akustycznym mowy. Splot w dziedzinie czasu jest iloczynem w dziedzinie częstotliwości, natomiast w dziedzinie quefrency jest sumą współczynników cepstrum w różnych regionach.



Rys. 5.11 – Okno Hamminga ze słowa ‘jeden’ oraz jego widmo



Rys. 5.12 – Cepstrum z okna sygnału z rysunku 5.11

Rysunek 5.11 przedstawia okno 25 milisekundowe i jego widmo z wypowiedzi słowa ‘jeden’. Natomiast na rysunku 5.12 przedstawiono pierwszych 256 współczynników cepstrum sygnału zilustrowanego na rysunku 5.11. Oś odciętych przedstawia wartości quefrency. Na wykresie spektrum można zauważyć regularne prążki pojawiające się 30 razy w przedziale 0-4000Hz. Komponent ten posiada zatem częstotliwość ok. 130Hz. Prążki te są kolejnymi harmonicznymi tonu krtaniowego, a 130Hz jest częstotliwością podstawową głosu mówcy (patrz rys. 5.10). Spójrzmy teraz na wykres cepstrum. Dla quefrency równego ok. 0.032 istnieje bardzo duży wierzchołek. Współczynnik ten odpowiada częstotliwości 0.32*4000Hz=128Hz, a więc reprezentuje on częstotliwość podstawową tonu krtaniowego. Ponieważ celem wyliczania cepstrum jest ekstrakcja parametrów filtra z widma sygnału, toteż interesują nas współczynniki reprezentujące małe częstotliwości. Dla kilku pierwszych składowych cepstrum widzimy jeden większy wierzchołek oraz kilka mniejszych. Komponenty te tworzą obwiednie widma, czyli reprezentują strukturę formantową traktu głosowego. Ponieważ podczas rozpoznawania mowy zależy nam na pozbyciu się informacji reprezentującej ton krtaniowy, toteż w algorytmie MFCC bierzemy jedynie 12 pierwszych składowych cepstrum sygnału.

Najczęściej w algorytmie MFCC do obliczania widma z logarytmu widma sygnału wykorzystuje się transformatę cosinusową DCT. Współczynniki cepstrum wyliczone w powyższy sposób posiadają jedną bardzo ważną własność, a

mianowicie składowe wektora MFCC są nieskorelowane. Z tego powodu rozkłady GMM nie muszą być reprezentowane przez pełną macierz kowariancji, co redukuje znacznie liczbę parametrów podlegających nauce.

5.3.7 Delta i energia

Poza 12 współczynnikami cepstrum do każdej ramki dodajemy również wartość energii dla ramki. Samogłoski oraz pozostałe głoski dźwięczne posiadają bardzo dużą wartość energii w porównaniu z głoskami szelestnymi, które reprezentują pseudolosowy szum. Zatem komponent energii pozwala nam odseparować głoski dźwięczne od bezdźwięcznych.

Kolejnym istotnym etapem jest dołączenie różnic pomiędzy sąsiednimi ramkami do wektora MFCC. Pamiętajmy, że sygnał mowy nie jest stały pomiędzy kolejnymi ramkami. Następuje koartykulacja, która powoduje płynne zmiany struktury formantowej widma sygnału akustycznego. Dlatego bardzo istotną informacją potrzebną do identyfikacji fonemów są zmiany pomiędzy kolejnymi ramkami. W tym celu dla 13 współczynników cepstrum i energii obliczamy szybkość zmian i akceleracje zmian.

5.3.8 Podsumowanie

Ostatecznie po dodaniu energii i delt pierwszego i drugiego rzędu, wektor MFCC składa się z 39 składowych. Kolejne komponenty wektora MFCC przedstawia poniższa tabela:

1E	Jeden element energii
12CEP	12 współczynników cepstrum sygnału.
1ΔE	Delta energii pomiędzy aktualną i poprzednia ramką
12ΔCEP	12 delt współczynników cepstrum
1ΔΔE	Delta drugiego rzędu (zwana tez akceleracją) energii
12ΔΔCEP	Delta drugiego rzędu współczynników cepstrum pomiędzy sąsiednimi ramkami

Podsumowując współczynniki MFCC tworzą parametryczny model aparatu głosowego produkującego sygnał mowy. Często są jedyną wiedza *a priori* w zadaniu rozpoznawania mowy, dlatego etap ekstrakcji cech jest tak ważny. Elementy MFCC są nieskorelowane co upraszcza budowę modelu akustycznego.

6 Trening dyskryminujący

6.1 Wstęp

Przypomnijmy, za [2], że w metodzie największej wiarygodności (MLE) dopasowujemy parametry modelu akustycznego (*A* i *B*) tak aby zmaksymalizować wiarygodność danych treningowych. Rozważmy pewną sekwencję obserwacji *O* i pewien model HMM M_k odpowiadający sekwencji stanów W_k.

Kryterium MLE maksymalizuje wiarygodność zdefiniowaną jako:

$$L_{MLE}(\lambda) = max_{\lambda}\, P_{\lambda}(O|M_k) \tag{6.1}$$

Naszym celem podczas implementacji systemu rozpoznawania mowy jest poprawne odróżnianie jednostek fonetycznych na podstawie cech akustycznych. Jednakże powyższe kryterium nam tego nie gwarantuje. Kryterium MLE zapewnia jedynie maksymalizację prawdopodobieństwa emisji ciągu obserwacji *O* dla modelu HMM M_k reprezentującego słowo W_k.

Celem budowy modelu akustycznego na potrzeby automatycznego rozpoznawania mowy jest taki dobór parametrów modelu, aby prawdopodobieństwo wyboru prawidłowej klasy M_k dla wypowiedzi O_k było większe niż prawdopodobieństwo wyboru klasy M_j dla j≠k. Trening spełniający to kryterium nazywany jest treningiem dyskryminującym (ang. *discriminative training*). Trening ten posiada tzw. własność dyskryminacji niepoprawnych hipotez.

Kryterium CMLE maksymalizuje prawdopodobieństwo warunkowe:

$$L_{CMLE}(\lambda) = \max_{\lambda} P_{\lambda}(M_k|O) \tag{6.2}$$

Stosując twierdzenie Bayes'a:

$$L_{CMLE}(\lambda) = \max_{\lambda} P_{\lambda}(M_k|O) = \frac{P_{\lambda}(O|M_k)P(M_k)}{P_{\lambda}(O)} = \frac{P_{\lambda}(O|M_k)P(M_k)}{\sum_{M\in L} P_{\lambda}(O|M)P(M)} \tag{6.3}$$

i zapisując mianownik jako sumę modelu poprawnego i modeli niepoprawnych dla obserwacji O otrzymujemy:

$$P_{\lambda}(M_k|O) = \frac{P_{\lambda}(O|M_k)P(M_k)}{P_{\lambda}(O|M_k)P(M_k) + \sum_{i\neq k} P_{\lambda}(O|M_i)P(M_i)} \tag{6.4}$$

Zatem aby zmaksymalizować $P_{\lambda}(M_k|O)$ musimy inkrementacyjnie zmieniać λ tak, aby zwiększać prawdopodobieństwo poprawnego modelu (wyrażenie w liczniku) i jednocześnie zmniejszać prawdopodobieństwo wyboru niepoprawnego modelu (suma w mianowniku). Zatem kryterium dyskryminujące minimalizuje błąd klasyfikacji.

Innymi słowy trening dyskryminujący [12] zmierza do takiego zamodelowania brzegów klas, aby poprawnie odróżniać klasy pomiędzy sobą. Kryterium to minimalizuje prawdopodobieństwo wyboru niepoprawnej klasy, jednocześnie maksymalizując prawdopodobieństwo wyboru poprawnej klasy, zamiast estymować dokładnie parametry lub rozkład prawdopodobieństwa cech dla klasy. Natomiast model budowany na podstawie kryterium największej wiarygodności konstruuje dokładny model statystyczny reprezentujący rozkład cech obiektów dla każdej klasy.

6.2 Wady algorytmu Baum'a-Welch'a

Algorytm Baum'a-Welch'a koncentruje się na dopasowaniu parametrów automatu HMM (dla każdego słowa mamy osobny model HMM) do danych treningowych, zamiast na wyróżnieniu (ang. *discriminate*) najlepszego modelu z pozostałych modeli.

Dodatkowo algorytm Baum'a-Welch'a wprowadza założenie co do kształtu rozkładu cech akustycznych dla fonemów. Najczęściej są to komponenty GMM – wielowymiarowego rozkładu normalnego. Często takie założenie jest zbyt daleko idące. Toteż na chwilę obecną rzadko stosuje się w wydajnych systemach ASR algorytm Baum'a-Welch'a + GMM. Pozostałe wady algorytmu Baum'-Welch'a to:

- Założenie, że wszystkie prawdopodobieństwa zależą tylko i wyłącznie od aktualnego stanu – co jest nieprawdą dla aplikacji rozpoznawania mowy.
- Mowa nie jest procesem stacjonarnym. Dlatego modele HMM nie potrafią dobrze zamodelować koartykulacji, gdzie rozkład prawdopodobieństwa mocno zależy od poprzednich stanów.
- Założenie, że sąsiednie obserwacje O są nieskorelowane – ponieważ HMM przetwarza jedynie jedna ramkę jednocześnie należy wektor cech rozszerzyć o informację kontekstową (np. delta pierwszego, drugiego rzędu pomiędzy sąsiednimi ramkami)
- Założenie, że poszczególne składowe wektora cech są niezależne.
- Założenie co do rozkładu prawdopodobieństwa emisji obserwacji – nie zawsze jest to prawidłowe podejście. Pomimo, że rozkład GMM może aproksymować dowolny rozkład prawdopodobieństwa – to nie wiemy ile należy użyć komponentów Gaussa, a więc liczba ta jest dobierana podczas procesu uczenia i należy ją traktować jako kolejną hipotezę.
- Kryterium MLE prowadzi do słabych właściwości dyskryminujących dla modelu akustycznego. Zastosowanie kryterium MMIE (ang. *maximum mutual information estimation*), maksymalizujące

informację wzajemną, poprawia dokładność klasyfikacji, jednak algorytm ten jest skomplikowany i trudny w implementacji.

6.3 Hybryda HMM-MLP

6.3.1 Wprowadzenie

Ponieważ klasyczne metody HMM opierają się na wielu niepoprawnych założeniach, które pociągają za sobą niezadowalającą skuteczność rozpoznawania mowy, dlatego badacze szybko znaleźli rozwiązanie wyżej wymienionych problemów stosując sieci neuronowe. Główne cechy hybrydy HMM-MLP to umiejętność nauki skomplikowanych analitycznie funkcji, wydajne generalizowanie, równoległość oraz trening dyskryminujący. Sieć neuronowa doskonale nadaje się do budowy modelu akustycznego, lecz słabo modeluje wymiar czasu oraz nie nadaje się do tworzenia kompozycji, dlatego powszechnie stosuje się w tym celu hybrydę HMM-MLP. Sieć neuronowa dokonuje klasyfikacji akustycznej (dla wektora cech wejściowych przypisuje prawdopodobieństwa *a posteriori* poszczególnych klas (fonemów)) natomiast HMM modeluje wymiar czasu [7]. Eksperymenty pokazują że hybryda NN-HMM posiada o wiele większą skuteczność niż klasyczne HMM-GMM.

6.3.2 Sieć neuronowa w systemach ASR

System połączeniowy (ang. connectionist system) składa się z połączonych ze sobą prostych węzłów (w przypadku SSN zwanych sztucznymi neuronami) realizujących elementarne operacje na sygnałach wejściowych. W sieci tej sygnał wyjściowy jest propagowany na wejścia węzłów kolejnej warstwy.

Cała informacja pozyskana z danych wejściowych jest reprezentowana przez strukturę i parametry węzłów sieci. A więc sieć ta modeluje złożone analitycznie zależności pomiędzy danymi wejściowymi i wyjściowymi bez jawnych założeń statystycznych (np. niezależność danych wejściowych, kształt rozkładu).

Aby obejść problemy przedstawione w punkcie 6.2 w systemach ASR powszechnie stosuje się hybrydę HMM – wielowarstwowy perceptron (MLP). Zastosowanie MLP w systemach ASR jest atrakcyjne z następujących powodów:

- sieć MLP doskonale nadaje się do uczenia
- trening dyskryminujący, który minimalizuje liczbę błędów
- potrafi w teorii wygenerować dowolną nieliniową funkcję danych wejściowych
- nie wymaga żadnych mocnych założeń co do rozkładu danych wejściowych
- łatwo można dołączyć informację kontekstową
- można łączyć różne wektory cech akustycznych (np. MFCC+PLP)

- zdolność adaptacyjna sieci

Niestety sieć MLP nie nadaje się do modelowania wzorców posiadających wymiar czasu. Sieć neuronowa może być tutaj wykorzystana jedynie do klasyfikacji elementarnych segmentów mowy takich jak fonemy, subfonemy itp.

Ponieważ HMM doskonale reprezentuje wymiar czasu oraz reprezentuje podejście hierarchiczne (zdania modelujemy za pomocą sekwencji słów, słowa za pomocą sekwencji fonemów, fonemy za pomocą sekwencji stanów procesu Markova) wskazane jest tutaj połączenie obu technik. Dlatego w [7] zaproponowano wykorzystanie hybrydy HMM-MLP, w której to sieć neuronowa dokonuje decyzji lokalnych, natomiast HMM dokonuje decyzji globalnych. Pomysł ten został rozwinięty w pracach [8, 9, 10, 11]. Ponieważ wiedza dziedzinowa na temat sygnału mowy jest zbyt uboga, aby dokonać właściwych założeń dlatego ilość wiedzy *a priori* ograniczamy do minimum – naszą jedyną wiedzą *a priori* jest algorytm ekstrakcji cech akustycznych MFCC, PLP

6.3.3 Wnioskowanie statystyczne za pomocą MLP

Celem nauki MLP jest skojarzenie wektora wejściowego z pożądanym wektorem wyjściowym. Jeżeli naszym zadaniem jest klasyfikacja N klas, wtedy sieć z N wyjściami powinna zostać użyta (po jednym wyjściu dla każdej klasy). Podczas nauki jedynka na wyjściu powinna oznaczać poprawną klasę, zero powinno zostać przyporządkowane do pozostałych klas. Nauczona sieć MLP wg powyższego schematu za pomocą algorytmu wstecznej propagacji estymuje prawdopodobieństwa *a posteriori* klas, tzn. po nauce typu 1-z-N wartości wyjść wielowarstwowego perceptronu dla danego wektora wejściowego x będą estymatorami prawdopodobieństwa *a posteriori* $P(c_i|x)$ dla klasy c_i skojarzonej z i-tym wyjściem. Powyższe twierdzenie można udowodnić dla różnorodnych błędów wykorzystywanych podczas algorytmu wstecznej propagacji [1][7].

6.3.4 Architektura hybrydy HMM-MLP

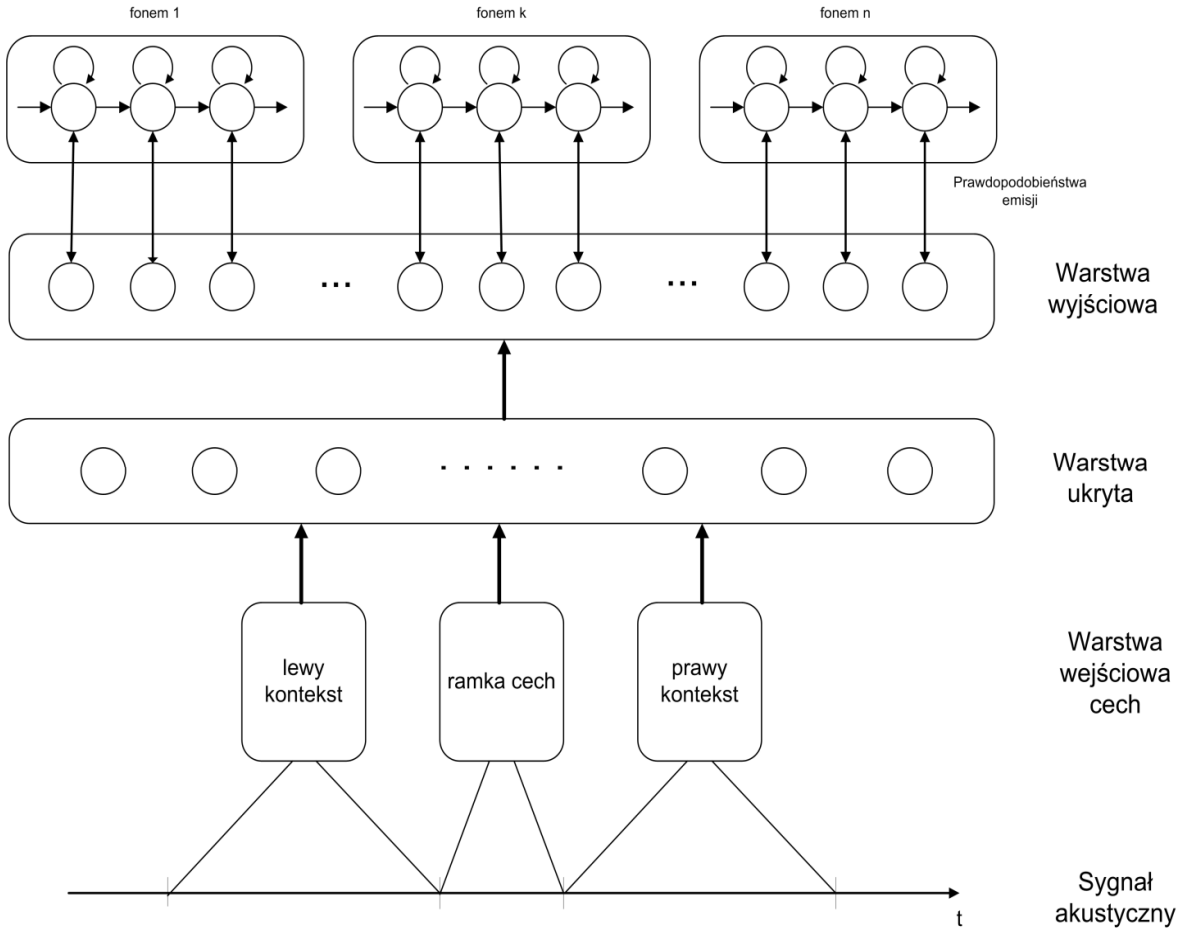
Ze względu na trening dyskryminujący hybryda HMM-MLP wymaga wyraźnie odseparowanych, niezależnych jednostek fonetycznych. Fonemy zależne od kontekstu (trifony, bifony) nie dają takiej poprawy błędu rozpoznawania jak w przypadku klasycznych systemów HMM, gdyż ich regiony decyzyjne nie są dobrze od siebie odseparowane. Toteż najlepsze rezultaty otrzymuje się dla jednostek fonetycznych posiadających niewielkie odległości akustyczne pomiędzy obiektami należącymi do tej samej klasy, a posiadającymi duże odległości dla obiektów należących do różnych klas (przykładem mogą być jednostki STU).

Hybryda HMM-MLP łączy zdolność rozpoznawania wzorców czasowych (za pomocą HMM) z klasyfikacją wzorców statycznych (za pomocą wielowarstwowego perceptronu). Słowa w takim systemie są modelowane za pomocą ukrytego modelu Markova (struktura modelu HMM jest liniowa, prawdopodobieństwa przejść pomijamy) do dekodowania wykorzystywany jest algorytm Viterbi'ego,

natomiast sieć neuronową delegujemy do wyznaczania prawdopodobieństw emisji.

W systemach hybrydowych wykorzystuje się dwa typy sieci neuronowej: o strukturze rekurencyjnej i sieć jednokierunkową. Struktura sieci jest zazwyczaj dobierana eksperymentalnie w zależności jakie części sygnału akustycznego modelujemy. Jeżeli modelujemy całe słowa to sieć rekurencyjna daje najlepsze rezultaty, natomiast gdy modelujemy fonemy to sieć typu jednokierunkowego jest wydajniejsza [8].

Na rysunku 6.1 przedstawiono topologię wielowarstwowego perceptronu wykorzystanego w systemie ASR [8] jako klasyfikator elementarnych jednostek akustycznych.



Rys. 6.1 – Topologia sieci MLP systemu przedstawionego w pracy [8].

Warstwa wejściowa sieci neuronowej została podzielona na trzy bloki, jeden dla ramki środkowej i po jednym dla lewego i prawego kontekstu. Każdy blok jest podzielony na sub-bloki każdy dla innego typu cech akustycznych [8] (ponieważ empirycznie udowodniono że taka struktura posiada o wiele większą skuteczność). Warstwa ukryta jest w pełni połączona z warstwą wyjściową, która

estymuje prawdopodobieństwa emisji stanów HMM (oczywiście po przekształceniu wg reguły Bayes'a). Każde wyjście wielowarstwowego perceptronu odpowiada jednemu stanowi ukrytemu automatu HMM.

6.3.5 Ekstrakcja cech akustycznych w hybrydzie HMM-MLP

Ekstrakcja cech akustycznych jest jedyną wiedzą *a priori* w architekturze hybrydowej HMM-MLP. Ponieważ wiedza dziedzinowa na temat akustyki mowy jest w dalszym ciągu niewystarczająca, aby dokonać ekstrakcji odpowiednich cech z sygnału akustycznego mowy, toteż odpowiedni algorytm ekstrakcji cech dopiera się eksperymentalnie. Ze względów historycznych większości współczesnych systemów ASR używa wektor cech MFCC (omówionego szczegółowo w rozdziale 5). Jednakże jak pokazują doświadczenia algorytm PLP lub RASTA-PLP [13] [14] daje znacznie lepsze rezultaty niż MFCC [15].

W pracach [15] [16] [17] przedstawiono propozycję znacznej redukcji błędu rozpoznania poprzez ekstrakcję kilku różnorodnych wektorów cech akustycznych z sygnału mowy i wprowadzenie każdego do warstwy wejściowej hybrydy HMM-MLP – każdemu wektorowi odpowiada oddzielny sub-blok warstwy wejściowej wielowarstwowego perceptronu.

Możliwość łączenia wielu wektorów cech akustycznych jest tutaj ogromnym atutem. W modelu akustycznym HMM-GMM zakładaliśmy, że składowe wektora obserwacji są nieskorelowane. Tym samym nie możemy w HMM-GMM połączyć cech z różnych algorytmów ekstrakcji.

6.3.6 Jednostki fonetyczne w HMM-MLP

Sieć neuronowa wykorzystywana jako klasyfikator wymaga wyraźnie odseparowanych, nienachodzących na siebie klas, dlatego należy użyć fonemów niezależnych od kontekstu (context independent phonems). W pracy [8] zaproponowano wykorzystanie jednostek fonetycznych Stationary-Transitional Units (STU) opisanych w [19]. Fonemy te można wykorzystać w architekturze z punktu 6.3.4. Jednostki te zostały podzielone na dwie części: jednostki stacjonarne i jednostki przejściowe. Pierwsze są stacjonarnymi częściami sygnałami akustycznego fonemów niezależnych od kontekstu, natomiast drugi typ jednostek to wszystkie dopuszczalne przejścia pomiędzy jednostkami stacjonarnymi. Jednostki *transistional* reprezentują koartykulację.

Na przykład transkrypcja fonetyczna w alfabecie IPA słowa marchew wygląda następująco [marxev], stosując jako jednostki fonetyczne STU słowo to zostanie zamodelowane za pomocą następujących fonemów:

<@-m> <m> <m-a> <a> <a-r> <r> <r-x> <x> <x-e> <e> <e-v> <v> <v-@>. Fonemy <m> <a> <r> <x> <e> <v> są jednostkami stacjonarnymi (@ - oznacza ciszę), natomiast <m-a> <a-r> <r-x> <x-e> <e-v> są jednostkami przejściowymi pomiędzy częściami stacjonarnymi.

6.3.7 Connectionist Viterbi Training

W klasycznym systemie ASR opartym o HMM/GMM podczas nauki estymujemy prawdopodobieństwo emisji obserwacji *O* dla modelu *M* wykorzystując metodę największej wiarygodności która uwzględnia tylko poprawną klasę. Natomiast w hybrydzie HMM-MLP estymujemy prawdopodobieństwo *a posteriori P(M|O)* uwzględniając podczas nauki wszystkie klasy (dyskryminując klasy niepoprawne) – a więc podczas nauki sieć neuronowa stara się jednoznacznie odseparować wszystkie klasy w przestrzeni R^d - gdzie *d* jest długością wektora cech akustycznych. Z tego powodu algorytm nauki modelu HMM-MLP jest zupełni inny niż klasyczny algorytm Baum'a-Welch'a.

Sieć neuronowa podejmuje decyzje lokalne natomiast algorytm Viterbiego podejmuje decyzję globalną. W pracy [7] przedstawiono dowód, że pomimo tego, że trening był dyskryminujący dla decyzji lokalnych to hybryda HMM- MLP również dyskryminuje globalnie, a więc na poziomie słów.

Algorytm uczenia takiego modelu łączy w sobie algorytm treningu Viterbi'ego oraz back-propagation. Standardowo procedura uczenia wygląda następująco: korzystając z początkowej segmentacji wykonanej za pomocą HMM-GMM (algorytm Baum'a-Welch'a), uczymy w pierwszej iteracji sieć neuronową wykorzystując tę segmentację, następnie wykonujemy ponowną segmentację wykorzystując nauczoną sieć neuronową – obydwa kroki powtarzamy tak długo, aż uzyskamy zbieżność. Jednakże trening taki trwa monstrualnie długo (dla przykładu 400 epok back-propagation dla małego zbioru treningowego – 200 nagrań – trwa kilkanaście godzin na procesorze AMD Turion64) w związku z czym trening jest niewykonalny na standardowych stacjach roboczych. Dlatego w pracy [8] autorzy zaproponowali zmodyfikowaną wersję algorytmu uczenia. Zamiast przeprowadzać re-segmentację po pełnym algorytmie back-propagation, iteracyjnie zmieniamy segmentacja w trakcie treningu na końcu każdej epoki. Technika ta znacznie redukuje czas treningu umożliwiając tym samym naukę na standardowych stacjach roboczych.

Procedura treningu wygląda następująco:

Inicjalizacja:

1) Normalizujemy dane treningowe

2) Wykonujemy pierwszą segmentację (ang. *bootstrap segmentation*) korzystając z modelu akustycznego HMM-GMM zbudowanego za pomocą algorytmu Baum-Welcha

3) Wagi sieci neuronowej inicjalizowane są małymi losowymi wartościami

Kolejne iteracje:

4) Ładujemy segmentacją danych treningowych dokonaną w poprzedniej epoce

5) Dla tej segmentacji łączącej wektor wejściowy z wektorem wyjściowym danych treningowych wykonujemy jedną iterację algorytmu back-propagation

6) Następnie za pomocą 'poduczonej' sieci neuronowej wyznaczamy nową segmentację zbioru treningowego

7) Aktualizujemy segmentację dla kolejnej iteracji wg następującego wzoru

$F(s_{old}, s_{new}) = \alpha s_{old} + (1 - \alpha)s_{new}$, gdzie

s_{old}- segmentacja z poprzedniej iteracji

s_{new}- segmentacja wykonana przez sieć neuronową z bieżącej iteracji

Funkcja błędu w algorytmie back-propagation to *cross entropy*. Wektor wyjściowy jest generowany na podstawie bieżącej segmentacji – 1.0 ustawiam dla aktywnego stanu, a dla pozostałych 0.0. Funkcja aktywacji ostatniej warstwy to Softmax, w warstwie ukrytej stosuję funkcję sigmoidalną. Warunkiem zakończenia nauki jest stabilizacja błędu.

CVT jest zatem algorytmem iteracyjnym, który polega na stopniowym przesuwaniu granic segmentacji aż do uzyskania segmentacji optymalnej.

Pseudo kod algorytmu wygląda następująco:

```
normalizeTrainingSet()
performBootstrapSegmentation(gmmAcousticModel, trainSet)
initRandomlyWeights()
alpha=1.0
epoch=0
do begin
  epoch++
  estimatePriors(trainSet)
  performBackPropagation(trainSet, nnet)
  resegment(trainSet, nnet, alpha)
  alpha = alpha - delta
until isConvergence
```

Listing 6.1 – Pseudokod algorytmu CVT

6.3.8 MLP jako estymator prawdopodobieństw emisji w HMM

Sieć MLP wyuczona wg schematu 1-z-N za pomocą algorytmu wstecznej propagacji estymuje prawdopodobieństwa *a posteriori* na wyjściach ostatniej warstwy. A więc na i-tym wyjściu ostatniej warstwy sieci neuronowej otrzymujemy prawdopodobieństwo $P(c_i|o)$, gdzie c_i – jest stanem ukrytym automatu HMM, natomiast o jest wektorem cech podawanym na wejściu MLP. Jednak prawdopodobieństwo emisji obserwowalnego zdarzenia ukrytego procesu Markova to prawdopodobieństwo warunkowe $P(o|c_i)$. Stosując regułę Bayes'a otrzymujemy:

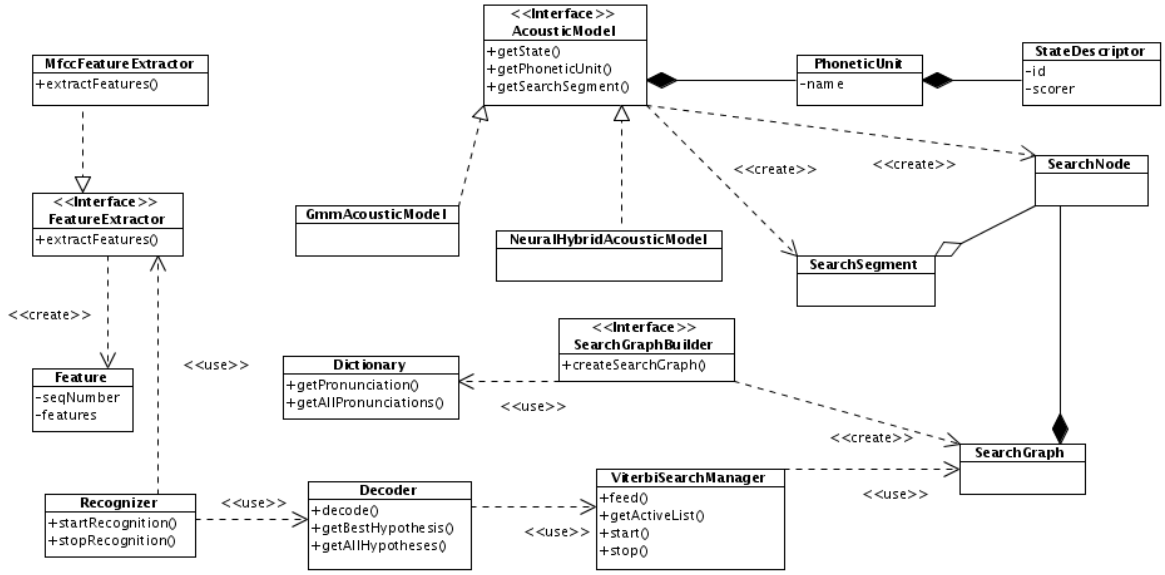
$$P(c_i|o) = \frac{P(o|c_i)p(c_i)}{P(o)}$$

$$\frac{P(c_i|o)}{P(c_i)} = \frac{P(o|c_i)}{P(o)}$$

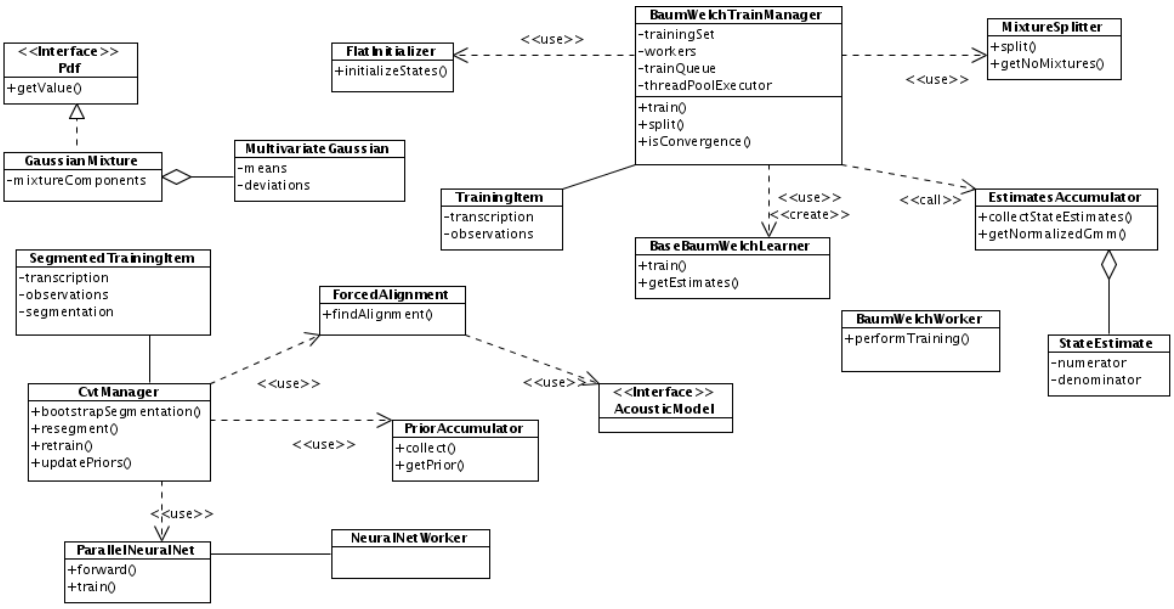
Zatem dzieląc każde wyjście ostatniej warstwy perceptronu przez częstość względną występowania klasy w danych treningowych, otrzymujemy przeskalowane prawdopodobieństwo emisji, które możemy wykorzystać podczas rozpoznawania.

7 Specyfikacja techniczna

7.1 Diagramy klas



Rys. 7.1 – Diagram klas odpowiedzialnych za dekodowanie sygnału mowy



Rys. 7.2 – Diagram klas odpowiedzialnych za trening

7.2 Karty CRC

FeatureExtractor	
Odpowiedzialność	*Współpracuje*
- interfejs ten specyfikuje operacje jakie powinien implementować komponent odpowiedzialny za ekstrakcję cech z sygnału akustycznego. - w pracy zrealizowałem adapter MFCC wykorzystujący algorytm zaimplementowany w projekcie Sphinx4.	- AudioPayload - Feature - FrontEnd - ConfigurationManager

Dictionary	
Odpowiedzialność	*Współpracuje*
- klasa ta odpowiedzialna jest za konwersję zapisu znakowego słowa na zapis fonetyczny.	

AcousticModel	
Odpowiedzialność	*Współpracuje*
- interfejs ten specyfikuje metody jakie powinna implementować każda klasa modelu akustycznego. Rolą modelu akustycznego jest zarządzanie dostępem do jednostek fonetycznych stosowanych przez model oraz dekomponowanie jednostek na stany HMM. Dodatkowo model odpowiedzialny jest za tworzenie segmentów grafu przeszukiwań dla jednostek fonetycznych oraz stanów.	- PhoneticUnit - StateDescriptor

GmmAcousticModel	
Odpowiedzialność	*Współpracuje*
- realizuje model akustyczny zbudowany w oparciu o funkcje gęstości prawdopodobieństwa Gaussa.	- GaussianMixture - LogScale - StateEstimates - SearchNode

	- SearchGraphSegment - HmmSearchNode

NeuralHybridAcousticModel	
Odpowiedzialność	*Współpracuje*
- model ten wykorzystuje sieć neuronową do estymacji prawdopodobieństw *a posteriori* dla kolejnych wektorów cech sygnału mowy	- NeuralNetwork - InputMapper - SearchNode - SearchGraphSegment - HmmSearchNode

SearchGraphBuilder	
Odpowiedzialność	*Współpracuje*
- interfejs ten odpowiedzialny jest za budowanie grafu poszukiwań wykorzystywanego przez dekoder podczas rozpoznawania mowy	- SearchGraphSegment - AcousticModel - Dictionary

PhoneticUnit	
Odpowiedzialność	*Współpracuje*
- reprezentuje jednostkę fonetyczną, z której składają się rozpoznawane słowa. Każda jednostka fonetyczna jest zrealizowana za pomocą jednego lub kilku stanów HMM.	- StateDescriptor

StateDescriptor	
Odpowiedzialność	*Współpracuje*
- reprezentuje stan HMM, który dekomponuje jednostki fonetyczne. Odpowiedzialnością klasy jest oszacowanie prawdopodobieństwa przynależności wektora wejściowego do klasy fonetycznej, którą reprezentuje stan.	- Pdf

SearchNode, SearchSegment, SearchGraph, SearchArc	
Odpowiedzialność	*Współpracuje*
- reprezentują składowe elementy grafu poszukiwań używanego podczas dekodowania	

Feature	
Odpowiedzialność	*Współpracuje*
- reprezentuje wektor cech sygnału mowy.	

ViterbiSearchManager	
Odpowiedzialność	*Współpracuje*
- klasa ta odpowiedzialna jest za implementacje algorytmu Viterbiego wykorzystywanego do poszukiwania najbardziej prawdopodobnej ścieżki w grafie poszukiwań. Klas ta posiada parametr *beam* – który jest promieniem zawężającym przestrzeń poszukiwań. - usuwa mało prawdopodobne ścieżki	- SearchGraphBuilder - Feature

Decoder	
Odpowiedzialność	*Współpracuje*
- zarządza procesem dekodowania - deleguje wywołania do SearchManagera - interpretuje wyniki poszukiwań.	- Feature - Result - SearchManager

Recognizer	
Odpowiedzialność	*Współpracuje*
- scala dekoder, feature extractor, search graph builder w całość, udostępniając prosty interfejs klasom wykorzystującym recognizer.	- FeatureExtractor - SearchGraphBuilder - Decoder

MultivariateGaussian	
Odpowiedzialność	*Współpracuje*
- klasa odpowiedzialna za wyliczanie wartości funkcji gęstości prawdopodobieństwa wielowymiarowego rozkładu Gaussa w skali logarytmicznej. Kapsułkuje parametry rozkładu.	- Feature - LogScale

GaussianMixture	
Odpowiedzialność	*Współpracuje*
- klasa ta kapsułkuje algorytm wyliczania wartości funkcji gęstości prawdopodobieństwa dla rozkładu składającego się z wielu komponentów Gaussa w skali logarytmicznej.	- Feature - LogScale - MultivariateGaussian

BaumWelchLearner	
Odpowiedzialność	*Współpracuje*
- implementuje algorytm Baum-Welch'a dla jednego modelu HMM. Jest to wersja jednowątkowa zoptymalizowana pod kątem rozpoznawania mowy. - szacuje parametry GMM oraz prawdopodobieństwa przejść HMM.	- GaussianMixture - TrainingItem - LogScale - StateEstimates - Hmm - PhoneticUnit

EstimatesAccumulator	
Odpowiedzialność	*Współpracuje*
- akumuluje estymatory parametrów modelu HMM-GMM. - estymatory są składowane w postaci licznik+mianownik. - wylicza zakumulowane wartości poszczególnych parametrów. - synchronizuje dostęp	- ComponentEstimates - StateEstimates - GaussianMixture - LogScale - MultivariateGaussian

FlatInitializer	
Odpowiedzialność	*Współpracuje*
- kapsułkuje algorytm inicjalizacji parametrów GMM dla poszczególnych stanów wartościami średnimi wyliczonymi ze zbioru treningowego.	- GmmAcousticModel - TrainSentence

MixtureSplitter	
Odpowiedzialność	*Współpracuje*
- odpowiedzialny jest za 'rozdwojenie' pojedynczego komponentu Gaussa na dwie składowe.	- GmmAcousticModel - MultivariateGaussian - GaussianMixture - StateDescriptor

TrainingItem	
Odpowiedzialność	*Współpracuje*
- reprezentuje próbkę uczącą. Składa się z transkrypcji oraz ciągu wektora cech sygnału akustycznego.	- PhoneticUnit - Feature

BaumWelchTrainManager	
Odpowiedzialność	*Współpracuje*
- zarządza procesem uczenia typu embedded training. - implementuje równoległy algorytm uczenia Baum'a-Welch'a	- GmmAcousticModel - ModelInitializer - MixtureSplitter - TrainingItem - BaumWelchWorker

BaumWelchWorker	
Odpowiedzialność	*Współpracuje*
- podstawowy element wykorzystywany w równoległym algorytmie Baum-Welcha. Działa w osobnym wątku. Odpowiedzialny jest za naukę próbek uczących pobieranych z kolejki.	- BaumWelchLearner - StateEstimates - GmmAcousticModel - TrainingItem

ForcedAlignment	
Odpowiedzialność	*Współpracuje*
- odpowiedzialny jest za segmentację wektora cech sygnału akustycznego. Segmentacja przeprowadzana jest za pomocą algorytmu Viterbiego dla grafu zbudowanego dla pojedynczego słowa.	- SentenceSearchGraphBuilder - ViterbiSearchManager - Decoder - AcousticModel

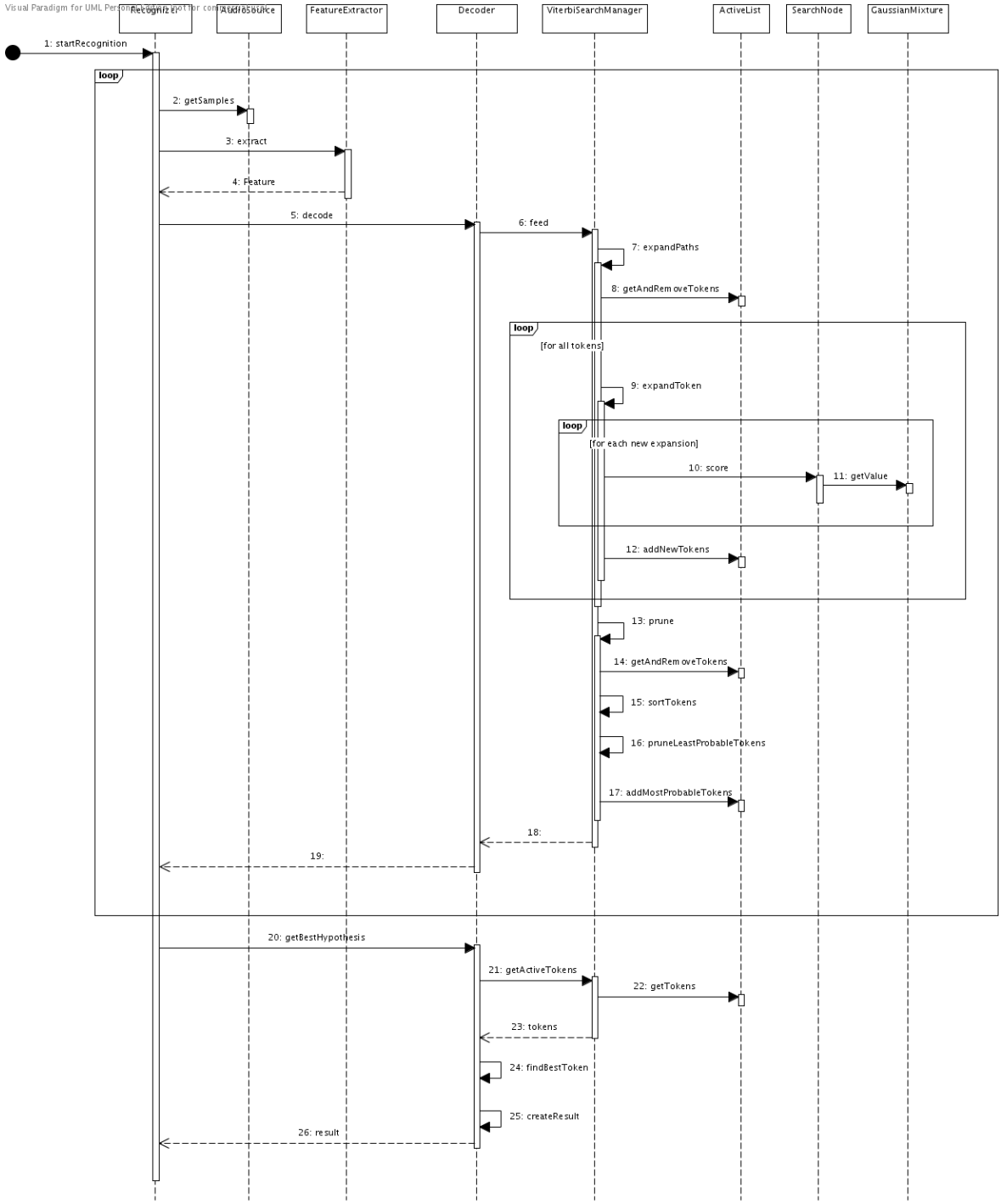
SegmentedTrainingItem	
Odpowiedzialność	*Współpracuje*
- reprezentuje próbkę uczącą. Zawiera ‘posegmentowany’ ciąg wektora cech.	- Feature - NeuralStateDescriptor - StateSegment

PriorAccumulator	
Odpowiedzialność	*Współpracuje*
- odpowiedzialny jest za oszacowanie prawdopodobieństw *a priori* klas na podstawie zbioru treningowego.	

ParallelNeuralNet	
Odpowiedzialność	*Współpracuje*
- kapsułkuje równoległy algorytm uczenia sieci neuronowej.	- NeuralNetWorker - NeuralNet - DeltaAccumulator

CvtManager	
Odpowiedzialność	*Współpracuje*
- implementuje algorytm connectionist Viterbi training. Zarządza procesem uczenia oraz re-segmentacją.	- NeuralNetwork - NeuralHybridAcousticModel - SegmentedTrainingItem - PriorAccumulator

7.3 Scenariusz dekodowania



Rys. 7.3 – Diagram sekwencji procesu dekodowania

Na rysunku 7.3 przedstawiono interakcje pomiędzy klasami uczestniczącymi w procesie dekodowania sygnału mowy. Diagram sekwencji dekodowania opisuje poniższy scenariusz:

1) Launcher wywołuje metodę startRecognition klasy Recognizer, tym samym inicjując proces rozpoznawania mowy

2) Recognizer w pętli pobiera kolejne próbki sygnału akustycznego z klasy implementującej interfejs AudioSource (np. z pliku lub mikrofonu)

3) Recognizer wywołuje metodę extract klasy realizującej interfejs FeatureExtractor. Parametrem wywołania jest ciąg próbek sygnału akustycznego.

4) Klasa FeastureExtractor dokonuje ekstrakcji cech akustycznych z sygnału mowy i zwraca wektor cech opakowany w obiekt Feature.

5) Recognizer wywołuje metodę decode klasy Decoder. Argumentem wywołania jest obiekt Feature.

6) Decoder deleguje przetwarzanie obiektu Feature do klasy ViterbiSearchManager wywołując metodę feed.

7) ViterbiSearchManager rozpoczyna przechodzenie grafu przeszukiwań poprzez przedłużanie ścieżek węzłów z poprzedniej iteracji. Węzły grafu są opakowane w obiekty Token.

8) W tym celu manager pobiera wszystkie Tokeny z ActiveList'y.

9) W pętli dla każdego token'u manager przedłuża ścieżkę, której koniec reprezentowany jest przez przetwarzany token. Przedłużanie ścieżki odbywa się poprzez dołączenie do obecnej ścieżki sąsiedniego węzła.

10) Dla nowego węzła obliczane jest prawdopodobieństwo emisji obecnego wektor cech Feature.

11) Węzeł deleguje obliczanie prawdopodobieństwa do obiektu GaussianMixture

12) Nowo utworzone ścieżki są dodawane do ActiveList'y

13) Manager rozpoczyna usuwanie najmniej prawdopodobnych ścieżek wywołując metodę prune.

14) Pruner pobiera wszystkie tokeny z ActiveList'y

15) Ścieżki są sortowane wg prawdopodobieństwa

16) Ścieżki najmniej prawdopodobne są usuwane

17) Najbardziej prawdopodobne ścieżki są zapisywane w ActiveList.

18) Koniec przetwarzania w ViterbiSearchManager

19) Koniec przetwarzania w dekoderze.

Recognizer pobiera kolejne próbki sygnału akustycznego i iteracyjnie powtarzane są kroki 2-19.

20) Recognizer pobiera rezultaty dekodowania wywołując metodę getBestHypothesis

21) Dekoder pobiera aktywne tokeny z ViterbiSearchManager'a

22) ViterbiSearchManager deleguje wywołanie do ActiveList

24) Dekoder wybiera token (ścieżkę) z najwyższym prawdopodobieństwem

25) Na podstawie tego tokenu tworzy obiekt Result

26) Rezultat jest przekazywany do Recognizer'a

7.4 Równoległy embedded training

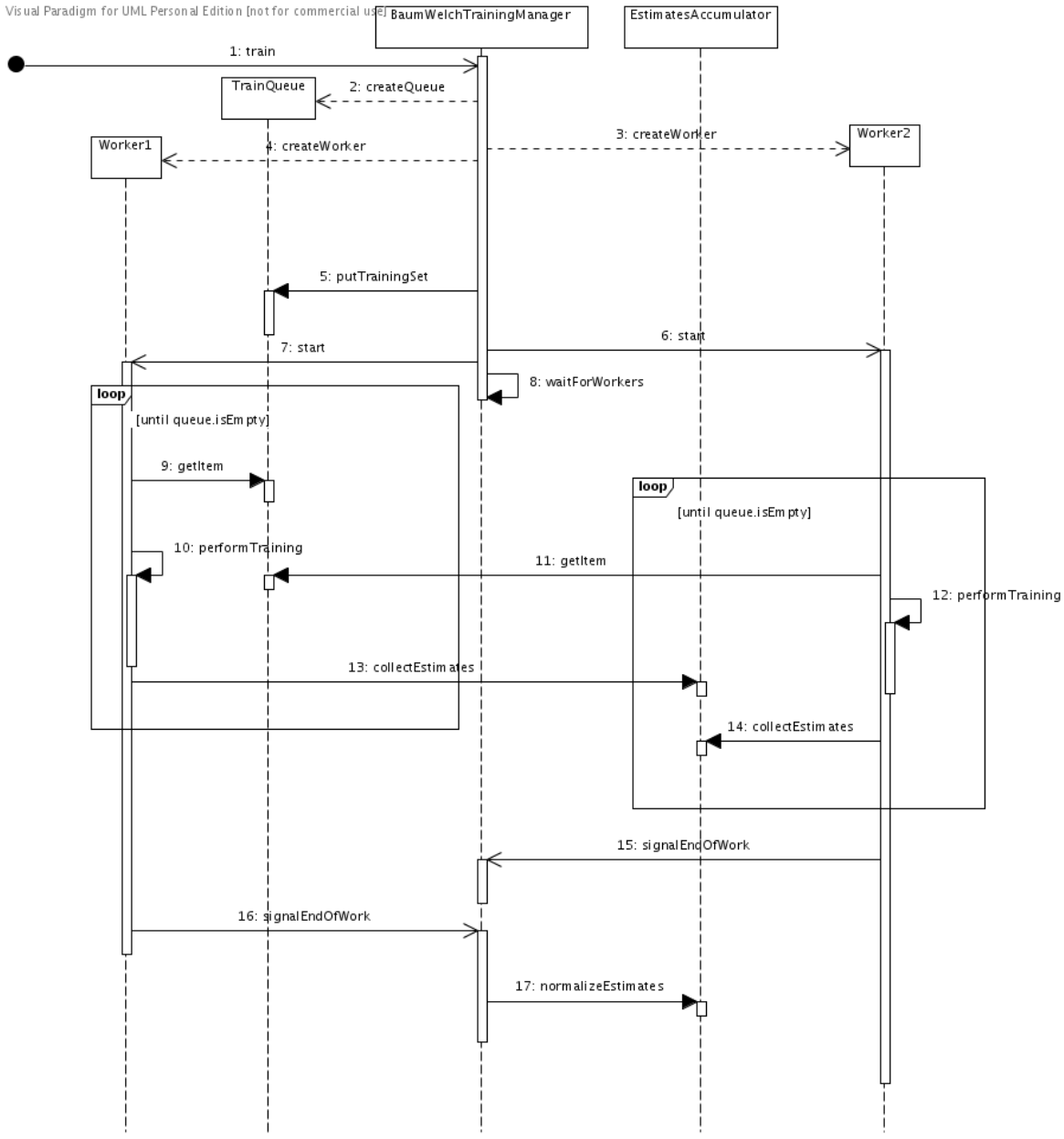
Zaimplementowałem w ramach pracy wersję algorytmu Baum-Welch'a dostosowaną do rozpoznawania mowy. Algorytm przeprowadza embedded training. Jest to tryb nauki stosowany w systemach ASR, gdy nie posiadamy ręcznej segmentacji danych treningowych. Znalezienie granic fonemów z sygnału akustycznego jest nawet dla człowieka niezmiernie trudnym i pracochłonnym zadaniem. Ponieważ modelujemy każdy fonem jako ukryty model Markova, taka ręcznie wykonana segmentacja zwiększa dokładność modelu i przyśpiesza algorytm uczenia. Aby poradzić sobie z tym problemem łączymy modele HMM dla każdego fonemu i dla tak skonstruowanego łańcucha przeprowadzamy trening. Segmentacja fonemów w sygnale akustycznym jest przeprowadzana jako część algorytmu uczenia. Podczas estymacji nowych parametrów modelu zachowuję osobno licznik i mianownik, który trafia do akumulatora. Po oszacowaniu nowych wartości parametrów dla każdego zdania ze zbioru treningowego, następuje faza normalizacji zebranych estymatorów częściowych i przypisanie nowych wartości parametrów dla każdego stanu.

Zaimplementowany algorytm estymuje wartości parametrów rozkładów dla ciągłej wersji ukrytego modelu Markowa. Oznacza to, że prawdopodobieństwa emisji stanów są opisane za pomocą funkcji gęstości prawdopodobieństwa. Ponieważ skończoną sumą rozkładów Gaussa można aproksymować dowolny rozkład, algorytm estymuje parametry wielokomponentowego rozkładu Gaussa. Każdy stan jest inicjalizowany wartościami średnimi wyznaczonymi na podstawie zbioru uczącego. Zaczynam więc algorytm uczenia z jednym komponentem Gaussa dla każdego stanu. Po wykonaniu 4-5 iteracji wykonuję operację *split* – polegającą na rozdwojeniu każdego komponentu. W następnych iteracjach estymuję parametry rozkładów Gaussa dla dwa razy większej liczby komponentów. Ponownie po 4-5 iteracjach wykonuję '*split*' komponentów, aż do momentu uzyskania żądanej liczby komponentów.

Dla dużych korpusów językowych pojedyncza iteracja algorytmu Baum'a-Welch'a może trwać kilka godzin, dlatego postanowiłem po szeregu kroków optymalizacyjnych zaimplementować wersję równoległą algorytmu. Ponieważ w każdej iteracji *embedded training*'u podstawowa część algorytmu Baum'a-Welch'a jest wykonywana dla innego modelu HMM postanowiłem ten fragment zrównoleglić. Wszystkie zdania ze zbioru treningowego są umieszczane w kolejce. Następnie tworzę tyle wątków ile znajduje się procesorów w systemie. Każdy wątek pobiera zdanie z kolejki i przeprowadza pełny algorytm Baum-Welch'a dla tego zdania. Wyliczone estymatory są zachowywane w akumulatorze – do którego dostęp jest od teraz synchronizowany. Gdy kolejka jest pusta każdy wątek

informuje menadżer o końcu pracy. Menadżer przeprowadza normalizację estymatorów sekwencyjnie. Następnie zdania są ponownie wrzucane do kolejki i przeprowadzana jest kolejna iteracja równoległego algorytmu *embedded training*'u.

Przypadek dwuprocesorowy został schematycznie przedstawiony na poniższym diagramie sekwencji.



Rys. 7.4 – Diagram sekwencji równoległego embedded training'u

Diagram sekwencji z rysunku 7.4 opisuje poniższy scenariusz:

1) Launcher wywołuje metodę train klasy BaumWelchTrainManager, tym samym inicjując proces budowy modelu akustycznego

2) Manager tworzy kolejkę współdzieloną przez worker'y do której trafiają elementy zbioru uczącego.

3,4) Manager tworzy worker'y odpowiedzialne za wykonywanie algorytmu Baum'a-Welch'a w oddzielnych wątkach.

5) Wszystkie elementy zbioru treningowe umieszczane są w kolejce.

6,7) Manager wysyła asynchroniczny sygnał start do wszystkich worker'ow. Sygnał wyzwala w worker'ach start procedury nauki.

8) Manager usypia swój wątek.

9,11) W pętli obydwa worker'y pobierają element z kolejki zawierającej próbki uczące.

10,12) Każdy worker dla swoich danych przeprowadza trening Baum'a-Welch'a

13,14) Oszacowane wartości parametrów automatu HMM są zachowywane w akumulatorze

15) Worker2 sygnalizuje koniec pracy, ponieważ kolejka jest pusta.

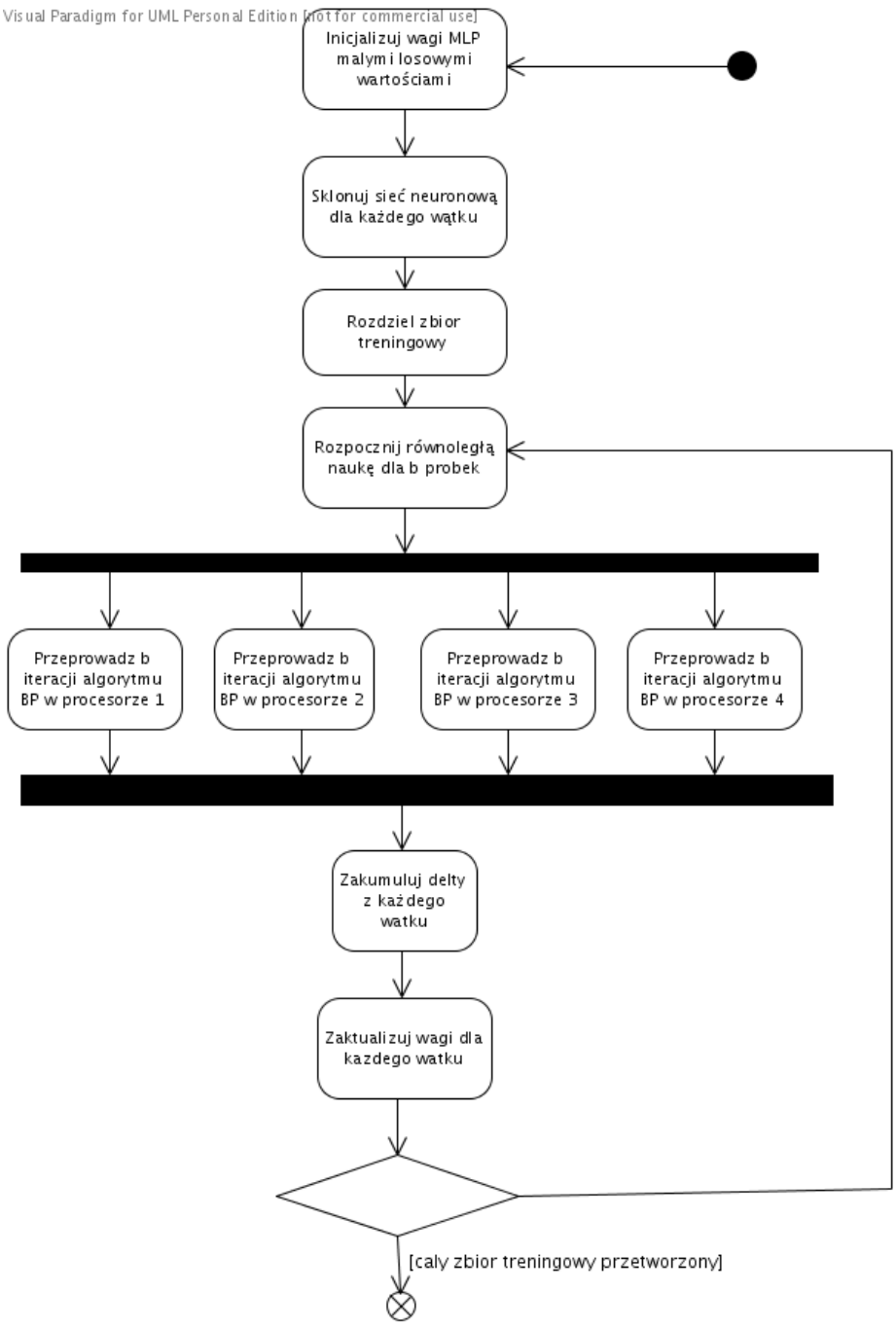
16) Worker1 sygnalizuje koniec pracy, tym samym 'budząc' manager.

17) Manager na podstawie zgromadzonych estymatorów parametrów HMM oblicza zakumulowane wartości tych parametrów.

7.5 Równoległy back-propagation

Pomimo niedużego korpusu językowego AN4, który wykorzystuję do budowy modelu akustycznego, nauka sieci neuronowej trwa dość długo. Dlatego postanowiłem zaimplementować równoległą wersję algorytmu back-propagation, aby wykorzystać w pełni moc obliczeniową platformy wieloprocesorowej.

Jedynym sposobem zrównoleglenia algorytmu back-propagation jest rozdzielenie zbioru treningowego na mniejsze równoliczne podzbiory, 'sklonowanie' sieci neuronowej dla każdego procesora i iteracyjne wyliczanie delt wag w każdym procesorze oddzielnie dla innego zbioru treningowego. W trybie *online* algorytmu back-propagation wagi sieci są aktualizowane w każdej iteracji (po wykonaniu kroku *forward* i *backward* dla każdej próbki treningowej), a więc należałoby synchronizować delty wag między procesorami dla każdej iteracji tak, aby każdy procesor przeprowadzał naukę dla tego samego zestawu wag sieci. Jest to rozwiązanie nie do zakceptowania, gdyż częsta synchronizacja między wątkami jest tutaj wąskim gardłem dramatycznie redukującym skalowalność algorytmu. Toteż postanowiłem zrównoleglić algorytm back-propagation działający w trybie *batch*. Rozmiar *batch* ustalany jest na taką wartość, aby wątki nie synchronizowały się między sobą zbyt często oraz by nauka nie była zbyt wolna. Schematycznie implementację parallel back-propagation przedstawia diagram czynności umieszczony na rysunku 7.5.



Rys. 7.5 – Równoległy batch back-propagation

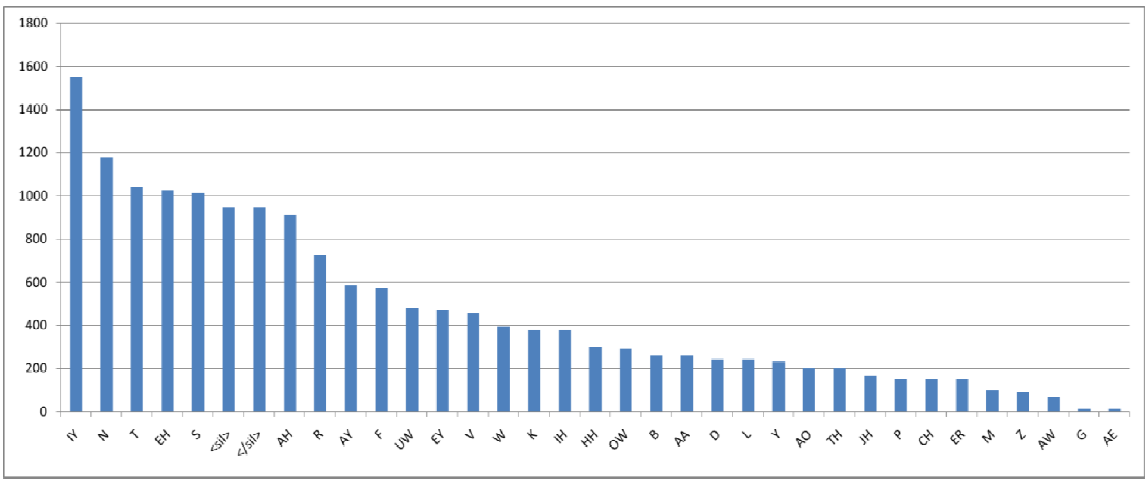
8 Budowa modelu akustycznego

8.1 Korpus mowy

Jedynym darmowym korpusem językowym dostępnym w sieci jest korpus AN4 nagrany na Carnegie Mellon University. Korpus składa się z 948 zdań (plus 130 zdań testowych) wypowiedzianych przez 53 mężczyzn i 21 kobiet. Całkowita długość zbioru treningowego wynosi 50 minut, natomiast testowego ok. 6 minut. Obydwa zbiory są rozdzielone (tzn. nagrania mówców ze zbioru testowego nie znajdują się w zbiorze treningowym) – dzięki temu można przetestować stopień generalizacji modelu akustycznego. Nagrania składają się jedynie z imion, cyfr, dat. Imiona i nazwy własne są przeliterowane. Ostatecznie słownik dla obu zbiorów składa się z 130 słów. Nagrania są wykonane w środowisku bezszumowym. Podczas testów wykorzystałem wersję próbkowaną z częstotliwością 8kHz. Format kodowania to LINEAR PCM.

Transkrypcje nagrań są zapisane przy użyciu grafemów, w trakcie treningu grafemy konwertuję na zapis fonetyczny ARPABET.

Poniżej znajduje histogram fonemów opracowany na podstawie transkrypcji ze zbioru treningowego.



Rys. 8.1 – Histogram fonemów AN4

Można zauważyć, że korpus jest mocno zdegenerowany; fonem IY pojawia się 1,5 tyś. razy w danych treningowych, natomiast fonem AE posiada jedynie 12 wystąpień. Tak ogromne dysproporcje w reprezentacji poszczególnych klas mają znaczący wpływ na budowę modelu akustycznego – o czym piszę w następnych punktach.

8.2 Parametryzacja procesu budowy modelu

Zaimplementowany w ramach pracy algorytm budowy modelu akustycznego można sparametryzować za pomocą plików konfiguracyjnych. W niniejszym podpunkcie omawiam znaczenie poszczególnych parametrów.

8.2.1 Parametry modelu HMM-GMM

asr.noCycles	Określa maksymalną liczbę iteracji algorytmu Baum'a-Welch'a przeprowadzaną dla stałej liczby komponentów GMM.
asr.deviationFloor	Definiuje minimalną wartość odchylenia standardowego komponentów GMM. W przypadku zbyt małej reprezentacji próbek fonemu wartość odchylenia często wynosi 0.
asr.noStatesPerPhoneme	Ustala liczbę stanów realizujących fonem.
asr.minRelativeErrorChange	Określa względną zmianę błędu podczas iteracji algorytmu Baum'a-Welch'a poniżej której kończymy wykonywanie algorytmu.
asr.logBase	Podstawa logarytmu dla skali logarytmicznej.
asr.dump	Określa czy robić zrzut wyników częściowych na dysk.
asr.work.dir	Katalog roboczy.
asr.an4.samplingRate	Częstotliwość próbkowania cyfrowego sygnału akustycznego.
asr.an4.noMixtures	Pożądana liczba komponentów Gaussa.
asr.an4.transcription.file	Ścieżka pliku z transkrypcjami.
asr.an4.dictionary.file	Ścieżka pliku słownika fonetycznego.
asr.an4.phoneSet.file	Ścieżka pliku zawierającego zbiór wszystkich fonemów języka.
asr.an4.feature.dir	Katalog z plikami zawierającymi ciągi cech dla nagrań zbioru treningowego.
asr.an4.am.file	Ścieżka pliku, w którym zostanie zachowany zbudowany model akustyczny.

8.3 Scenariusz budowy modelu akustycznego HMM-GMM

1) Konwertujemy nagrania korpusu AN4 na ciągi wektorów cech MFCC. W tym celu uruchamiamy WaveConverter, która deleguje obliczenia do metody MfccConverter.computeMfcc(). Parametrami klasy WaveConverter są plik z transkrypcjami i katalog w którym znajduje się korpus. Wyznaczone ciągi cech są serializowane do plików.

2) Następnie tworzymy plik konfiguracyjny bw-pho.properties zawierający konfigurację algorytmu budowy modelu akustycznego HMM-GMM dla fonemów. Poniżej znajduje się zawartość tego pliku dla której osiągnięto najlepsze rezultaty:

```
asr.noCycles=4
asr.deviationFloor=1E-12
asr.noStatesPerPhoneme=3
asr.minRelativeErrorChange=1E-4
asr.logBase=1.0001
asr.dump=true
asr.work.dir=/mnt/work/an4/work
asr.an4.samplingRate=8000
asr.an4.noMixtures=16
asr.an4.transcription.file=/mnt/work/an4/train.txt
asr.an4.dictionary.file=/mnt/work/an4/an4.dic
asr.an4.phoneSet.file=/mnt/work/an4/an4.phone2
asr.an4.feature.dir=/mnt/work/an4/an4-8/feat
asr.an4.am.file=/mnt/work/an4/8khz-phonemes.xml
```

3) Fragment pliku z transkrypcjami:

```
an4_clstk/fash/an251-fash-b  YES
an4_clstk/fash/an253-fash-b  GO
an4_clstk/fash/an254-fash-b  YES
an4_clstk/fash/an255-fash-b  U M N Y H SIX
…
```

Fragment pliku słownika:

```
A                    AH
A(2)                 EY
AND                  AE N D
AND(2)               AH N D
APOSTROPHE           AH P AA S T R AH F IY
APRIL                EY P R AH L
```

```
AREA                    EH R IY AH
AUGUST                  AA G AH S T
…
```

Obydwa pliki są parsowane i budowany jest obiektowy model słownika (klasa Dictionary) oraz zbioru treningowego.

4) Klasa uruchomieniowa (bootstrap) wywołuje metodę train klasy BaumWelchTrainManager, która steruje algorytmem równoległej nauki. Dla każdego procesora tworzony jest osobny Worker, który wykonuje algorytm Baum'a-Welch'a.

5) Ponieważ składowe wektora MFCC przyjmują wartości ujemne, musimy je skonwertować na przedział (0,1). Implementacja algorytmu Baum'a-Welch'a przeprowadza obliczenia w skali logarytmicznej, a wartośc logarytmu dla wartości ujemnej jest niezdefiniowana.

6) Następnie rozpoczyna się proces nauki parametrów HMM. Automat HMM reprezentuje pojedynczy fonem (zdekomponowany na 3 stany ukryte).

7) Ponieważ parametr `asr.noCycles=4` toteż co 4 iteracje następuje rozdwojenie komponentów GMM wg schematu z punktu 4.4.3.

8) Po każdej iteracji embedded traing'u wyniki częściowe są zapisywane do pliku

9) Na koniec manager zapisuje oszacowane parametry automatów HMM do pliku xml. Oto fragment pliku modelu akustycznego dla 4 komponentów Gaussa (ze względu na przejrzystość zostały umieszczone jedynie 2 pierwsze wymiary MFCC).

```
<?xml version="1.0" encoding="UTF-8"?>

<acousticModel type="GMM_HMM">
  <noMixtures>2</noMixtures>
  <noDimensions>39</noDimensions>
  <phoneticUnit name="D">
    <state id="D_1" index="0">
      <selfLoopProbability>0.5000000000000001</selfLoopProbability>
      <gmm>
        <mixtureComponent>
          <weight>0.33930505689164303</weight>
          <dimension index="0" mean="0.690441261308893"
                                          deviation="0.05520399642630017"/>
          <dimension index="1" mean="0.5020033063434427"
                                          deviation="0.011600632962998512"/>
               </mixtureComponent>
        <mixtureComponent>
          <weight>0.6606949431078968</weight>
          <dimension index="0" mean="0.6173401537127431"
                                          deviation="0.026816966637834155"/>
          <dimension index="1" mean="0.504379503803045"
                                          deviation="0.006176261759472653"/>
        </mixtureComponent>
      </gmm>
    </state>
    <state id="D_2" index="1">
      <selfLoopProbability>0.5000000000000001</selfLoopProbability>
      <gmm>
        <mixtureComponent>
          <weight>0.5267428241730372</weight>
```

```
          <dimension index="0" mean="0.6220368564560923"
                                          deviation="0.03758008301423541"/>
          <dimension index="1" mean="0.5038015711917568"
                                         deviation="0.007031179027045366"/>
        </mixtureComponent>
        <mixtureComponent>
          <weight>0.4732571758264746</weight>
          <dimension index="0" mean="0.6584155346550271"
                                           deviation="0.0575577589064946"/>
          <dimension index="1" mean="0.5046822923551738"
                                         deviation="0.010006953671438452"/>
        </mixtureComponent>
      </gmm>
    </state>
    <state id="D_3" index="2">
      <selfLoopProbability>0.5000000000000001</selfLoopProbability>
      <gmm>
        <mixtureComponent>
          <weight>0.5684950444179133</weight>
          <dimension index="0" mean="0.757686773746113"
                                          deviation="0.04554362920092077"/>
          <dimension index="1" mean="0.4862343601076161"
                                          deviation="0.007964851138641 28"/>
        </mixtureComponent>
        <mixtureComponent>
          <weight>0.43150495558164037</weight>
          <dimension index="0" mean="0.7229033022495219"
                                          deviation="0.05397686468034135"/>
          <dimension index="1" mean="0.4891380939003 1315"
                                         deviation="0.014548172305532289"/>
        </mixtureComponent>
      </gmm>
    </state>
  </phoneticUnit>
```

9 Rezultaty

9.1 Środowisko testowe

Ze względu na długi czas nauki budowę modelu akustycznego postanowiłem przeprowadzić na 8 procesorowej instancji High-CPU Extra Large Instance dostępnej w ramach usługi Amazon EC2. Instancja ta posiada 64-bitową architekturę AMD64, do wykorzystania jest 7GB pamięci RAM. Moc obliczeniowa jednego rdzenia to 2.5 razy moc procesora 1.0-1.2 GHz 2007 Opteron. Testy przeprowadziłem na 64 bitowym systemie Ubuntu Linux, implementacja maszyny wirtualnej Javy to 64 bitowa wersja Sun'owska.

9.2 Jednostki fonetyczne

Testowałem dwa typy jednostek: fonemy i stationary-transitional acoustic units. Fonemy zostały zdekomponowane na 3 stany HMM, STU składają się z jednego stanu.

9.3 HMM-GMM

W rozdziale 8 wspomniałem, że korpus językowy AN4 nie jest poprawnie zrównoważony. W danych treningowych niektóre fonemy są reprezentowane zbyt licznie z kolei niektóre zbyt słabo. W efekcie przy dużej liczbie komponentów Gaussa wariancja dla dodatkowych komponentów wynosi 0. Mamy więcej komponentów Gaussa niż próbek uczących i w rezultacie parametry modelu dokładnie dostosowują się do danych treningowych. Dodatkowo dla bardzo małych wariancji wartość funkcji gęstości prawdopodobieństwa jest sporo większa niż zakres liczb zmiennoprzecinkowych podwójnej precyzji. Dlatego postanowiłem ustalić minimalną wartość wariancji, aby poprawnie wykonać wszystkie iteracje algorytmu Baum-Welch'a.

We wszystkich testach zastosowałem wektor cech akustycznych MFCC wyznaczany dla 25ms ramek, przesuwanych co 10ms. Wszystkie nagrania przed wyliczaniem MFCC zostały przefiltrowane za pomocą filtra pasmowo przepustowego 200Hz-3500Hz. Liczba kanałów w skali mel'owej została ustalona na 31, a transformata Fouriera liczona była dla 256 próbek. Ponieważ podczas wyliczania średniej i wariancji rozkładów Gaussa korzystam ze skali

logarytmicznej, a składowe wektora MFCC przyjmują wartości ujemne, toteż wartości zostały przeskalowane na przedział (0,1).

Testy skuteczności rozpoznawania mowy zostały przeprowadzone dla grafu zbudowanego ze wszystkich zdań zbioru testowego. Tylko w całości poprawnie rozpoznane zdanie było klasyfikowane jako rozpoznane. Dokładność rozpoznania rozumiem jako liczbę poprawnie rozpoznanych zdań do liczby wszystkich zdań testowych. Algorytm wykorzystany przy dekodowaniu to alg. Viterbi'ego rozszerzony o parametr *beam=r* będący promieniem zawężającym przestrzeń poszukiwań tylko do *r* najbardziej prawdopodobnych ścieżek.

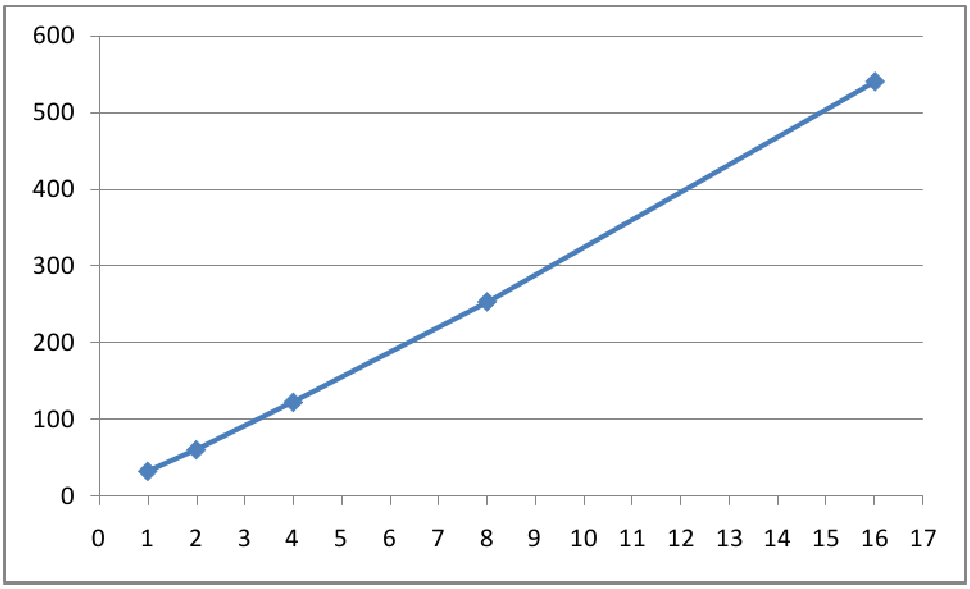
9.3.1 Fonemy

Całkowita liczba fonemów = 35, jeden fonem składa się z 3 stanów HMM.

9.3.1.1 Czas nauki

W pierwszej kolejności przeprowadziłem testy czasu trwania treningu w zależności od liczby komponentów Gaussa. Uzyskane wyniki potwierdzają że czas ten jest liniowo zależny od liczby komponentów.

Liczba komponentów	Czas jednej iteracji [sek]
1	32
2	60
4	122
8	253
16	540

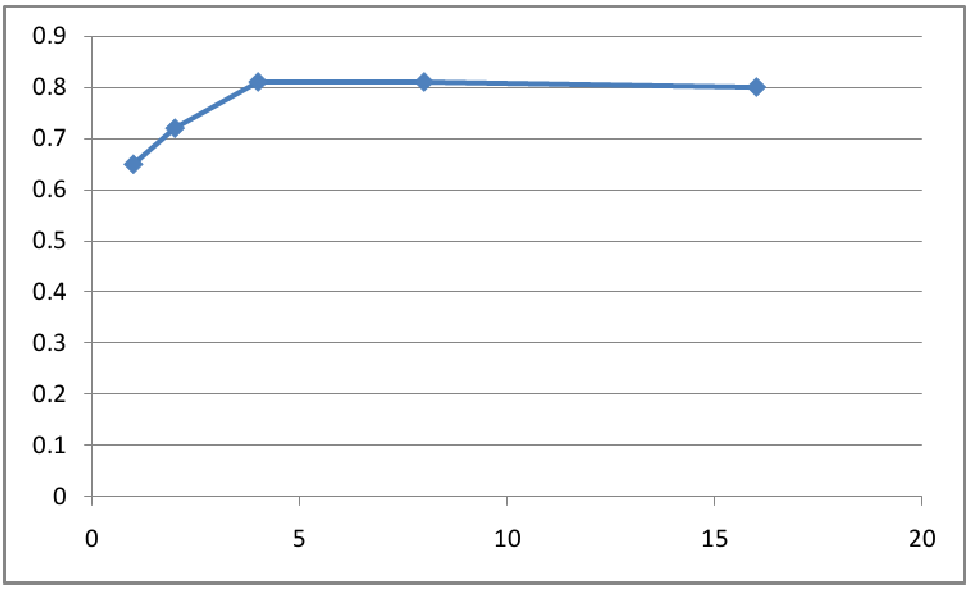


Rys. 9.1 – Zależność czasu nauki od liczby komponentów GMM

9.3.1.2 Skuteczność rozpoznania

Beam = 100

Liczba komponentów	Dokładność rozpoznania
1	65%
2	72%
4	81%
8	81%
16	80%



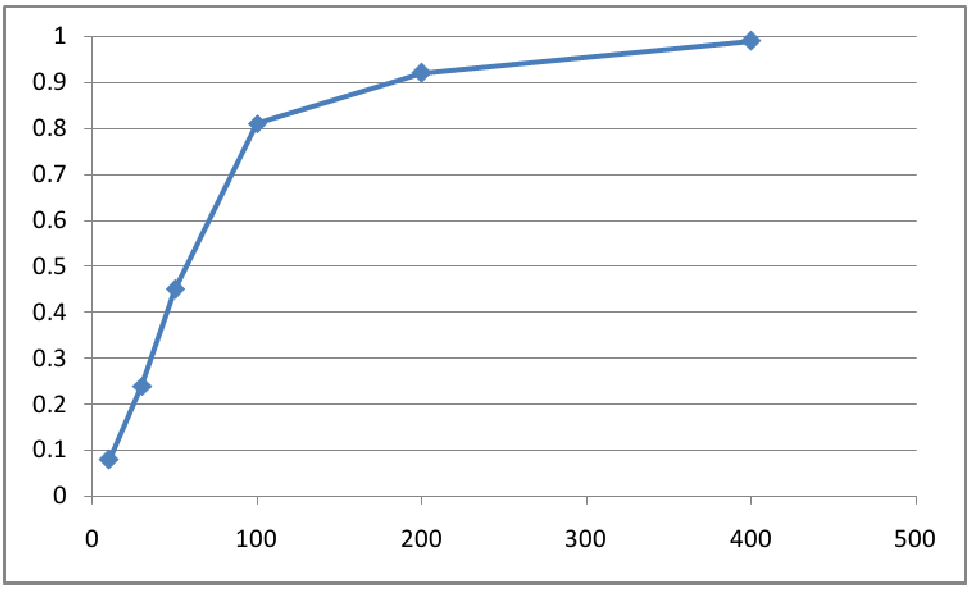
Rys. 9.2 - Wykres dokładności rozpoznania w zależności od liczby komponentów

Otrzymane wyniki pozwalają postawić tezę, że do pewnego momentu wraz ze wzrostem liczby komponentów (a więc liczby parametrów modelu) stopień generalizacji rośnie co przekłada się na skuteczność klasyfikacji nieznanych próbek. Natomiast od pewnego momentu (tutaj m=4) model zbyt dokładnie dopasowuje się do danych treningowych i generalizacja spada (pomimo że błąd wyliczany podczas treningu jest sporo mniejszy). W literaturze poświęconej systemom rozpoznawania mowy można znaleźć stwierdzenie, że najlepsze rezultaty otrzymuje się dla m=32. Jednak stwierdzenie to jest prawdziwe dla bardzo dużych korpusów językowych składających się z kilkudziesięciu godzin nagrań. Mój korpus treningowy składa się jedynie z 50 minut nagrań, a więc zawiera zbyt małą liczbę próbek uczących, aby poprawnie (mając na myśli stopień generalizacji modelu) nauczyć taką liczbę parametrów.

9.3.1.3 Wpływ promienia poszukiwań na skuteczność

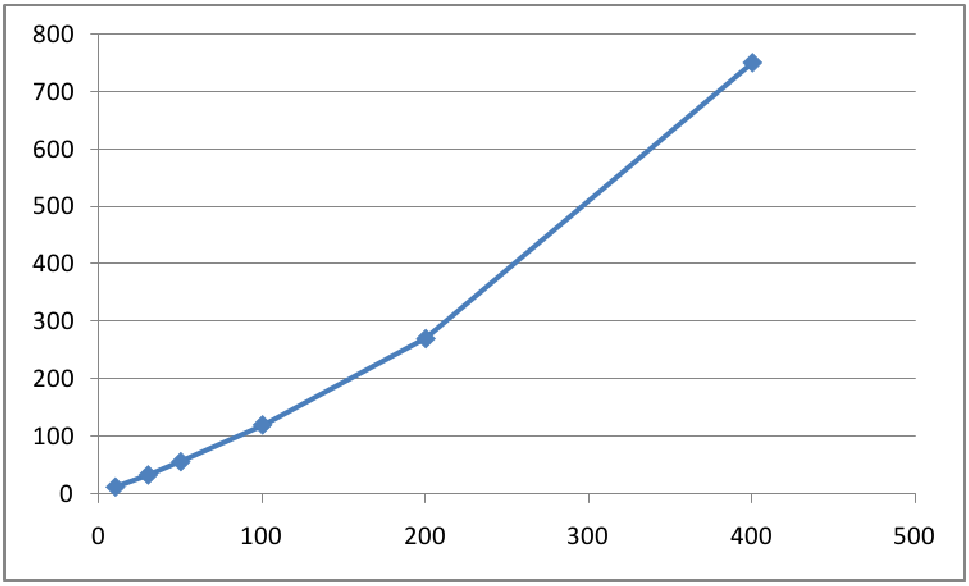
Liczba komponentów = 4.

Promień przeszukiwań	Dokładność rozp.	Czas dekodowania 360sek [sek]
10	8%	12
30	24%	33
50	45%	56
100	81%	120
200	92%	270
400	99,2%	750



Rys. 9.3 - Wykres skuteczności rozpoznania w zależności od wielkości promienia

Wraz ze wzrostem promienia poszukiwań rośnie liczba poprawnie zdekodowanych zdań (rys. 9.3), gdyż coraz więcej ścieżek w grafie podczas dekodowania jest klasyfikowanych jako prawdopodobne. Jednak wraz ze wzrostem promienia rośnie liniowo czas dekodowania (rys. 9.4). Dla modelu zbudowanego z fonemów czas dekodowania równa się czasowi wypowiedzi dla r=250 (liczba zdań wynosi 130) – otrzymujemy wtedy dokładność rozpoznania powyżej 92%.



Rys. 9.4 - Wykres zależności czasu dekodowania od promienia poszukiwań

9.3.2 Jednostki STU

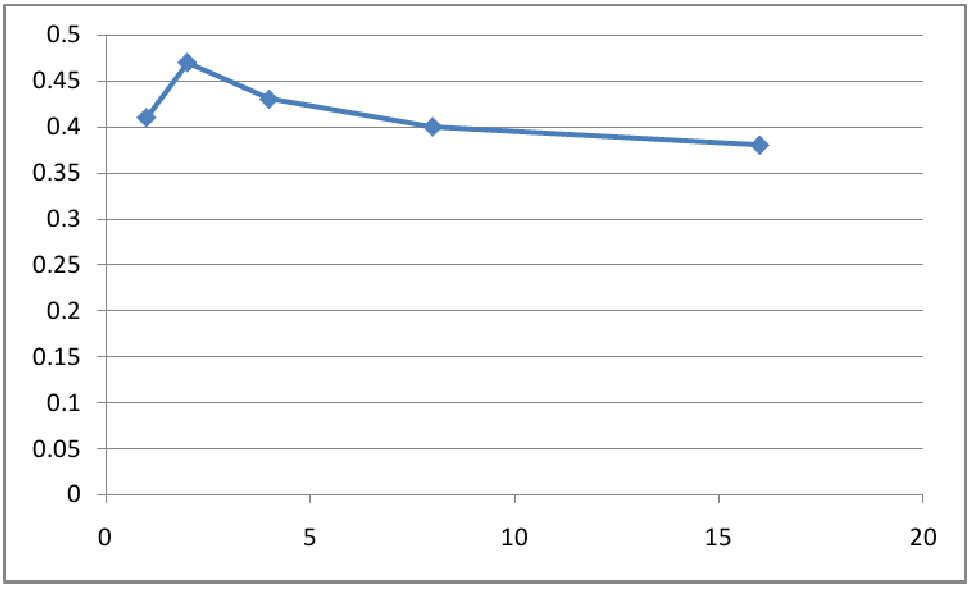
Drugim typem jednostek fonetycznych które postanowiłem przebadać są jednostki Stationary-Transitional Acoustic Units. Jednostki te są podzielone na dwa typy: reprezentujące część stacjonarną sygnału, oraz reprezentujące przejścia między poszczególnymi fonemami. Obydwie części składają się z jednego stanu. Ponieważ nie wszystkie kombinacje fonemów występują, jednostki Transitional zostały wyznaczone na podstawie transkrypcji ze zbioru treningowego. Całkowita liczba jednostek STU ostatecznie wyniosła 482. A więc mamy 5 razy więcej parametrów do nauczenia, a tyle samo danych treningowych. Poza tym wiele jednostek Transitional zostało wygenerowanych sztucznie poprzez literowanie w nagraniach nazw własnych. A więc stopień zrównoważenia danych treningowych maleje (pamiętając że pomimo tego dane treningowe nie są reprezentatywne). Zatem wyniki dla STU nie są na tyle miarodajne, aby wyciągnąć jednoznaczne wnioski.

9.3.2.1 Skuteczność rozpoznania

Dla beam=100.

Liczba komponentów	Dokładność rozpoznania
1	41%
2	47%
4	43%
8	40%

16	38%



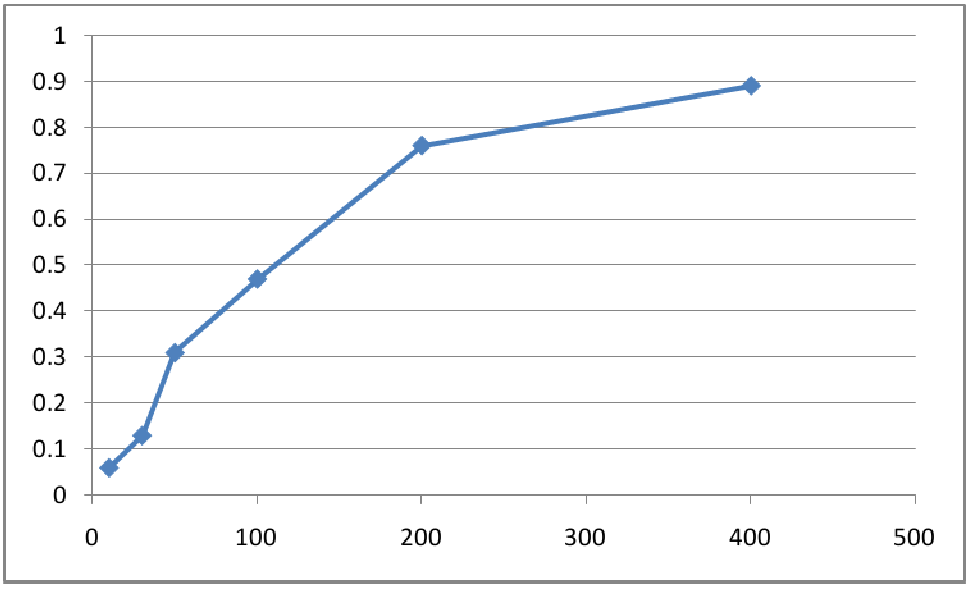
Rys. 9.5 - Wykres skuteczności dekodowania w zależności od liczby komponentów

W przypadku jednostek STU o wiele wyraźniej obserwowalny jest spadek stopnia generalizacji modelu dla coraz większej liczby parametrów.

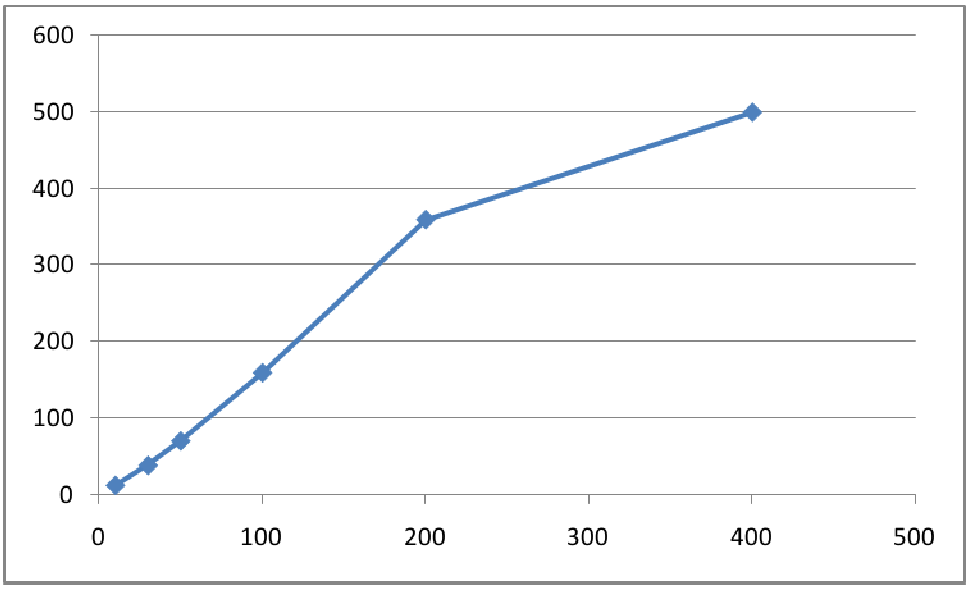
9.3.2.2 Wpływ promienia poszukiwań na skuteczność

Liczba komponentów = 4

Promień przeszukiwań	Dokładność rozp.	Czas dekodowania 360sek [sek]
10	6%	13
30	13%	39
50	31%	71
100	47%	160
200	76%	360
400	89%	500



Rys. 9.6 - Wykres skuteczności dekodowania w zależności od wielkości promienia

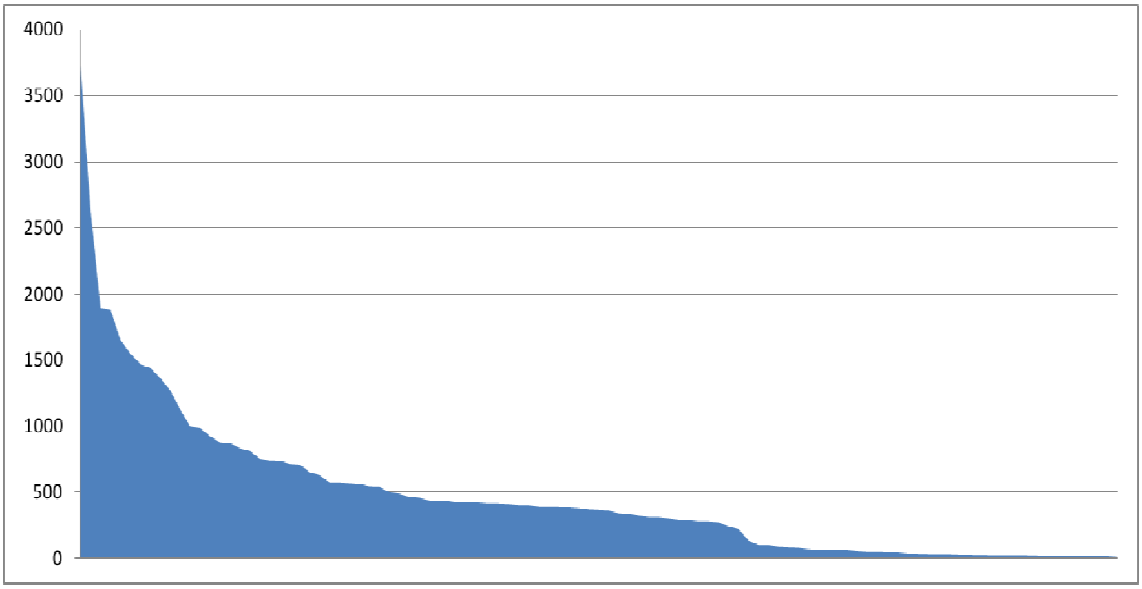


Rys. 9.7 - Wykres czasu dekodowania w zależności od promienia poszukiwań

9.4 HMM-MLP

Aby przeprowadzić trening CVT należy dokonać wstępnej segmentacji nagrań wykorzystywanych podczas budowy modelu, w tym celu korzysta się z segmentacji wykonanej przez model zbudowany za pomocą algorytmu Baum'a-Welch'a (który jak pamiętamy jest nauką nienadzorowaną). Oto histogram stanów

HMM będący efektem segmentacji przeprowadzonej przez model akustyczny HMM-GMM z punktu 9.3.1:



Rys. 9.8 – Histogram stanów HMM zbioru uczącego po tzw. segmentacji bootstrap

Największą reprezentację posiada stan <sil>_1 (reprezentujący początek ciszy) składający się z 3725 próbek uczących, pierwszy najbardziej liczny fonem S_2 składa się z 1649 próbek natomiast najmniej liczna jest reprezentacja stanu CH_1 składająca się z 3 próbek! A więc próbka losowa jest w dużym stopniu niereprezentatywna.

Sieć neuronowa w tym przypadku jest swego rodzaju modelem statystycznym i jej wyjścia estymują prawdopodobieństwa *a posteriori*, które są w dużym stopniu obciążone prawdopodobieństwami *a priori* klas ze zbioru treningowego. Dlatego rezultaty otrzymane podczas treningu są bardzo słabe (bliskie 0%). Jest to jedynie hipoteza do zweryfikowania w przyszłości za pomocą innego korpusu językowego. Autorzy pracy [8], którzy opracowali tę metodę budowy modelu akustycznego zaznaczają, że do treningu wykorzystują korpus bardzo dobrze zrównoważony, nieposiadający takich dysproporcji w reprezentacji poszczególnych fonemów jak wykorzystany przeze mnie. Wygląda na to że potwierdza się stara maksyma uczenia maszynowego - Garbage In, Garbage Out - pomimo poprawnej procedury uczenia jeśli dostarczymy błędne dane to wyniki przetwarzania również będą błędne. Rezultaty klasyfikacji sieci neuronowej estymującej prawdopodobieństwa *a posteriori* można zinterpretować w myśl Bayes'owskiej reguły wnioskowania następująco: sieć neuronowa podczas dekodowania podejmuje decyzje lokalne przypisując ramkom MFCC prawdopodobieństwa *a posteriori* przynależności do poszczególnych stanów HMM

skoro 16 (na 105 możliwych) stanów w zbiorze treningowym jest reprezentowanych przez 50% danych treningowych, to statystycznie najmniejszy błąd klasyfikacji uzyskamy częściej przypisując obiekty do najbardziej prawdopodobnych klas.

Skoro prawd. *a posteriori* opisane jest wzorem (zgodnie z regułą Bayes'a):

$$P(C_n|x) = \frac{P(x|C_n)P(C_n)}{P(x)} \tag{9.1}$$

Gdzie $P(C_n)$ jest prawdopodobieństwem *a priori* klasy C_n, $P(x)$ – prawdopodobieństwo wektora MFCC. Pamiętając, że algorytm Viterbi'ego oczekuje przeskalowanego prawd. $\frac{P(x|C_n)}{P(x)}$ należy wyjścia sieci podzielić przez prawdopodobieństwa *a priori* $P(C_n)$. W pracy [18] zostały przedstawione rezultaty tej techniki, która znacznie zredukowała błąd rozpoznania. Ku zaskoczeniu autorów największa redukcja błędu (o 50%) nastąpiła dla języka angielskiego. Jednakże stosując tę technikę nie otrzymałem zadowalających rezultatów (tylko nieznaczna redukcja błędu). Podejrzewam, że w przypadku korpusu AN4 słaba reprezentacja niektórych fonemów jest na tyle istotna, że podczas treningu sieci neuronowej minimalizowany jest głównie błąd dla klas najlepiej reprezentowanych w zbiorze treningowym. Dodatkowo wyjścia sieci są obciążone prawdopodobieństwem $P(x)$. Jeśli dla najbardziej licznych fonemów w zbiorze uczącym wektory cech znajdują się w przestrzeni R^d blisko siebie, to prawdopodobieństwo to w znacznym stopniu wpływa na błędną klasyfikację.

10 Podsumowanie

W pracy przeprowadziłem przegląd najpopularniejszych technik stosowanych w systemach automatycznego rozpoznawania mowy. Przedstawione w niniejszym tekście algorytmy mieszą się w ramach znacznie szerszej dyscypliny rozpoznawania obiektów. W detalach omówiłem najistotniejsze aspekty ukrytych modeli Markowa, które na chwilę obecną są jedynym wspólnym mianownikiem szerokiego spektrum technologii wykorzystywanych w rozwiązaniu zadnia rozpoznawania mowy. Zastosowanie modeli HMM w latach siedemdziesiątych przyczyniło się do gwałtownego rozwoju najróżniejszych technik coraz skuteczniej dekodujących ludzką mową. Ukryte modele Markowa reprezentują model statystyczny opisujący niepewność związków występujących w procesie produkcji sygnału mowy. Natomiast wektor cech akustycznych MFCC jest pierwszym algorytmem inspirowanym biologiczną budową aparatu mowy. Ze względu na wagę obu zagadnień niniejsza praca w głównej mierze koncentruje się wokół tych technologii.

Wszystkie przedstawione w pracy algorytmy zostały zaimplementowane w języku Java. Osobistym wkładem autora pracy w zrealizowane algorytmy są techniki zrównoleglenia procesu nauki jak i dekodowania. Otrzymane rezultaty potwierdzają zarówno mocne i słabe strony omówionych algorytmów. Dla dużych wartości promienia przeszukiwań zaprezentowane mechanizmy posiadają skuteczność rozpoznawania mowy powyżej 90%. Zaimplementowany w ramach pracy system ASR z powodzeniem można wykorzystać w praktycznych zastosowaniach dla gramatyk składających się z niewielkiej liczby słów.

Co więcej zreferowane w pracy rozwiązania posiadają ogromny potencjał pozwalający wprowadzać kolejne rozszerzenia poprawiające skuteczność rozpoznawania mowy. Dlatego możemy obecnie mówić o rodzinie technik bazujących na automatach HMM. Szereg technik optymalizacyjnych, omówionych również w niniejszej pracy, przyczynił się do wdrożenia wielu komercyjnych systemów ASR.

11 Literatura

[1] J.B. Hampshire, B.A. Pearlmutter “Equivalence Proofs for Multi-Layer Perceptron Classifiers and the Bayesian Discriminant Function”

[2] D. Jurafsky, J.H. Martin “Speech and language processing”

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains"

[4] Sakoe, H. and Chiba, S., “Dynamic programming algorithm optimization for spoken word recognition”

[5] C. S. Myers and L. R. Rabiner. “A comparative study of several dynamic time-warping algorithms for connected word recognition.”

[6] Aldrich, John "R.A. Fisher and the making of maximum likelihood 1912-1922”

[7] H. Bourlard, N. Morgan “Connectionist Speech Recognition A Hybrid Approach”

[8] R. Gemello, D. Albesano, F. Mana, “Continuous Speech Recognition with Neural Networks and Stationary-Transitional Acoustic Units”

[9] R. Gemello, D. Albesano, F. Mana, "Multi-source neural networks for speech recognition

[10] R. Gemello, D. Albesano, F. Mana “CSELT Hybrid HMM/Neural Networks Technology for Continuos Speech Recognition”

[11] M.A. Franzini, K.F. Lee and A. Waibel, "Connectionist Viterbi Training: A new hybrid method for continuous speech recognition"

[12] S. Renals , N. Morgan, H. Bourlard, M. Cohen, H. Franco “Connectionist Probability Estimators in HMM Speech Recognition”

[13] H. Hermansky, “Perceptual linear predictive analysis of speech”

[14] H. Hermansky, N. Morgan, “RASTA Processing of Speech”

[15] R. Gemello, D. Albesano, F. Mana, “Multi-source neural networks for speech recognition”

[16] R. De Mori, R. Gemello, D. Albesano, F. Mana, “Ear-model derived features for automatic speech recognition”

[17] R. Gemello, D. Albesano, F. Mana, P. Pegoraro, “Multi Source Neural

Networks based on fixed and Multiple Resolution Analysis for Speech Recognition”

[18] R. Gemello, D. Albesano, F. Mana, “Hybrid HMM-NN for speech recognition and prior class probabilities”

[19] L. Fissore, F. Ravera, P. Laface, “Acoustic-Phonetic Modeling For Flexible Vocabulary Speech Recognition”

[20] R. G. Lyons “Wprowadzenie do cyfrowego przetwarzania sygnałów”

[21] Andrew J. Viterbi. “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”

[22] http://www.cslu.ogi.edu/people/hosom/cs552/

[23] http://www.stanford.edu/class/cs224s/

[24] C.M. Bishop „Neural networks for pattern recognition”

[25] M. Kurzyński „Rozpoznawanie obiektów – metody statystyczne”

[26] M. Sobczyk „Statystyka”

[27] L. R. Rabiner „A tutorial on Hidden Markov Models and Selected Applications In Speech Recognition” IEEE

[28] B.H. Juang, L.R. Rabiner “Hidden Markov Models for Speech Recognition”

[29] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, P. Woodland “The HTK Book”

[30] L.E. Baum, J.A. Egon “An inequality with applications to statistical estimation for probabilistic functions of Markov process and to model for ecology”

[31] A.P. Dempster, N.M. Laird, D.B. Rubin “Maximum likelihood from incomplete data via EM algorithm”

[32] S.B. Davis, and P. Mermelstein (1980), "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," in IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(4), pp. 357–366

[33] E. Gouvea, W. Walker et al. “Sphinx 4 for the JavaTM platform Architecture Notes”

[34] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, J. Woelfel, “Sphinx-4: A Flexible Open Source Framework for Speech Recognition”

[35] Jeff A. Bilmes, “A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models”

[36] S. Kumar “Feature Selection and Non-linear Feature Extraction”

[37] J. Picone, “Signal Modeling Techniques In Speech Recognition”

[38] E. Ozimek “Dźwięk i jego percepcja. Aspekty fizyczne i psychoakustyczne”

[39] wikipedia.org