|  |
| --- |
| Politechnika Śląska |
| Raport techniczny |
| Implementacja systemu ASR |
|  |
| **Łukasz Olczak** |
| **2009-09-03** |

|  |
| --- |
| Raport z postępów pracy nad pracą magisterską, przedstawia aspekty techniczne implementacji oraz omówienie wyników. |

1. Wstęp
   1. Cel dokumentu

Celem dokumentu jest przedstawienie stanu implementacji systemu ASR realizowanego w ramach praca dyplomowej. Dokument zawiera zarówno szczegóły techniczne zaimplementowanych algorytmów jak również omówienie uzyskanych wyników, szczegółowy opis metodyki przeprowadzania testów oraz platformy na której przeprowadzono pomiary.

* 1. Zaimplementowane algorytmy
     1. Algorytm Baum-Welch’a

Zaimplementowana została wersja dostosowana do treningu typu embedded training. Ze względu na długi czas trwania nauki zaimplementowano wersję równoległą, która skaluje się prawie liniowo.

* + 1. Algorytm Viterbi’ego

Wersja parametryzowana za pomocą promienia zawężającego przestrzeń poszukiwań.

* + 1. Connectionist Viterbi Training

Wersja z iteracyjną segmentacją, plus równoległa implementacja algorytmu back-propagation w trybie batch.

* + 1. Ekstrakcja cech z sygnału akustycznego

Adapter algorytmu zaimplementowanego w Sphinx4.

* + 1. Recognizer

Modułowy recognizer pozwalający dekodować sygnał akustyczny mowy. Wersja abstrahuje czy rozpoznawanie odbywa się za pomocą klasycznego modelu akustycznego GMM/HMM czy hybrydy zbudowanej w oparciu o sieć neuronową.

* 1. Technologia

- całość została zaimplementowana w języku Java 1.6

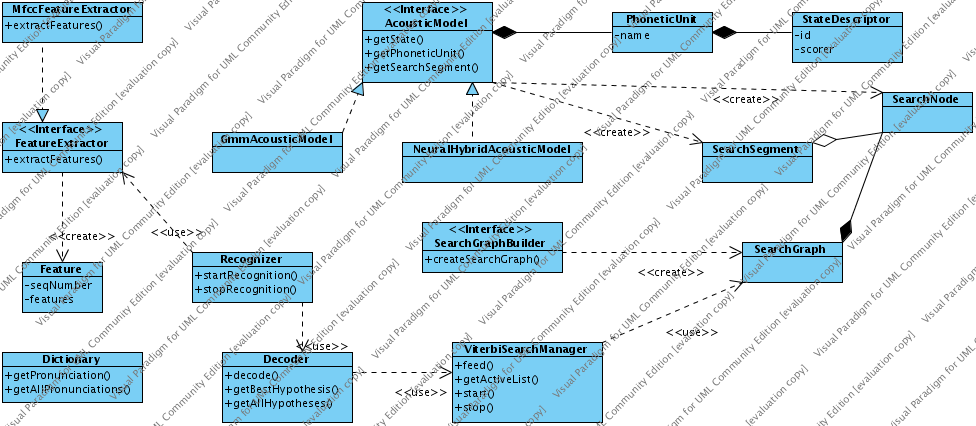
- zarządzanie strukturą projektu oraz zależnościami zajmuje się Maven 2

- testy jednostkowe uruchamiane za pomocą TestNG

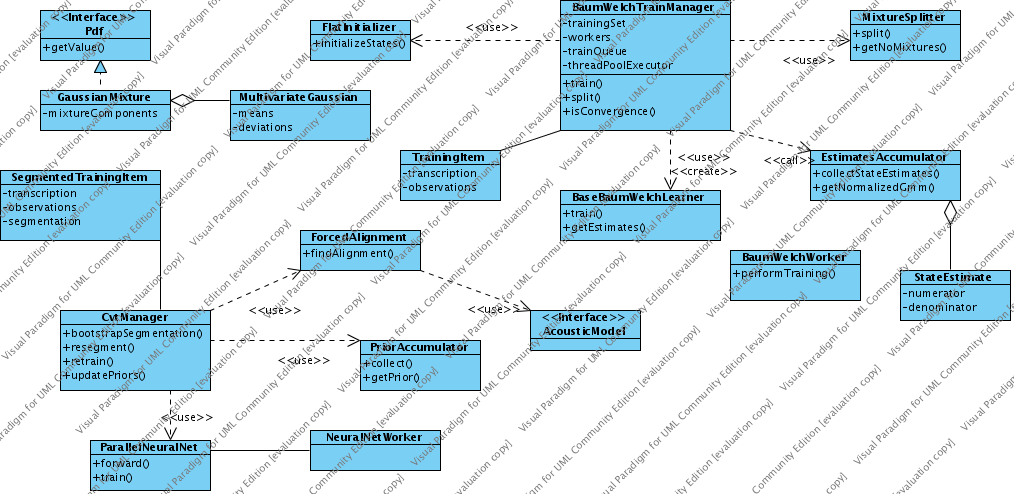
- implementacja sieci neuronowej Joone 2

- biblioteki JMF, Sphinx4, SpringFramework

1. Specyfikacja techniczna
   1. Konceptualny diagram klas odpowiedzialnych za rozpoznawanie



* 1. Diagram klas odpowiedzialnych za naukę



* 1. Odpowiedzialności poszczególnych klas
     1. FeatureExtractor

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.

* + 1. Dictionary

Klasa ta odpowiedzialna jest za konwersję zapisu znakowego słowa na zapis fonetyczny.

* + 1. AcousticModel

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.

GmmAcousticModel – jest modelem akustycznym zbudowanym w oparciu o funkcje gęstości prawdopodobieństwa Gaussa. Implementacja realizuje wielowymiarowe funkcje Gaussa złożone z wielu komponentów.

NeuralHybridAcousticModel – model ten wykorzystuje sieć neuronową do estymacji prawdopodobieństw a posteriori dla kolejnych wektorów cech sygnału mowy.

* + 1. SearchGraphBuilder

Interfejs ten odpowiedzialny jest za budowanie grafu poszukiwań wykorzystywanego przez dekoder podczas rozpoznawania mowy.

* + 1. PhoneticUnit

Reprezentuje jednostkę fonetyczną z której składają się rozpoznawane słowa. Każda jednostka fonetyczna jest zrealizowana za pomocą jednego lub kilku stanów HMM.

* + 1. StateDescriptor

Reprezentuje stan HMM, który dekomponuje jednostki fonetyczne. Odpowiedzialnością klasy jest oszacowanie prawdopodobieństwa przynależności wektora wejściowego do klasy fonetycznej którą reprezentuje stan.

* + 1. SearchNode, SearchSegment, SearchGraph

Reprezentują składowe elementy grafu poszukiwań używanego podczas dekodowania.

* + 1. Feature

Reprezentuje wektor cech sygnału mowy.

* + 1. ViterbiSearchManager

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.

* + 1. Dekoder

Deleguje wywołania do SearchManagera, interpretuje wyniki poszukiwań.

* + 1. Recognizer

Scala dekoder, feature extractor, search graph builder w całość, udostępniając prosty interfejs klasom wykorzystującym recognizer.

* + 1. MultivariateGaussian

Klasa odpowiedzialna za wyliczanie wartości funkcji gęstości prawdopodobieństwa wielowymiarowego rozkładu Gaussa. Enkapsułkuje parametry rozkładu.

* + 1. GaussianMixture

Klasa ta enkapsułkuje algorytm wyliczania wartości funkcji gęstości prawdopodobieństwa dla rozkładu składającego się z wielu komponentów Gaussa.

* + 1. BaumWelchLearner

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.

* + 1. EstimatesAccumulator

Akumulator estymatorów parametrów modelu GMM/HMM. Estymatory są składowane w postaci licznik+mianownik. Odpowiedzialnością tej klasy jest również wyliczanie zakumulowanej wartości poszczególnych parametrów.

* + 1. FlatInitializer

Enkapsułkuje algorytm inicjalizacji parametrów GMM dla poszczególnych stanów wartościami średnimi wyliczonymi ze zbioru treningowego.

* + 1. MixtureSplitter

Odpowiedzialny jest za ‘rozdwojenie’ pojedynczego komponentu Gaussa na dwie składowe.

* + 1. TrainingItem

Reprezentuje próbkę uczącą. Składa się z transkrypcji oraz ciągu wektora cech sygnału akustycznego.

* + 1. BaumWelchTrainManager

Zarządza procesem uczenia typu embedded training. Implementuje równoległy algorytm uczenia Baum-Welch’a.

* + 1. BaumWelchWorker

Podstawowy element wykorzystywany w równoległym algorytmie Baum-Welcha. Działa w osobnym wątku. Odpowiedzialny jest za naukę próbek uczących pobieranych z kolejki.

* + 1. ForcedAlignment

Odpowiedzialny jest za segmentację wektora cech sygnału akustycznego. Dokonywany jest za pomocą algorytmu Viterbiego dla grafu zbudowanego dla pojedynczego słowa.

* + 1. SegmentedTrainingItem

Reprezentuje próbkę uczącą. Zawiera ‘posegmentowany’ ciąg wektora cech.

* + 1. PriorAccumulator

Odpowiedzialny jest za wyliczanie prawdopodobieństw a priori klas na podstawie zbioru treningowego.

* + 1. ParallelNeuralNet

Enkapsułkuje równoległy algorytm uczenia sieci neuronowej.

* + 1. CvtManager

Implementuje algorytm connectionist Viterbi training. Zarządza procesem uczenia, re segmentacją.

* 1. Algorytm Baum-Welch’a

Zaimplementowałem wersję algorytmu Baum-Welch’a dostosowaną do rozpoznawania mowy. A więc 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. A więc 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 tzn. że prawdopodobieństwa emisji stanów są opisane za pomocą ciągłej 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 wszystkich próbek uczących. 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. Szczegółowy algorytm opiszę w pracy magisterskiej, celem tego dokumentu jest jedynie zarysowanie szczegółów implementacji.

Oto pseudokod treningu:

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

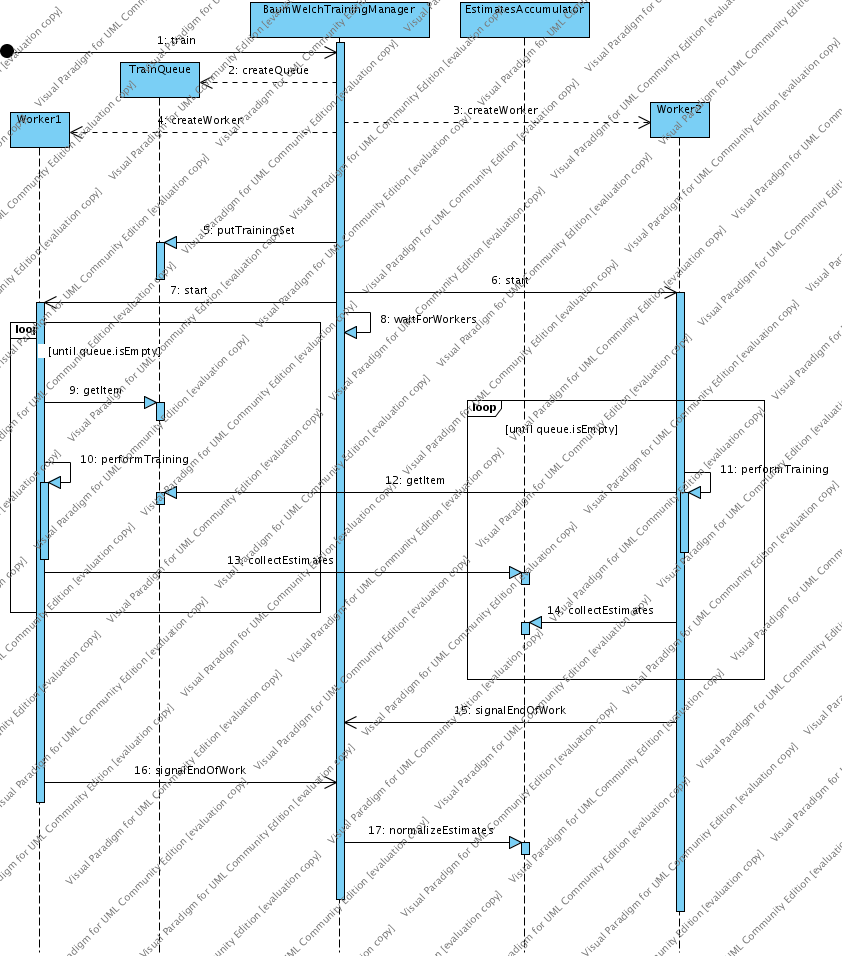
mixtureSplitter.split(acousticModel)

endIf

while not convergence or noMixture < desiredNoMixtures

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, aby przyspieszyć czas nauki. Ponieważ każda iteracja wewnętrznej pętli for (parz listing wyżej) wykonywana jest dla innych danych postanowiłem tę część zrównoleglić. A więc 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 menadżera o końcu pracy. Menadżer przeprowadza normalizację estymatorów sekwencyjnie. Następnie zdania są ponownie wrzucane do kolejki i przeprowadzana jest kolejna iteracja równoległego algorytmu Baum-Welcha.

Przypadek dwuprocesorowy został schematycznie przedstawiony na poniższym diagramie przebiegu.



Podczas obliczeń często wykonuje się mnożenie bardzo małych wartości i w efekcie uzyskujemy "floating point underflow". Toteż wszystkie obliczenia są wykonywane w skali logarytmicznej, gdzie operację mnożenia zastępujemy operacją dodawania.

* 1. Connectionist Viterbi Training

Ze względu na wady algorytmu Baum’a-Welch’a takie jak brak treningu dyskryminującego, mocne założenia co do kształtu rozkładu prawdopodobieństw emisji (Rabiner badając rozkłady jednostek akustycznych zauważył, że jednostki posiadają rozkład zbliżony do rozkładu gamma a nie Gaussa) oraz z góry zadana liczba komponentów, postanowiłem w ramach magisterki zaimplementować jedno z rozwiązań hybrydowych ASR łączące zdolność modelowania wymiaru czasu za pomocą HMM z treningiem dyskryminującym. W tym celu postanowiłem przetestować hybrydę MLP/HMM opisaną w pracy [1].

Rozwiązanie to wykorzystuje wyjścia sieci neuronowej jako estymatory prawdopodobieństwa *a posteriori* klas (w naszym przypadku jednostek fonetycznych). Sieć neuronowa podejmuje decyzje lokalne natomiast algorytm Viterbiego podejmuje decyzję globalną. W pracy [2] można znaleźć dowód, że pomimo to, że trening był dyskryminujący dla decyzji lokalnych to hybryda MLP/HMM również dyskryminuje globalnie, a więc na poziomie słów.

Algorytm uczenia takiego modelu łączy w sobie algorytm treningu Viterbiego oraz backpropagation. Standardowo procedura uczenia wygląda następująco: korzystając z początkowej segmentacji wykonanej za pomocą GMM/HMM (algorytm Baum-Welcha), 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 backpropagation dla małego zbioru treningowego – 200 nagrań – trwa kilkanaście godzin na procesorze AMD Turion64) w związku z czym jest niewykonalny na standardowych stacjach roboczych. Dlatego w pracy [1] 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 stacjach roboczych.

Topologię sieci neuronowej oraz detale związanie z wykorzystaniem MLP do rozpoznawania mowy opisałem w poprzednim raporcie.

Pseudokod algorytmu:

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

Inicjalizacja:

- normalizuję dane treningowe

- wykonuję pierwszą segmentację (bootstrap segmentation) korzystając z modelu akustycznego GMM/HMM zbudowanego za pomocą algorytmu Baum-Welcha

- inicjalizuję wagi sieci neuronowej małymi losowymi wartościami

Kolejne iteracje:

- ładuję segmentację z poprzedniej epoki

- dla tej segmentacji obliczam estymatory prawdopodobieństw a posteriori dla wszystkich stanów

- wykonuję jedną iterację algorytmu back-propagation

- wykorzystuję sieć neuronową do wyznaczenia nowej segmentacji próbek uczących

- aktualizuję segmentację dla kolejnej iteracji za pomocą funkcji

F(sold, snew) = α·sold+(1-α) ·snew , gdzie

sold – to segmentacja z poprzedniej iteracji

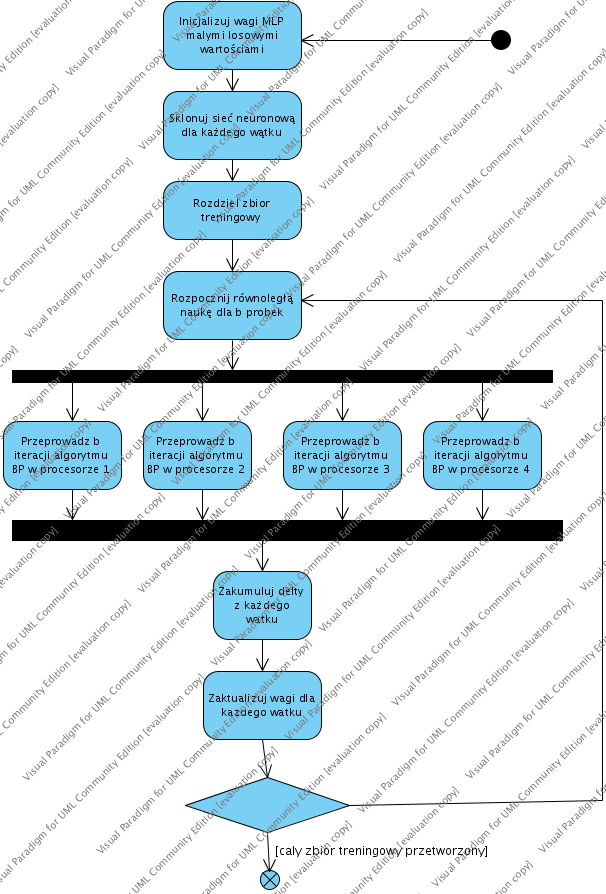
snew – segmentacja wykonana przez sieć neuronową z bieżącej iteracji

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.

* 1. Równoległa wersja back-propagation

Pomimo niezbyt dużego 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 akceptowalne, gdyż częsta synchronizacja między wątkami jest tutaj wąskim gardłem dramatycznie redukującym skalowalność algorytmu. Toteż postanowiłem zrównoleglić algorytm BP 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 backpropagation przedstawia poniższy diagram czynności.

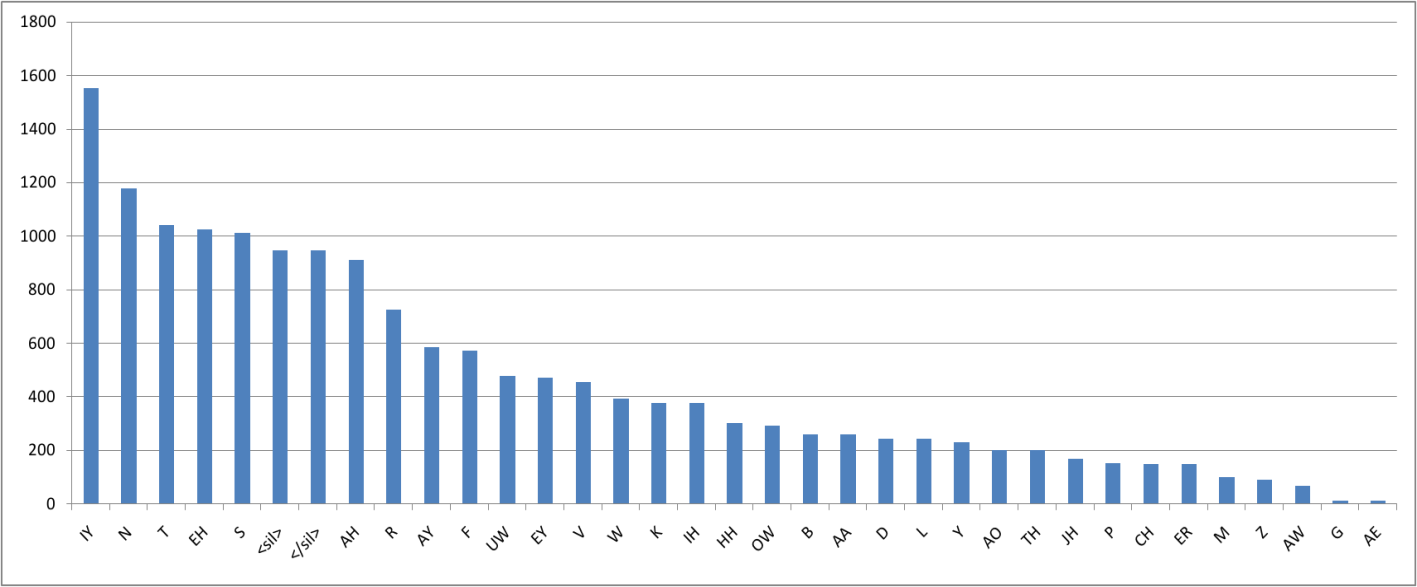


1. Korpus językowy

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 środowisko 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.



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.

1. Testy
   1. Środowisko testowe

Ze względu 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 były przeprowadzane na 64 bitowym systemie Ubuntu Linux, implementacja maszyny wirtualnej Javy to 64 bitowa wersja Sun’owska.

* 1. 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.

* 1. Wyniki dla GMM/HMM

W punkcie 3 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).

Ostatecznie wektor MFCC składa się z 12 elementów cepstrum mocy (CEP) 1 elementu energii E ramki, oraz pierwszej (ΔCEP,ΔE) i drugiej (ΔΔCEP,ΔΔE) pochodnej. Tak skonstruowany wektor MFCC ma jedną bardzo ważną zaletę: wszystkie składowe wektora są niezależne. Zatem podczas nauki możemy założyć, że macierz kowariancji jest diagonalna i tym samym ograniczyć liczbę parametrów wyznaczanych podczas nauki.

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.

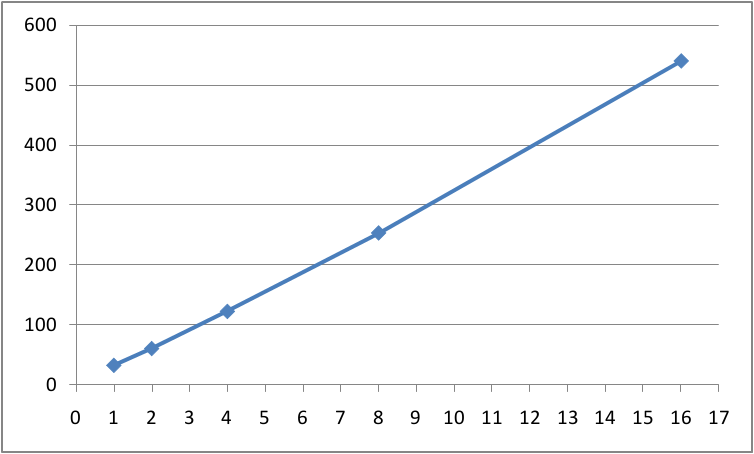
* + 1. Fonemy

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

* + - 1. Długość treningu

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 |

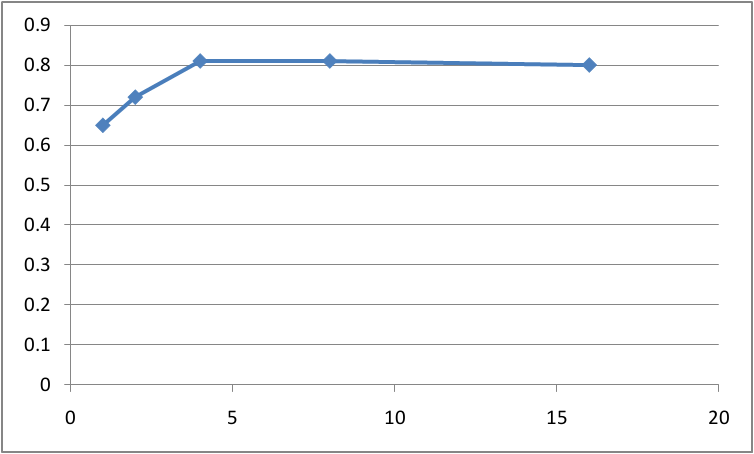


* + - 1. Dokładność rozpoznania

Beam=100

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

Wykres dokładności rozpoznania w zależności od liczby komponentów.



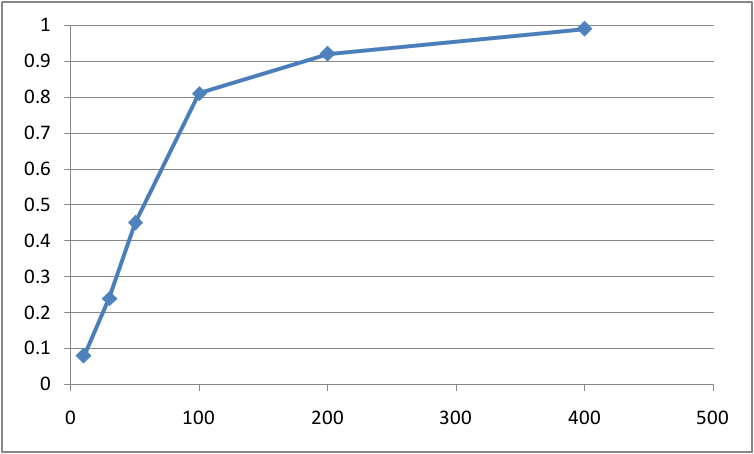
Otrzymane wyniki pozwalają postawić tezę, że do pewnego momentu wraz ze wzrostem liczby komponentów (a więc liczby parametrów modelu) stopień generalizacji rośnie co przekłada się na skuteczność klasyfikacji nieznanych próbek. Natomiast od pewnego momentu (tutaj m=4) model zbyt dokładnie dopasowuje się 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.

* + - 1. Wpływ promienia poszukiwań na dokładność

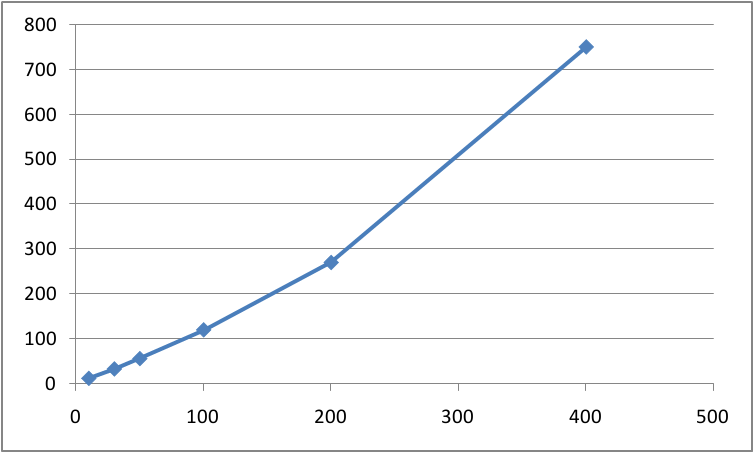
Liczba komponentów = 4

|  |  |  |
| --- | --- | --- |
| 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 |

Wykres dokładności rozpoznania w zależności od wielkości promienia:



Wykres zależności czasu rozpoznania od promienia poszukiwań:



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

* + 1. Jednostki STU

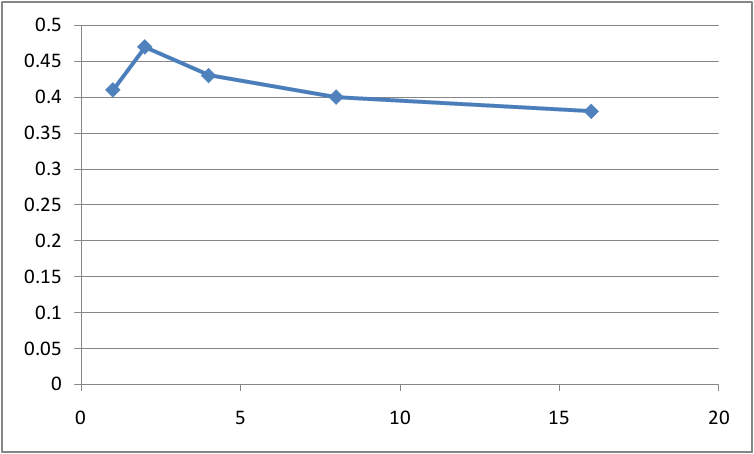
Drugim typem jednostek fonetycznych które postanowiłem przebadać są jednostki Stationary-Transitional Acoustic Units. Jednostki te są podzielone na dwa typy: reprezentujące część stacjonarną sygnału, oraz reprezentujące przejścia między poszczególnymi fonemami. Obydwie części składają się z jednego stanu. 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.

* + - 1. Dokładność rozpoznania

Beam=100

|  |  |
| --- | --- |
| Liczba komponentów | Dokładność rozpoznania |
| 1 | 41% |
| 2 | 47% |
| 4 | 43% |
| 8 | 40% |
| 16 | 38% |

Wykres dokładności dekodowania w zależności od liczby komponentów.



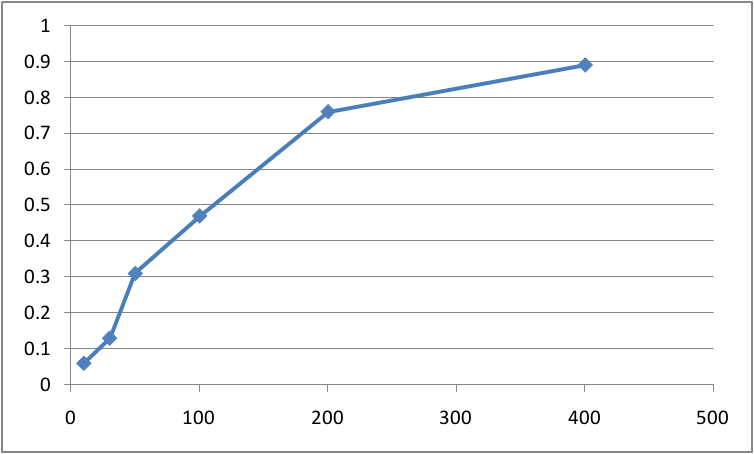
W przypadku jednostek STU o wiele wyraźniej obserwowalny jest spadek stopnia generalizacji modelu dla coraz większej liczby parametrów.

* + - 1. Wpływ promienia poszukiwań na dokładność

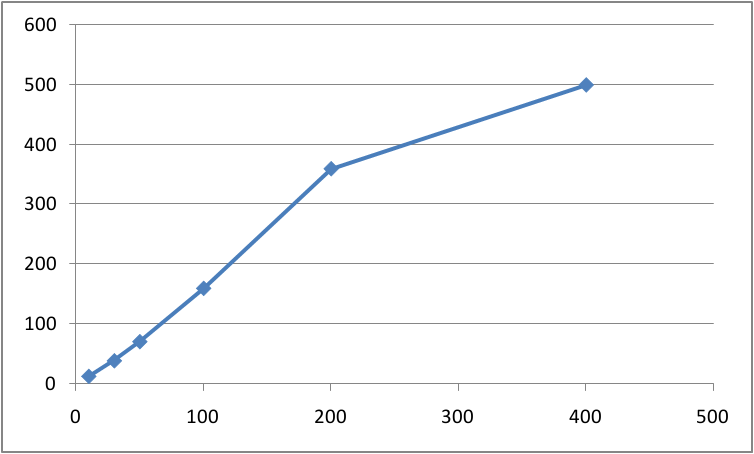
Liczba komponentów = 4

|  |  |  |
| --- | --- | --- |
| 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 |

Wykres dokładności rozpoznania w zależności od wielkości promienia:

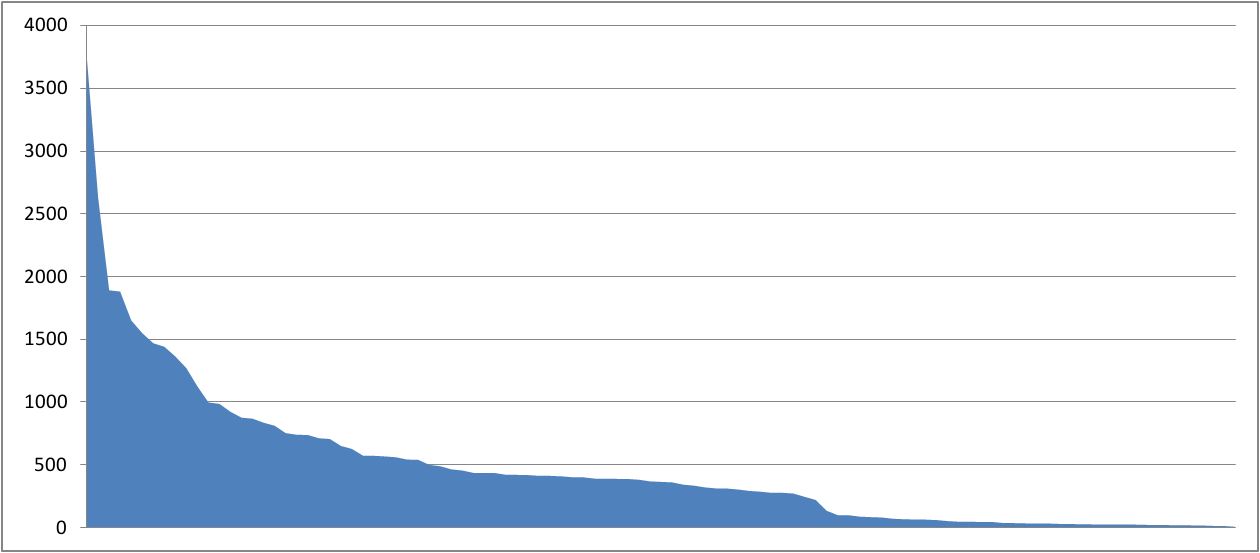


Wykres zależności czasu dekodowania od promienia poszukiwań:



* 1. Rezultaty uzyskane dla hybrydy MLP-HMM

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 GMM/HMM z punktu 4.3.1:



Największą reprezentację posiada stan <sil>\_1 (reprezentujący początek ciszy) składający się z 3725 próbek uczący, 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 [1] którzy opracowali tę metodę budowy modelu akustycznego zaznaczają że do treningu wykorzystują korpus bardzo dobrze zrównoważony, nie posiadający 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 przyporzadkując dane wejściowe do najbardziej prawdopodobnych klas.

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

Gdzie *P(Cn)* jest prawdopodobieństwem *a priori* klasy *Cn, P(x)* – prawdopodobieństwo wektora MFCC. Pamiętając, że algorytm Viterbi’ego oczekuje przeskalowanego prawd. (*P(x)* jest do pominięcia ponieważ jest ono takie samo podczas dekodowania) należy wyjścia sieci podzielić przez prawdopodobieństwa *a priori P(Cn)*. W pracy [3] 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.

1. Podsumowanie

Ogólnie implementacja zajęła mi 3 miesiące (27 tyś. linijek kodu), ostatni miesiąc głównie poświęciłem na algorytm CVT, który ostatecznie nie udało mi się zmusić do działania. Aczkolwiek algorytm dla danych syntetycznych (z równomiernie reprezentowanymi klasami) działa poprawnie, dlatego jako najbardziej prawdopodobną hipotezę braku rezultatów hybrydy MLP-HMM przyjmują nieodpowiedni korpus językowy.

W przyszłości chciałbym skupić się na przetestowaniu pozostałych algorytmów cech akustycznych PLP, RASTA-PLP, HFCC, implementacji filtrów adaptacyjnych (noise cancellation, AGC etc) poprawiających skuteczność dekodera oraz zastosowanie hybrydy MLP, która zapewnia dyskryminujący trening.

1. Literatura

|  |  |
| --- | --- |
| [1] | R. Gemello, D. Albesano, F. Mana, “Continuous Speech Recognition with Neural Networks and Stationary-Transitional Acoustic Units” |
| [2] | H. Bourlard, N. Morgan “Connectionist Speech Recognition A Hybrid Approach” |
| [3] | R. Gemello, D. Albesano, F. Mana, “Hybrid HMM-NN for speech recognition and prior class probabilities” |