OPRACOWANIE

Nie ma rozpoznawania mowy -> można rozpoznawać słowa OOV (spoza słownika)  
Składniki:  
 > enkoder/dekoder mowy  
 > enkoder/dekoder wejść (query enc/dec)  
 > attention mechanism  
 > Energy scorer

„First, we obtain speech and query embeddings by speech and query encoder, respectively. After that, we apply the attention mechanism to output a set of attention weights. Finally, the energy scorer takes all of them as input and outputs the final results”

DANE:

„We do not transform the input audio into a fixed-length vector but a variable-length matrix instead, where we keep the time dimension.”  
Babel-102 Assamese, Babel-103 Bengali, Babel-104 Pashto  
^ pliki .sph (konwertowalne do .wav) i transkrypcje  
<https://catalog.ldc.upenn.edu/LDC2016S06> <- tutaj babel102 kosztuje 25$  
<https://github.com/kaldi-asr/kaldi/blob/master/egs/babel/s5b/conf/lang/102-assamese-limitedLP.official.conf>  
<https://github.com/espnet/espnet/blob/master/egs/babel/asr1/README.md>  
  
FREE DATASETS WITH TRANSCRIPTIONS:  
[librispeech dataset](https://datasets.activeloop.ai/docs/ml/datasets/librispeech-dataset/)  
[tedlium dataset](https://www.tensorflow.org/datasets/catalog/tedlium)  
  
^ przykład e2e processing na tych samych danych  
Wszystkie nagrania podzielone na 1s fragmenty (precyzja wyszukiwania do 1s)  
Wejście = wektor wektorów odpowiadających 1s fragmentom nagrań

UCZENIE:

Uczenie osobno: enkodery/dekodery mowy i query, attention mechanism, Energy scorer  
Uczenie z wykorzystaniem cross-entropy loss function + Adam optimiser (learning rate 0.001)

[ADAM OPTIMISER](https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/):

**Adam Configuration Parameters**

* **alpha**. Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
* **beta1**. The exponential decay rate for the first moment estimates (e.g. 0.9).
* **beta2**. The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
* **epsilon**. Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).

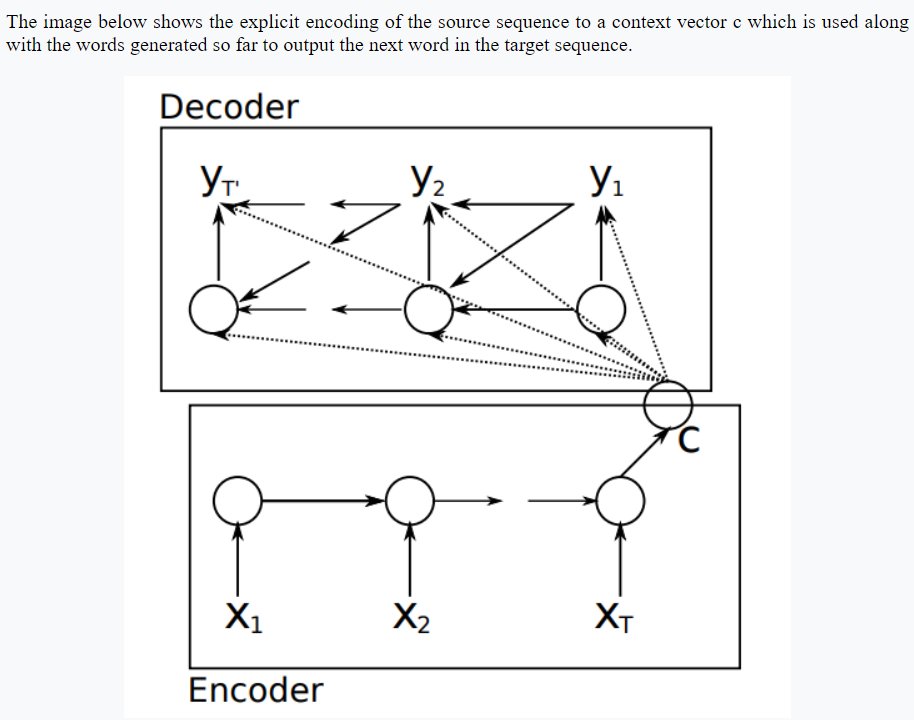
Zarys ogólny architektury:

Input Speech -> Speech Encoder -> Speech Embeddings  
 Input Query -> Query Encoder -> Query Embeddings  
 Speech, Query Embeddings -> Attention Mechanism -> Attention Weights  
 Speech, Query Embeddings + Attention Weights -> Energy Scorer -> Decision

ENKODER/DEKODER MOWY

(dekoder jest wykorzystywany tylko w procesie uczenia)  
X\* = {x1\*,x2\*, …, xT\*} // cechy mowy na wejściu  
// xi – wektor odpowiadający 1s fragmentowi nagrania  
-> 1D CNN(128 filters) -> wektor[T] -> // 128 to kernel\_size? Punkt 3.1 w artykule  
-> 1D CNN(256 filters) -> wektor[T] ->  
 -> 1D max pooling (stride = 2) -> X = {x1, x2, …, x­T/2} -> // wektor[T/2]  
-> GRU(input\_size=T/2, hidden\_size=T/2, num\_layers = 256) -> // GRU Encoder  
-> Fully Connected Layer (ReLu) -> embeddings ES = {e1,…,eT/2} -> // GRU Encoder  
// X = {x1,…,xT/2} -> GRU Encoder -> ES = {e1, …, eT/2}  
// index s oznacza „speech” – Eq -> query embedding  
// indeks S to krok przetwarzania - encoder zwraca na wyjściu S-ty embedding  
// docelowo E­S trafia dalej do Energy Scorer, Attention Mechanism  
// w procesie uczenia ES przekazywane jest na wejście Attention-GRU-Decodera  
Es -> GRU(input\_size, hidden\_size, num\_layers = 256) -> ht  
At = {αt(1), αt(2), …, αt(T/2)}  
// ^ wagi attention przypisane każdej próbce es ∈ Es  
αt(s) =   
ct = 𝛴s αt(s)es // context (wektor[T/2]) - suma ważona  
-> ot = FCL([htT, ctT]T) // softmax  
// nb of units in the FCL = nb of characters in the target laguage  
// o­t­ – wyjście dekodera (prawdopodobieństwo występowania KW w momencie dekodowania t.

GRU ENCODER

[seq2seq](https://towardsdatascience.com/word-level-english-to-marathi-neural-machine-translation-using-seq2seq-encoder-decoder-lstm-model-1a913f2dc4a7)  
[attention mechanism](https://towardsdatascience.com/intuitive-understanding-of-attention-mechanism-in-deep-learning-6c9482aecf4f)  
  
[^POWYZSZY OBRAZEK](https://tutorials.one/gentle-introduction-to-global-attention-for-encoder-decoder-recurrent-neural-networks/%23content)

GRU – Gated Recurrent Unit (CLASStorch.nn.GRU(*\*args*, *\*\*kwargs*))

* **input\_size** – The number of expected features in the input x
* **hidden\_size** – The number of features in the hidden state h
* **num\_layers** – Number of recurrent layers. E.g., setting num\_layers=2 would mean stacking two GRUs together to form a stacked GRU, with the second GRU taking in outputs of the first GRU and computing the final results. Default: 1

QUERY ENCODER

Podobnie jak w przypadku enkodera mowy, dekoder nie jest wykorzystywany – uzyskane query embeddings są  
przekazywane na wejście Attention Mechanism i Energy Scorer.  
Input = sequence of characters  
-> embedding layer(in: var length vector[char], out: 256 length zapętlone) -> ?? ->  
-> GRU encoder  
 (in: 256(??), out: 256, 128 layers)  
 -> Eq

Input (sequence of chars) -> GRU (??, 256,128 layers (units)) -> 256-element vector // [seq2seq](https://towardsdatascience.com/word-level-english-to-marathi-neural-machine-translation-using-seq2seq-encoder-decoder-lstm-model-1a913f2dc4a7) method

ATTENTION MECHANISM

Na wyjściu każdemu time-step t speech embeddingsów przypisywana jest waga -> prawdopodobieństwo występowania  
w danym momencie słowa kluczowego.  
CONCAT(speech embeddings, query embeddings) -> bidirectional LSTM-RNN ->   
// query embeddins zapętlone aby ich długość pasowała do speech emb.  
// [bi-LSTM-RNN](https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html) – input\_size = 2\*length(speech embedding) = length(concatenated vector),  
// hidden\_size = length(speech embedding)  
// num\_layers = default = 1  
// bidirectional = True  
// „The loss function applied for both attention mechanism and energy scorer are binary cross-entropy, and we set their // loss weights equally”  
-> FCL (in: length(speech embedding), out: (0;1), [Sigmoid](https://pytorch.org/docs/stable/generated/torch.nn.Sigmoid.html), one single neural unit) -> Attention output (α)

ENERGY SCORER

#TODO r ≶

PONIŻEJ BRUDNOPIS

T – numer of time steps  
x1, x2, …, xT – wektory!!!  
1dcnn@128 przerabia T wektorów  
1dcnn@256 przerabia T wektorów  
1d max pooling (stride = 2) przerabia T wektorów na T/2 wektorów

input size = T / 2  
hidden size = T / 2  
num\_layers = 256 // paragraf 3.1  
  
  
multilayer GRU -> fully connected layer with ReLu  
Es = Encoder(X) // type(wejście) = type(wyjście) = Tensor(1, T/2) (wektor)

GRU DECODER

input size = T / 2  
hidden size = ?  
num\_layers = 256 // paragraf 3.1  
ES -> GRU dekoder (global attention):  
for es in Es:  
 αt(s) = align(ht,es) = // waga uwagi (attention weight) dla sth embedding w tth kroku dekodowania  
 exp(x) = ex, ht – stan ukryty dekodera w momencie t, score – iloczyn skalarny,

\-> wektor kontekstu ct = 𝛴s αt(s)es // suma wazona embeddingsow es  
\-> ot = FCL([htT, ctT]T) // predykcja: konkatenacja wektora kontekstu i stanu ukrytego, Fully Connected Layer with softmax activation; ilość jednostek (neuronów) w FCL = ilość liter w danym języku

QUERY ENCODER/DECODER

Seq2seq zamiast attention  
input query (wektor znakow) -> 2\*GRU(128layers(units), input\_size=??, hidden\_size=256)  
->wektor[256]  
  
ATTENTION MECHANISM

W każdym kroku wyznacza wagę -> prawdopodobieństwo wystąpienia danego KW  
Concat(zakodowana cecha mowy, zakodowana cecha KW) -> bidirectional LSTM-RNN  
// KW (queries) należy powtarzać żeby pasowały długością do mowy  
\-> FCL(sigmoid activation, jeden neuron) -> attention output

α = Attend(Es, E\*q) // speech, repeated query embeddings  
 α = { α1, …, αT/2 }, αt e (0, 1) – prawdopodobieństwo wystąpienia KW w kroku t

ENERGY SCORER

Training processed is divided into three parts, speech encoder-decoder, query encoder-decoder, attention mechanism and energy scorer. Namely, we train them separately using the cross-entropy loss function and Adam optimiser [16] with a learning rate of 0.001.

END-TO-END KEYWORD SEARCH  
based on attention and Energy scorer

E2E avoids speech recognition (out-of-vocabulary queries),  
easier to train  
4 parts: speech encoder-decoder, query encoded-decoder,  
attention mechansm, energy scorer

Encoder-decoder returns character strings;  
attention mechanism recognizes keywords;  
Energy scorer makes the final decision

Na wejściu wektory różnej długości (time dimension)  
speech and query encoders -> attention mechanism  
-> set of attention weights -> energy scorer -> final result

SPEECH ENC-DEC  
input speech features -> 1D CNNs, 1D max pooling with a stride of 2 ->  
-> GRU-Encoder -> Speech Embeddings -> Attention GRU-Decoder ->  
-> Decoding results  
^ multilayer GRU-RNNs (Recursive NNs) with the same hyperparameters

E2E  
input speech -> speech encoder -> s. embeddings -> Attention mechanism  
 \-> Energy scorer  
input query -> query encoder -> q. embeddings -> Attention mechanism  
Attention weights + s. embeddings -> Energy scorer -> Decision

maxPooling with a stride of 2 => numer of time-steps will be halved after the processing  
\-> quicker to train the RNN

input X\* = {x1\*, x2\*, …, xT\*}  
after 1D CNNs and maxPooling: X = {x1, x2, …, xT/2}  
E­­S = Encoder(X) – speech embeddings  
\-> GRU-Encoder (multilayer GRU -> fully connected layer with ReLu activation)

ES -> GRU-Decoder with Global Attention (at each time step t, attention weights for every eS)  
\-> alfat(s) = align(ht, eS) = exp(score(ht,eS­)) / SIGMAS’ exp(score(ht,eS’))  
// s-th speech embedding, t-th decoding time step, ht – hidden state of the GRU-Dec at time t and score (a, b) = dot product of vector a and b

ct – context vector (weighted sum of the speech embeddings)  
ct = SIGMAS(alfat(s)eS)  
ot = FCL([htT, ctT]T) – fully connected layer with softmax activation <- concatenation of ct and ht  
\-> decision