Summary of "Language Independent End-to-End Architecture For Joint Language and Speech Recognition (2017)" by Watanabe, S.; Hori, T.; Hershey, J.R.

Author

1. Motivation / Goal

Recognize multiple languages at the same time

- Two tasks: identify language AND recognize speech (simultaneously)
- Use a single model for 10 languages (EN, JP, CH, DE, ES, FR, IT, NL, PT, RU)
- Find out if transfer learning between languages work
- End to end: Directly train sequence to sequence, no lexicon, phoneme pronounciation maps, or manual alignment

2. Related Work

- *Multilingual Speech Recognition With A Single End-To-End Model* (Shubham Toshniwal, Google) [1]
  - separate output for language id
  - only on 9 indian languages, hard to compare
- *Hybrid CTC/Attention Architecture for End-to-End Speech Recognition* (Watanabe et al. 2017) [2]
  - Same as this paper except only one language and more detailed

3. Model description

3.1. Input and output format decisions

3.1.1. Input

Apart from the model architecture, there are two main decisions to be made: How the audio should be input into the system, and what the output should look like.

For the input format, the authors chose the common method of extracting spectral features from the audio file, and chunking it into frames of e.g. 10ms. There is no description of what the exact features are. Because the architecture is very similar to the one the same authors mention in [2], we can assume the input structure to be similar. [2] mentions 40 features from the output of a filter bank, and two dimensions from simple pitch features. These are concatenated into a 42-dimensional input feature vector, which causes a potential issue described in sec. 5

3.1.2. Output

There are multiple formats that are viable for text output in a end-to-end neural network. Mainly we can either output words as a whole (using one-hot or word embeddings like word2vec [3]) or single characters. For speech recognition it makes sense to output single characters, since that way no fixed dictionary is needed and the network can learn to output words not seen before in the training data. To allow multilingual output, the authors propose to simply unify all the character sets (latin, cyrillic, CJK) to get a total of 5500 characters.

For the language identification output, the authors propose adding a special "language-tag" character that is prepended to the output, for example "`[EN]Hello`" or "

[CH]你好

". An alternative would be to add a separate one-hot encoded language id output as seen in [1].

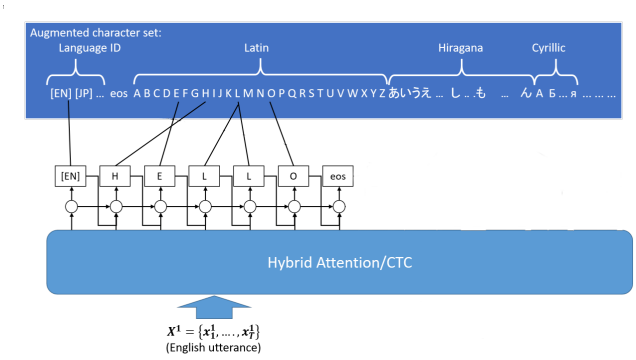

Figure 1: Input and output structure overview (from the paper)

3.2. Simple Model overview

1. Input: Basically a spectrogram as a 2D image
2. Encoder (CNN + LSTM)
3. Decoder
   (a) Soft Attention for each input frame to each output character
   (b) LSTM Layer
4. Output
   - N characters from union of all languages (one-hot / softmax)

3.3. Input Encoder

The authors use a image processing pipeline for the encoder. The input audio is thus formatted like an RGB image, with y being the feature index and x being time. The first channel is the spectral features described in sec. 3.1.1, and the second and third channel are delta and deltadelta features of the first channel.

The encoder consists of two parts, a convolution network and a recurrent network. The convolutional network is taken directly from the first six layers of the VGG Net architecture, as seen in fig. 2. Due to the two pooling layers, the dimensionality of the spectral input image is reduced 4 times in both the time and feature dimension. For simplicity, we will refer to t as 1/4th of the input time dimension, so the encoded state can be indexed by t.

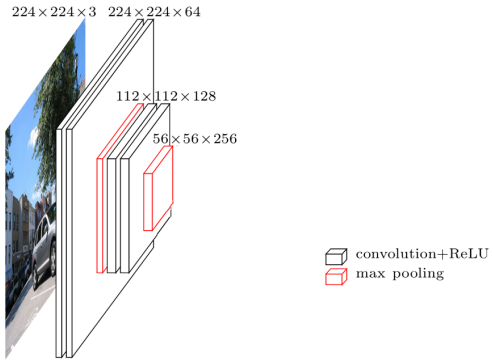

Figure 2: VGG Net for image processing - first 6 layers [4]

The convolved input is then fed into a bidirectional LSTM with 320 cells in each direction, which results in a encoded vector dimension (h_t) of 640 scalars per time step t.

This hidden state is then decoded with two separate decoders in parallel, one based on attention and one based on CTC.

3.4. Decoder A (Attention-based)

First the input sequence $\vec{x}_1, \ldots, \vec{x}_{4t}$ is encoded to $\vec{h}_1, \ldots, \vec{h}_t$ with the above described VGG+BLSTM. The goal is to get $l \leq t$ output characters $c_1, \ldots, c_l$. The soft attention weights a_{lt} are calculated based on three values: The attention on the same input for the previous output ($a_{(l-1)t}$), the current encoded state $\vec{h}_t$, and the previous hidden decoder state $\vec{q}_{l-1}$.

The encoded state is then combined weighted with the soft alignment to get the input of the decoder network $\vec{r}_l = \sum_t a_{lt}\vec{h}_t$. The decoder is another (unidirectional) LSTM layer, followed by a softmax layer:

$$c_l = \text{Softmax}(\text{FC}(\text{LSTM}(\vec{r}_l, \vec{q}_{l-1}, c_{l-1})))$$

Using this decoder without any additions is possible, but the authors argue that pure temporal attention is too flexible since it allows nonsensical alignments: Compared to machine translation, the word order can not change from input to output in speech recognition, it is strictly monotonic. Thus, the authors add a second decoder based on CTC described in the next section.

3.5. Decoder B (CTC-based)

The input and encoder are the same as before. After the encoder, a simple softmax layer per time step is added that directly converts the 640 outputs ($\vec{h}_t$ from the encoder into one of the N output characters. This results in one output character per timestep t, which is then reduced to a flexible number of output characters using the CTC loss function [5].

As a simple explanation, CTC works by adding a blank character `"="` to the output character set. For example, if we only allow the output HELLO our output set would be $\{H, E, L, O, -\}$. For inference, first all duplicate characters are removed and then all blank symbols. For example: `HHHH-EEEEEEEE-LL-LLL-\-\-\-OOOOOO` $\rightarrow$ `H-E-L-L-O` $\rightarrow$ `HELLO`. Note that the double l sequence is retained since there is a blank symbol between two blocks of `l` symbols. For training, we simply define all combinations of character duplications that result in the ground truth at inference time to be correct. The loss function is then the negative logarithm of the ground truth probability the network predicts. This probability can be efficiently computed using the Viterbi / forward-backward algorithm.

As opposed to the attention-based decoder, this method enforces the monotonic alignment of the output to the input.

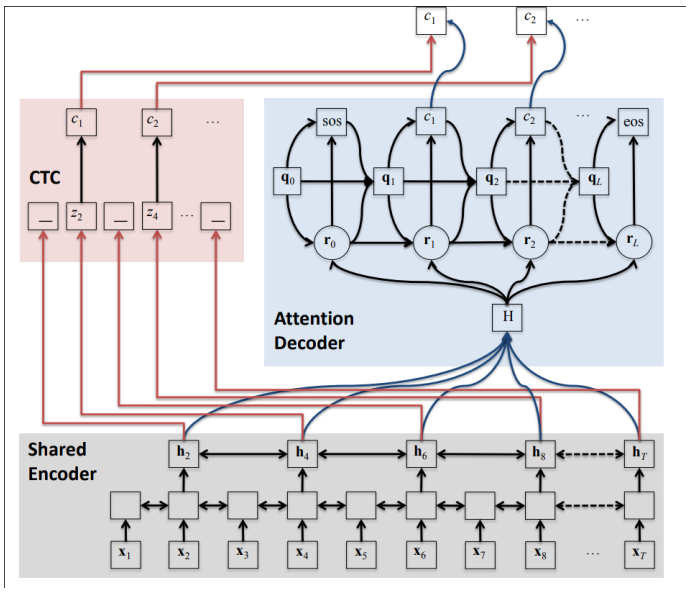

Figure 3: Hybrid CTC/attention-based end-to-end architecture

3.6. Language Model

With the model described so far, the neural network needs to implicitly learn a language model for all the output languages and store it together with the decoding task in the weights. The authors propose adding a separate, explicit language model called RNN-LM, that is trained only to model the distribution and sequences of the output characters, while ignoring the input speech. It is also based on a single LSTM layer that predicts the next character based on the previous one (together with its hidden state). It is trained separately, though it would be possible to jointly train it.

3.7. Final loss function

The final loss function is a simple linear combination of the loss functions of the attention decoder, the CTC decoder, and the language model. The two decoders are weighted equally ($\lambda = 0.5$), and the language model is weighted at 1/10 of the decoders together ($\gamma = 0.1$).

$$\mathcal{L}_{\text{MTL}} = \lambda \log p_{\text{ctc}}(C|X) \quad (1)$$
$$+ (1-\lambda) \log p_{\text{att}}(C|X) \quad (2)$$
$$+ \gamma \log p_{\text{rnn-lm}}(C) \quad (3)$$

The authors use the AdaDelta optimizer and train for 15 epochs. The inference is done via beam search on the attention output weighted by the above loss function.

4. Results

Fig. 4 shows the average character error rate for all languages for different experiments. Comparing the second and third columns shows that adding the convolutional network in front of the LSTM encoder improves performance by 7%. The fourth column shows that the RNN-LM improves performance, though not by much (3%). Adding data from three more languages (NL, RU and PT) increases the performance for the *other* 7 languages by 9%, which shows that transfer learning between the languages works. Note that the authors do not provide a significance analysis, but the data set is of all languages together is very large. The authors also do not provide a baseline or results with only one of both decoders.

	Language-dependent 4BLSTM	**7lang** 4BLSTM	**7lang** CNN-7BLSTM	**7lang** CNN-7BLSTM RNN-LM	**10lang** CNN-7BLSTM RNN-LM
Avg. 7 langs	22.7	20.3	18.9	18.3	**16.6**

Figure 4: Character Error Rates (abbrev.)

Fig. 5 shows that the language identification task has very good results. The only strong confusion is that Spanish is often (30%) mistaken for Italian, but not the other way around. The authors do not provide an interpretation for this.

		CH	EN	JP	DE	ES	FR	IT	NL	RU	PT
CH	train_dev	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	dev	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
EN	test_eval92	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	test_dev93	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
JP	eval1_jpn	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	eval2_jpn	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	eval3_jpn	0.0	0.0	99.9	0.0	0.0	0.0	0.1	0.0	0.0	0.0
DE	et_de	0.0	0.0	0.0	99.7	0.0	0.0	0.0	0.3	0.0	0.0
	dt_de	0.0	0.0	0.0	99.7	0.0	0.0	0.0	0.3	0.0	0.0
ES	dt_es	0.0	0.0	0.0	0.0	67.9	0.0	31.9	0.0	0.0	0.2
	et_es	0.0	0.0	0.0	0.1	91.1	0.0	8.4	0.1	0.0	0.2
FR	dt_fr	0.0	0.0	0.0	0.1	0.0	99.4	0.0	0.2	0.0	0.3
	et_fr	0.0	0.0	0.0	0.1	0.0	99.5	0.0	0.1	0.0	0.3
IT	dt_it	0.0	0.0	0.0	0.0	0.3	0.4	99.1	0.0	0.0	0.3
	et_it	0.0	0.0	0.0	0.0	0.4	0.4	98.3	0.2	0.1	0.7
NL	dt_nl	0.0	0.0	0.0	1.3	0.0	0.1	0.1	97.2	0.0	1.3
	et_nl	0.0	0.0	0.0	1.0	0.0	0.2	0.2	97.6	0.0	0.9
RU	dt_ru	0.2	0.0	0.0	0.0	0.2	0.6	0.5	0.0	97.9	0.8
	et_ru	0.0	0.0	0.0	0.2	0.2	0.3	4.3	0.0	94.7	0.3
PT	dt_pt	0.0	0.0	0.0	0.3	0.3	2.6	1.7	3.4	0.6	91.2
	et_pt	0.0	0.3	0.0	0.3	0.0	0.0	3.9	3.6	0.3	91.5

Figure 5: Language identification (LID) accuracies/error rates (%). The diagonal elements correspond to the LID accuracies while the offdiagonal elements correspond to the LID error rates

5. Potential problems / future work?

- Only fed with a single language utterance at a time
  - maybe we want to allow switching? (append utterances from different languages)
- Uniform random parameter initialization with $[-0.1, 0.1]$ seems statistically unsound? (use Xavier / Hu)
- Input feature convolution is weird
  - *[. . .] we used 40-dimensional filterbank features with 3-dimensional pitch features*
  - redundancy (delta, deltadelta)
- Unbalanced language sets (500h CH, 2.9h PR)
- Same latin characters are used for multiple languages, while others (RU, CH, JP) get their own character set
  - Try transliterating them to Latin?
- Does not work online (without complete input utterance)
  - Bidirectional LSTM in encoder
    - Could try one directional, but Language ID would completely break
    - aggregate limited number of future frames (e.g. add 500ms latency between input and output)
  - Attention does not work in realtime
  - CTC should work online

6. References

[1] S. Toshniwal *et al.*, "Multilingual Speech Recognition With A Single End-To-End Model," Nov. 2017.

[2] S. Watanabe, T. Hori, S. Kim, J. R. Hershey, and T. Hayashi, "Hybrid CTC/Attention Architecture for End-to-End Speech Recognition," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 8, pp. 1240–1253, Dec. 2017.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," Jan. 2013.

[4] K. Simonyan and A. Zisserman, "Very Deep Convolu-

tional Networks for Large-Scale Image Recognition," Sep. 2014.

[5] A. Graves, S. Fernández, and F. Gomez, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *In Proceedings of the International Conference on Machine Learning, ICML 2006*, 2006, pp. 369–376.