Summary of "Language Independent End-to-End Architecture For Joint Language and Speech Recognition (2017)" by Watanabe, S.; Hori, T.; Hershey, J.R. [1]

Author

1. Motivation and Goals

The given paper [1] is a follow-up to the authors' previous paper [2]. The goal is to extend the same end-to-end architecture using hybrid Attention and Connectionist Temporal Classification (CTC) to recognize speech in multiple languages. In addition, the authors want to explicitly identify the spoken language. The model is trained jointly for all of the ten languages (EN, JP, CH, DE, ES, FR, IT, NL, PT, RU) in an end-to-end fashion, i.e. it is directly trained from input speech sequence to output text sequence, with no manual alignment, no lexicon, and no phoneme pronounciation maps. The model is fully shared between the languages. The results should show whether transfer learning between languages works by checking whether performance for one languages increases when adding data from a different language.

2. Model description

2.1. Input and output format decisions

2.1.1. 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 three dimensions from simple pitch features. These are simply concatenated into a 43-dimensional input feature vector, which causes a potential issue described in sec. 4.

2.1.2. 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 "`[ES]Hola`". An alternative would be to add a separate one-hot encoded language id output as seen in [4].

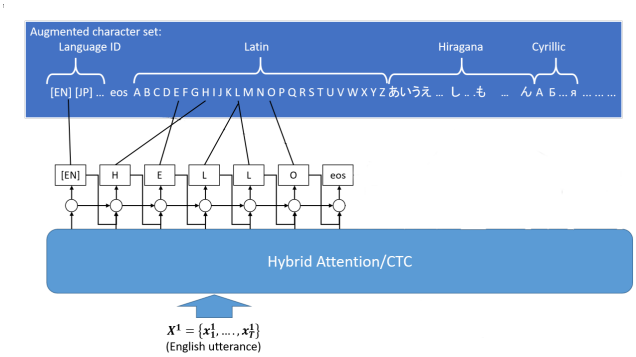

Figure 1: Input and output structure overview (from the paper)

2.2. Base Model overview

The authors chose a encoder-decoder architecture, with a CNN+LSTM-based encoder and two parallel decoders, one using Soft Attention, one using Connectionist temporal classification (CTC).

2.3. Input Encoder

The authors use an image processing pipeline for the encoder. The input audio is thus formatted like an RGB image, with the y coordinate being the feature index and x being time. The first "color" channel is the spectral features described in sec. 2.1.1, and the second and third channel are delta and deltadelta features of the first channel over time.

The encoder consists of two parts, a convolutional network and a recurrent network. The convolutional network is taken directly from the first six layers of the VGG Net architecture [5], 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.

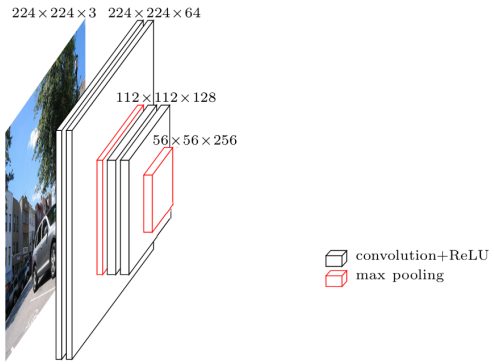

Figure 2: Original VGG Net for image processing - first 6 layers [5]. For our use, the input dimension is $4t \times 43 \times 3$.

The convolved input is then fed into a bidirectional LSTM with 320 cells in each direction, which results in a encoded vector dimension ($\vec{h}_t$) 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.

2.4. 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 VGG+BLSTM described above. 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 alone 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 as described in the next section.

2.5. 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 from the encoder $\vec{h}_t$ 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 [6].

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. This results in a unambiguous mapping from the CTC output to the corresponding text output. This means that the neural network is able learn the correct alignment by utilizing character duplications and the blank symbol.

The loss function for this decoder is the negative logarithm of the ground truth probability the network predicts. This probability can be efficiently computed using the Viterbi / forward-backward algorithm [6].

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

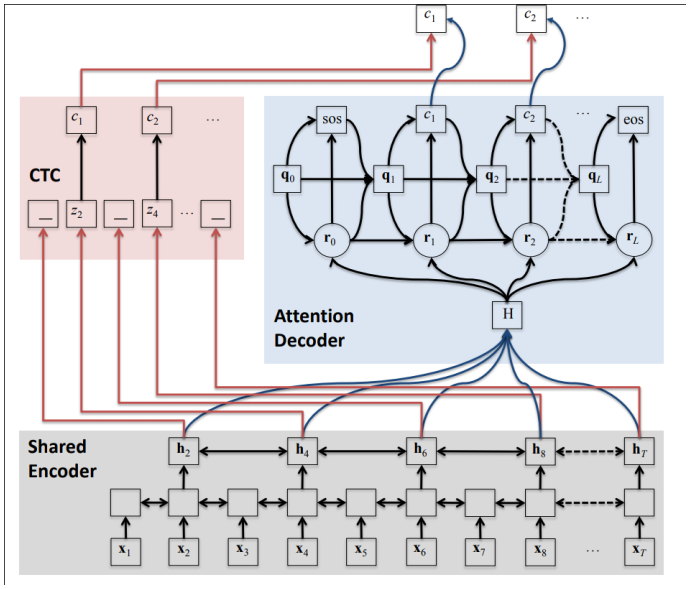

Figure 3: Hybrid CTC/attention-based end-to-end architecture

2.6. 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 parameters for the decoding task in the weights of the neural network. 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 based on a single unidirectional LSTM layer that predicts the next character based on the previous one (together with its hidden state). It is trained separately, though the authors mention that it would be possible to jointly train it.

2.7. 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 ($\gamma = 0.1$).

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

$p(C|X)$ denotes the probability of a character sequence C depending on an input sequence X, while $p(C)$ denotes the apriori probability of a character sequence C. 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. They do not describe any experiments with different optimization methods or with different values for γ and λ.

3. 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* seven languages by 9%, which shows that transfer learning between the languages works very well. Note that the authors do not provide a significance analysis, but the data set 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 (abbreviated)

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 of this phenomenon.

		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

4. Potential Problems and Future Work

There are some potential problems with the given paper and several interesting follow-up questions.

One minor point is that the authors use uniform random parameter initialization with $[-0.1, 0.1]$, which seems arbitrarily chosen. Considering how the initialization is often important for convergence speed as well as the final performance, it might make sense to use a statistically sound initialization method like Xavier or Hu initialization instead.

A larger issue that is harder to rectify is that the input data sets are very unbalanced, e.g. there is 500 hours of Chinese training data but only 2.9 hours of Portugese training data. To balance this, one could duplicate the data for languages with little available data, but considering this would cause the Portugese data to be duplicated 170 times, this might cause the model to overfit on those sequences. Another potential balancing issues is that some languages use the same character sets (e.g. latin characters), while others use their own character set (e.g. Cyrillic for Russian, Hanzi for Chinese, etc.). As a result, the model might learn to group the languages by their character sets as an unwanted side-effect – latin languages can have very different pronounciations of different characters. This could be solved by either giving each language its own character set (duplicating the latin characters for each latin language), or by transliterating every language to a single character set, assuming such a bidirectional mapping exists (e.g. Hanyu Pinyin romanization).

The model is only fed with a single language utterance at a time. This means that for a multi-lingual recognition task, the input data would need to be split at the language boundaries beforehand, making the model unusable for a dialog where two or more languages are spoken. Since the model already has to identify the language, it would be interesting to train

the model using a concatenation of two or more utterances in different languages, which would force the model to learn language switching within one input sequence. This would also implicitly solve the problem of splitting input audio by language boundaries.

The input feature convolution is weird: The authors concatenate a 40-dimensional filterbank with three-dimensional pitch features, which they then feed into the convolutional VGG-architecture. The convolutional architecture assumes shift invariance, i.e. it forces the same parameters to be used to interpret the pitch features and the filterbank features. In addition, the pitch and filterbank features are assumed to be spatially correlated at the boundary where they are concatenated, even though they are completely independent. To fix this, the two input feature kinds would need to get separate instantiations of the convolutional architecture and then concatenated for example using late fusion [7]. In addition, there is a lot of redundancy in the inputs due to the addition of delta and deltadelta features. These features are already implicitly present in the first VGG layer due to the convolution in the time dimension.

The final interesting issue is that the model does not work online, i.e. without aquiring the complete input utterance beforehand. To enable online transcription, several changes would need to be made to the model. For one, the bidirectional LSTM in the encoder needs the complete utterance by design. It could be changed to be unidirectional, but that would cause the Language ID output to completely break since in this model it is output first, but definitely needs information from future frames to be accurate. Another issue is that the soft attention mechanism uses a weighted version of all hidden states, meaning it also depends on the complete input sequence at once. One idea to solve these issues is to aggregate a limited number of future frames and add an artifical latency of e.g. 500ms between input and output. This way, the model gets half a second of future information to decide on the language identification and recognized text for the current speech. Since the speech recognition task is monotone as discussed earlier, this might work ok.

5. References

[1] S. Watanabe, T. Hori, and J. R. Hershey, "Language independent end-to-end architecture for joint language identification and speech recognition," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2017, pp. 265–271.

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

[5] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Sep. 2014.

[6] 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.

[7] C. G. M. Snoek, "Early versus late fusion in semantic video analysis," in *In ACM Multimedia*, 2005, pp. 399–402.